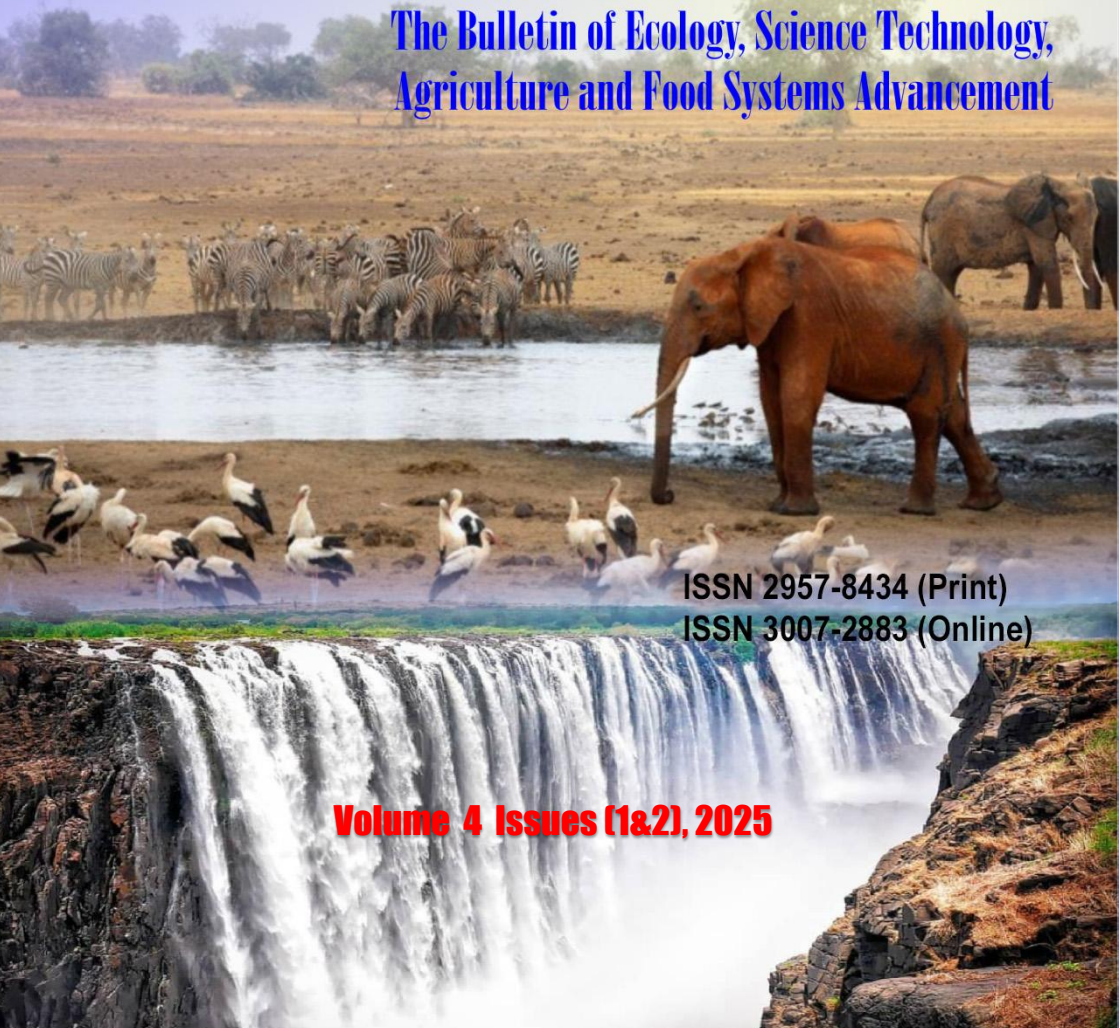
ZEGU
ZIMBABWE
EZEKIEL GUTI
UNIVERSITY
Committed to Excellence

Oikosis

The Zimbabwe Ezekiel Guti University
The Bulletin of Ecology, Science Technology, Agriculture and Food Systems Advancement



ISSN 2957-8434 (Print)
ISSN 3007-2883 (Online)

Volume 4 Issues (1&2), 2025

Published by the Zimbabwe Ezekiel Guti University Press
Stand No. 1901 Barrassie Rd,
Off Shamva Road
Box 350
Bindura, Zimbabwe

Typeset by Divine Graphics
Printed by Divine Graphics

SUBSCRIPTION AND RATES
Zimbabwe Ezekiel Guti University Press Office
Stand No. 1901 Barrassie Rd,
Off Shamva Road
Box 350
Bindura, Zimbabwe
Telephone: ++263 8 677 006 136 | +263 779 279 912
E-mail: zegupress@admin.uz.ac.zw
http://www.zegu.ac.zw/press

About the Journal

JOURNAL PURPOSE

The purpose of *the Oikos - The Zimbabwe Ezekiel Guti University Bulletin of Ecology, Science Technology, Agriculture and Food Systems Review and Advancement* is to provide a forum for scientific and technological solutions based on a systems approach and thinking as the bedrock of intervention.

CONTRIBUTION AND READERSHIP

Natural scientists, engineering experts, technologists, multidisciplinary teams are encouraged.

JOURNAL SPECIFICATIONS

Oikos - The Zimbabwe Ezekiel Guti University Bulletin of Ecology, Science Technology, Agriculture and Food Systems Review and Advancement

ISSN 2957-8434(Print)
ISSN 3007-2883(Online)

SCOPE AND FOCUS

The journal is a forum for the discussion of ideas, scholarly opinions and case studies of natural and physical science with a high proclivity to multidisciplinary approaches. The journal is produced bi-annually.

Guidelines for Authors for the *Oikos* Journal

Articles must be original contributions, not previously published and should not be under consideration for publishing elsewhere.

Manuscript Submission: Articles submitted to the *Oikos - The Zimbabwe Ezekiel Guti University Bulletin of Ecology, Science Technology, Agriculture and Food Systems Review and Advancement* are reviewed using the double-blind peer review system. The author's name(s) must not be included in the main text or running heads and footers.

A total number of words: 5000-7000 words and set in 12-point font size with 1.5 line spacing.
Language: British/UK English
Title: must capture the gist and scope of the article
Names of authors: beginning with the first name and ending with the surname
Affiliation of authors: must be footnoted, showing the department and institution or organisation.
Abstract: must be 200 words
Keywords: must be five or six containing words that are not in the title
Body: Where there are four authors or more, use *et al.*
Italicise *et al., ibid.,* words that are not English, not names of people or organisations, etc. When using more than one citation confirming the same point, state the point and bracket them in one bracket and in ascending order of dates and alphabetically separated by semi-colon e.g. (Falkenmark, 1989, 1990; Reddy, 2002; Dagdeviren and Robertson, 2011; Jacobsen *et al.,* 2012).

Referencing Style: Please follow the Harvard referencing style in that:
— In-text, citations should state the author, date and sometimes the page numbers.
— The reference list, centred alphabetically, must include all the works cited in the article.

In the reference list, use the following guidelines, religiously:

Source from a Journal

Anim, D.O and Ofori-Asenso, R (2020). Water Scarcity and COVID-19 in Sub-Saharan Africa. *The Journal of Infection*, *81*(2), 108-09.

Banana, E, Chitekwe-Biti, B and Walnycki, A (2015). Co-Producing Inclusive City-Wide Sanitation Strategies: Lessons from Chinhoyi, Zimbabwe. *Environment and Urbanisation*, *27*(1), 35-54.

Neal, M.J. (2020). COVID-19 and Water Resources Management: Reframing Our Priorities as a Water Sector. *Water International*, *45*(5), 435-440.

Source from an Online Link

Armitage, N, Fisher-Jeffes L, Carden K, Winter K, *et al. (*2014). Water Research Commission: Water-sensitive Urban Design (WSUD) for South Africa: Framework and Guidelines. Available online: https://www.greencape.co.za/assets/Water-Sector-Desk-Content/WRC-Water-sensitive-urban-design-WSUD-for-South-Africa-framework-and-guidelines-2014.pdf. Accessed on 23 July 2020.

Source from a Published Book

Max-Neef, M. (1991). Human Scale Development: Concepts, Applications and Further Reflections, London: Apex Press.

Source from a Government Department (Reports or Plans)

National Water Commission (2004). Intergovernmental Agreement on a National Water Initiative. Commonwealth of Australia and the Governments of New South Wales, Victoria, Queensland, South Australia, the Australian Capital Territory and the Northern Territory. Available online: https://www.pc.gov.au/inquiries/completed/water-reform/national-water-initiative-agreement-2004.pdf. Accessed on 27 June 2020.

The source from an online Newspaper article

Herald, The (2020). Harare City Could Have Used Lockdown to Clean Mbare Market. *The*

Herald, 14 April 2020. Available online: https://www.herald.co.zw/harare-city-could-have-used-lockdown-to-clean-mbare-market/. Accessed on 24 June 2020.

ENHANCING PROGRAMMING PROFICIENCY, EVALUATING THE IMPACT OF AI-POWERED CODE ASSISTANT TOOLS ON LEARNING OUTCOMES

TRUST MHLANGANISO[1] AND MAKOTA JUSTIN[2]

Abstract

There always exists a naiver, tinier approach to writing good code and, likewise, a longer and more comprehensive way. Imagine a tool that takes advantage of deep machine learning algorithms to fish the most appropriate code and put it in a drop-down menu only for a programmer to select. In recent years, the use of AI-powered programming tools has grown in leaps and bounds and gained much attention. Whereas AI has been around for roughly 30 years, it is still uncertain for educators on how to make instructive advantage of it on a larger scale and how it can essentially have a profound effect on teaching and learning in tertiary education. This article investigates the effectiveness of AI-powered code assistant tools in learning programming. For the research, two leading coding platforms were picked (eclipse and VS code) and a selected AI-powered code assistant tool (Tabnine) was installed. After training the research participants, an experiment was carried out, whereby all 40 participants were given two tests, one on AI-assistant enabled coding platform and the other on non-AI-assistant enabled coding platform. Results indicate that AI-coding tools significantly increase students' coding efficiency and general motivation to code. Results also show that AI code assistant tools do not affect participant's code comprehension. From the results found it is recommended that AI code

[1] Bindura University of Science Education, Department of Computer Science, Zimbabwe, https://orcid.org/0009-0006-9748-125, tmhlanganiso2001@gmail.com

[2] Zimbabwe Ezekiel Guti University, Department of Data Science and Computer Technology, Zimbabwe https://orcid.org/0009-0006-8648-115X , jmakota@gmail.com

assistant tools must be incorporated to aid students in developing a positive attitude towards programming and also to improve their coding efficiency.

Keywords: AI-powered, learning, programming, AI-assistant enabled coding platform, non-AI-assistant enabled coding platform

Introduction

In the global world of Computer Science (CS), it is well documented that learning to program poses a challenge to many students (Khomokhoana and Nel, 2020a). Therefore, a lot of strides have so far been undertaken to try and assist the CS student. Teachers have always been inundated with trying to come up with good ideas to make students comprehend programming languages and yet not every idea ever thought of is a good idea (Ì and Ì, 2024). Therefore, in this world, to decode how good an idea is dependent on experiments (Khomokhoana and Nel, 2020b). The code editors in use today offer a code assistant in the form of a floating menu coupled with contextually applicable variables, methods and other code segments. As programmers navigate and make choices from menu options, they are able to evade countless collective syntax and logic errors and also eradicate redundant keystrokes. Ongoing refinements to the code editors have seen the birth of AI-powered code assistant tools in the market today.

In this study, the effectiveness of AI-powered code assistant tools in learning computer programming using popular modern IDEs, is investigated. If computer programming is utterly challenging, then there is an inevitable prerequisite to make it more intriguing and attention-grabbing (Zhou *et al*., 2022). The key medicine to this scenario is to expose the learner to the complexities of AI-powered code assistant tools.

This research is based on the following objectives.

1. To measure the impact of GitHub Copilot on the coding efficiency.
2. To evaluate the effect of using an AI code assistant on code comprehension.

3. To determine the impact of AI-powered code assistants on student motivation.

To address the objectives highlighted above, trending AI-powered code assistant tools are first analysed. In the next segment, an overview of other work related to AI-powered coding is reviewed. In the coming sections, trending coding platforms capable of handling AI-powered code assistant tools to use for the experiment, are selected. The next section attempts to interpret the results of the study and provide conclusion, alongside recommendations.

AI-powered code assistant tools

An AI-powered code assistant tool can be described as a tool that is programmed to mimic how humans came up with decisions. It must have the capability of providing what most people refer to as an 'artificial thought' that empowers it to solve issues on its own (Saini *et al*., 2025). Below are some of the characteristics of an AI-powered code assistant tool.

It must do things proactively

This involves automation. The tool must be able to do things on its own without requiring a human to press a button or prompt it to do something (Zhou *et al*., 2022).

The AI tool is supportive

An AI-powered tool should be able to supplement work done by its users via intelligent suggestions and recommendations (Alanazi *et al*., 2025).

AI is smart

Any AI-powered code assistant tool is programmed to imitate how humans arrive at decisions. The tool must provide an 'artificial thought' that allows it to solve challenges without human intervention (Tüfekci and Köse, 2013).

AI continues to learn

Though the tool is already smart, it must continue to acquire information from its users such that the longer a person uses it, the more it becomes proficient (Ciniselli *et al*., 2022)

Types of AI-powered code assistant tools

TabNine

TabNine is an artificial intelligence-based code auto-completer that assists developers to write codes quicker and is owned by Codota (Bruch *et al*., 2018) . It is unique as compared to other code auto completion tools in that the team that developed it incorporated a deep learning model. TabNine is built on GPT-2, open-source artificial intelligence architecture, a product of OpenAI in February 2019 which uses the Transformer network architecture. At the writing of this article, GPT-2 comprised 1.5 billion parameters that were trained on a dataset of 8 million web pages with a simple objective of predicting the next word, given all of the previous words within some text (Warth, 2008).
GPT-2 generates synthetic text samples in response to the model being primed with an arbitrary input. It familiarises with the format as well as content of the conditioning text. This allows the user to generate realistic and coherent continuations about a topic of their choosing Code auto completion feature is handled by an external software which TabNine communicates with by means of the Language Server Protocol (Warth, 2008). TabNine also contains default install scripts that support several common language servers and is fully configurable, for those who wish to use a different language server or add code auto completion for a new language (Alanazi *et al*., 2025). TabNine is available as a free plugin for major platforms like:

- IntelliJ PyCharm
- VS Code
- Sublime
- IntelliJ PhpStorm
- VIM
- Atom etc

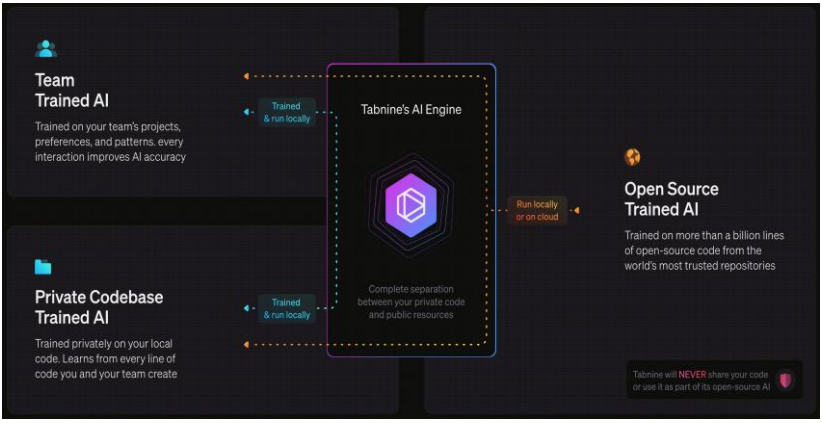


Figure 1: A Tabnine's AI engine.
Source: (https://www.tabnine.com)

The TabNine's AI engine in Figure 1 is grounded on three fundamental principles, that is, better collaboration, better privacy protection and better code completion. For coding collaboration, there are new features that now incorporate the growing suite of tools for teams, that is, naming your team, invitation of team members and managing your account, all from your My TabNine profile (Munisamy, 2024). The more the team members invited and added, the faster the TabNine's Team Trained AI. This also implies that Private Codebase Trained AI will have the opportunity to learn your team's projects, preferences and patterns and suggesting even more accurate code completions (*ibid*.). In enduring privacy and compliancy, the three TabNine's AI code completion models can be executed locally and under no circumstances share your code. Lastly as an AI assistant, the tool provides facilities such as IntelliSense, IntelliCode, autocompletion, AI-assisted code completion, AI code snippets, code suggestion, code prediction, code hinting and content assist (*i id*.).

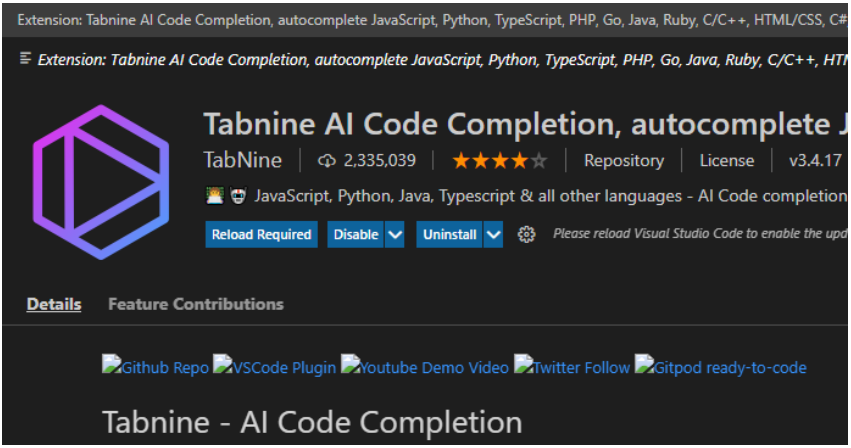


Figure 2: Icon of TabNine plugin on VS Code platform

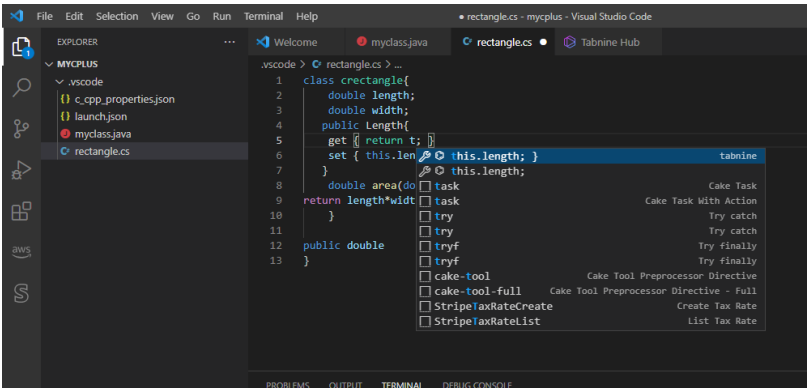


Figure 3: Top two options generated by TabNine plugin AI-code assistant Tool on VS Code

Kite

Kite is an AI driven coding assistant that offers massive editor integration facilities, giving the programmer an opportunity to work seamlessly on the same screen (Liang, 2025). It is freely available and helps programmers to write their code faster in Python. Kite's auto Completions feature takes advantage of deep learning to offer key

context-based code completions in real time. It is installed as a plugin on your IDE. Kite is able to coach its machine learning models using thousands of freely open code sources from highly rated developers in the world (Alanazi *et al*., 2025). In addition to the above. Kite is also capable of foretelling quite a lot of "words" of code at a time and is built on the most sophisticated AI engine available for modelling code (*ibid*.). At the time of writing this article, Kite was capable of supporting the following platforms:

- VS Code,
- IntelliJ Platform
- Vim
- Atom
- Sublime Text etc

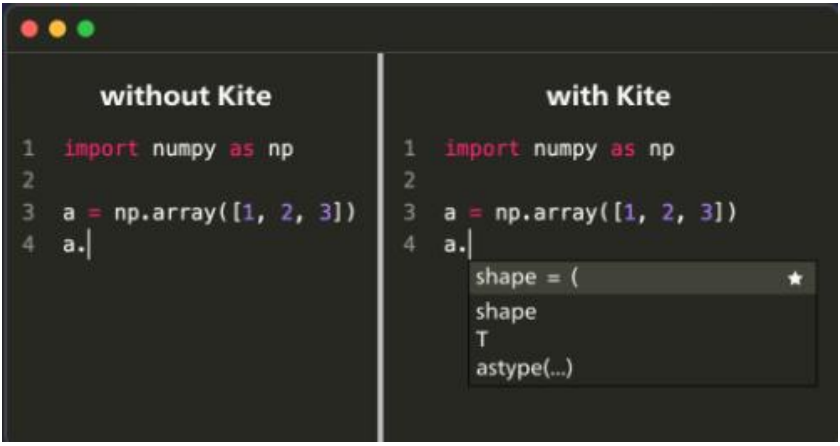


Figure 4: Comparison of options generated on IDE without Kite plugin AI-Tool and another with Kite plugin AI-code assistant Tool

Copilot

This is an AI-based programming tool cooperatively constructed with GitHub and is built on top of GPT-3 architecture, OpenAI's prestigious language model. Copilot is capable of offering relevant suggestions as you do the coding. Copilot's website describes it as an "AI pair programmer" that suggests "whole lines or entire functions right inside

your editor" (*ibid.*). At times, the user may just type a function signature or description and Copilot will generate an entire block of code.

Underlying Copilot is a deep learning model called Codex, which is essentially a distinct version of GPT-3 tailor-made for programming tasks (Choi, 2025). When in operation, the tool simulates GPT-3 by accepting a prompt as input and produces a categorisation of bytes as output. Copilot is perceived as a simple autocomplete coding tool that is additionally context aware than other code assistants (*ibid.*). Matt Shumer, co-founder and CEO of OthersideAI (*ibid.*) told TechTalks that: *"If you know a bit about what you're asking Copilot to code for you and you have enough experience to clean up the code and fix the errors that it introduces, it can be very useful and save you time,".*

At the time of writing this article, the AI code assistant tool was then supported by Visual Studio Code and platforms powered by a VS Code backend such as GitHub's Code spaces. Copilot can work well with several languages, especially Python, JavaScript, Typescript, Ruby and Go (Alanazi *et al*., 2025).

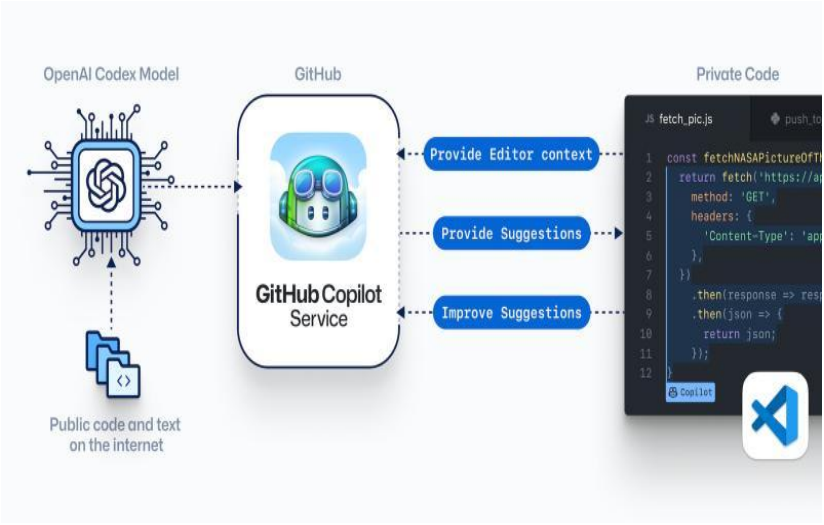


Figure 5: Copilot Engine

Figure 6: Highlighted code generated by Copilot when user simply types function declaration extracted from (https://copilot.github.com/)

Overview of AI powered programming tools

To date, number of papers seem to delve into code assistant tools that are not powered by AI (Saini *et al*., 2025). A research done by Bruch *et al.* (2009) (Ì and Ì, 2024), reveals that auto-completion features provided by current IDE's are grounded entirely on static type system provided by the programming language. Consequently, most of the predictive texts on the IDE are inappropriate for that specific working context. Similarly, these predictive texts are arranged alphabetically instead of by their relevance in a certain domain.

Bruch *et al.* (2009) carried research in which they propose smart code assistant tools that are capable of learning from existing code storehouses. They investigate three such systems, each by means of the information contained in storehouses in a different way. They gone on to execute a large-scale quantitative assessment of smart code assistant tools, incorporating the top performing one into Eclipse and evaluation (Alanazi *et al*., 2025). The result of their experiments suggests that smart code assistant tools which are able to learn from patterns, significantly outstrip conventional code completion tools on the basis of the relevance of their suggestions, thereby offering notable potential towards enriching programmers' productivity.

Aslıhan and Utku (2013) introduced an Artificial Intelligence based software system, developed to enhance quality of tutoring computer programming courses in universities. They express that Artificial Intelligence-based software systems are one of the most effective learning tools that can be harnessed to reduce difficulties and afford more effective learning experience for learners. It can be noted that computer programming learners, for the first time, can also find it difficult to understand algorithmic thinking as well as other principles of computer programming, hence the need for an AI-intelligent assistance tool. Aslıhan and Utku (*ibid,*) introduced some kind of an Artificial Intelligence based software system whereby students can take some exercises using a user-friendly interface and teachers can create new C Programming-based tasks by using the management interface. In each task, the teacher can state the problem text and develop what would be the correct solution to that problem in the same way as a student would do.

Evaluation instrument of the software system is based on a domain prepared according to the expert knowledge and domain expert knowledge of the software system with room for some adjustments. The software system introduced in this study has been intended for C Programming and the results reveal that the system ensures a successful approach on teaching computer programming generally and C programming specifically. The limitation is that the study concentrated on one language which is procedural-oriented. Therefore, this study seeks to extend the research by also including trending Object-oriented programming languages and powerful AI based programming tools impacting the world during the writing of this article.

Svyatkovskiy *et al*. (2019) suggest a new end-to-end approach for AI-assisted code completion termed Pythia. It is capable of producing ordered lists of methods as well as API suggestions which can be harnessed by programmers when writing code (Bruch *et al*., 2018). Their tool is currently installed as a component of Intellicode extension available in Visual Studio Code. Pythia is believed to make use of trending across-the-board deep learning models that were trained on various code contexts pulled out from abstract syntax trees. It is set to

operate at a high throughput, foretelling the top most matching code completions at a rate of 100ms. The outcome from their evaluation on a large dataset of 15.8 million method calls obtained from practical source code, demonstrated that their best model achieves 92% which is top-5 accuracy, meaning that they were able to beat simpler baselines. They triumphed and wrote a number of real-world problems of training deep neural networks and hyperparameter alteration on HPC clusters and model deployment on lightweight client devices to foretell the top matching code suggestions when coding (Tüfekci and Köse, 2013). However, for future, they recommend that advanced deep learning methods must be traversed with emphasis on programming languages, instead of Python and also to explore more sophisticated code completion situations (Alanazi *et al*., 2025). It is in this light that the study dug deeper into these AI-assisted coding tools.

In the last 30 years, Al has grown from research laboratory prototypes to become a key element in many areas of high-tech development such as robotics (Profile, 2016). Consequently, AI techniques have also been applied to education through development of intelligent tutoring systems (ITS). Leddo and Garg (2021) express that since the advent of intelligent tutoring systems; there has been an ever-increasing desire to see if instructive software that uses AI to mimic human instructors can eventually result in improvements in educational achievement. Consequently, Tüfekci and Köse (2013) carried a research in which they found out that students using their AI-based ITS software outperformed, on average, those using Khan Academy's software by 80% and those using Pearson Education's electronic textbooks by 300%.

Milan, *et al.* (), research developed an Interactive Programming Assistance tool (iPAT) to help students in solving introductory programming problems and help instructors in conducting programming lab sessions effectively. The tool permitted students to carry out programming exercises and receive collaborative guidance in getting their programmes to compile and run. However, the tool was limited in that it assists students and instructors in solving their practical lab session problems, which comes only with features such as error

handling, remote access, handling PC inventory and a solution archive to solve common errors only in C# programming.

Concerning this study, none of the articles were found to deal with an investigation into the effectiveness of AI based programming tools in the learning of programming languages today and this study is a valuable addition to the existing research.

Methods /Procedure

The study adopted the Plowright's Frameworks for an Integrated Methodology (FraIM). In this model, the research largely concentrated on collecting narrative and numeric data using observations, questionnaires as well as analysis of written code (Figure 7).

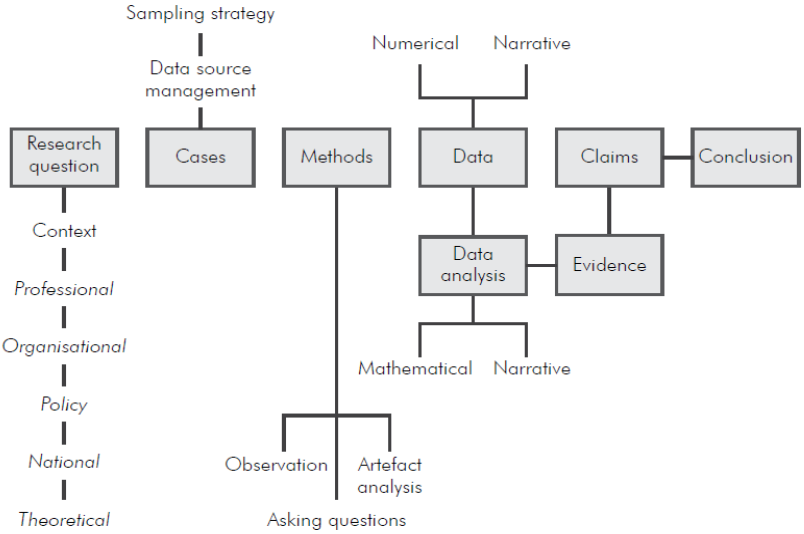


Figure 7: Plowright's Frameworks for an Integrated Methodology

This research was conducted at a Zimbabwe higher education institution. The research group consisted of final-year undergraduate computer science students. In their first study year, these students take

Computer Science modules that focus on building foundational knowledge regarding software development, network engineering and hardware engineering. In their second year, the students will be exposed to more knowledge in these disciplines, followed by a third year of internship. Fourth year students were chosen because they would have explored and acquired vast knowledge during attachment.
The empirical part of the study comprised the following activities. Activity 1: A popular AI code auto-completion tool was identified and installed. Activity 2: Students were exposed to AI-enabled IDEs and trained rigorously and followed by practical based assessments and observations. Activity 3: Basing on student's performance, interviews were carried out on chosen students.

Activity1: Choosing and Installing AI tools

The market today is characterised with vast AI code assistant tools. In order to choose the best AI tool, several factors such as portability and availability were considered. The research needed something that could easily fit in IDEs that students were familiar with in order to minimise complexities associated with learning features of new IDEs. In this regard, the popular IDEs and text editors found in their lab were VS code, eclipse, NetBeans, Microsoft Visual studio, IntelliJ IDEA., Code Blocks, etc. Using results of IDEs evaluation, the study chose to use eclipse IDE and VS code Interpreter. Figure 8 shows Tabnine tool plugin about to be installed in the eclipse IDE from the eclipse marketplace. After installation, the plugin is automatically enabled and is highly context sensitive as demonstrated by Figure 8 below whereby tabnine is assisting the coder to instantiate a Calculator class in Java.

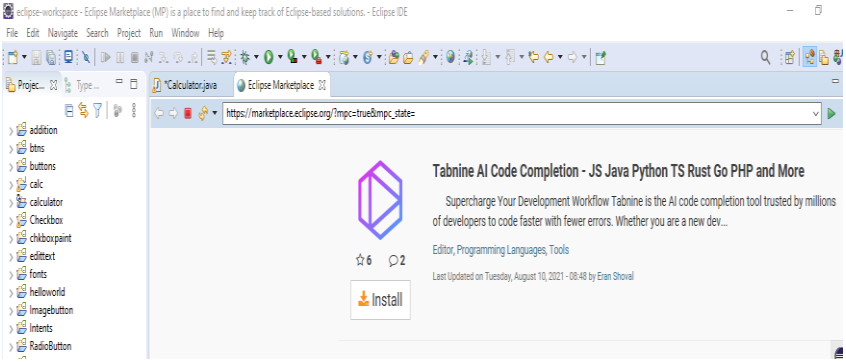


Figure 8: Tabnine tool under installation on eclipse IDE

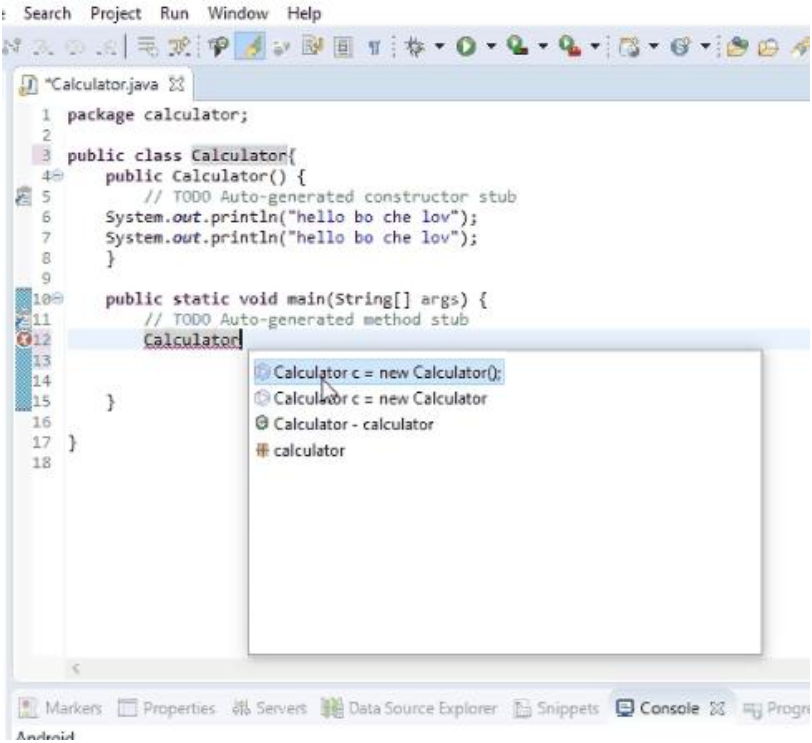


Figure 9: Instantiating a calculator class example using tabnine AI-code assistant tool in Java

Tabnine was also installed on VS code (figure 1) and it was tested using a simple shape class with a parameterised constructor that accepts base and height integer arguments to calculate area of a Triangle using C# (Figure 10). Two context-based lines were generated by Tabnine and are very relevant to the situation being tackled.

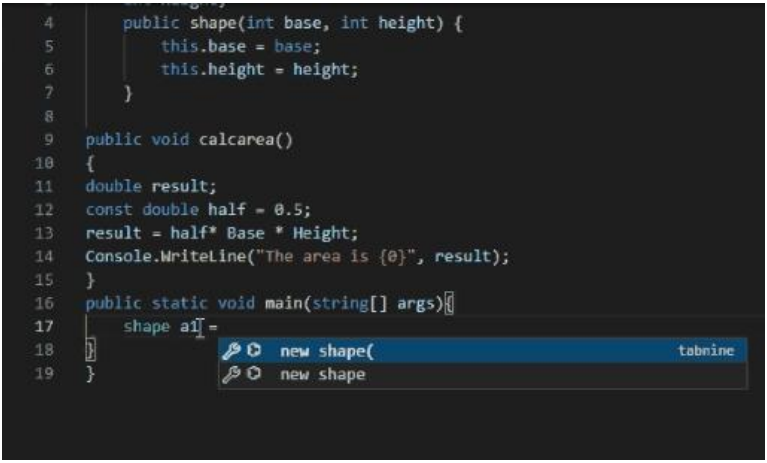


Figure 10: Instantiating an animal class using TabNine AI-code assistant tool in c#

Activity 2: Data collection

For this research, a total of 40 computer science students were chosen and split into two groups of 20 each. The first twenty (Group A) was rigorously trained on Tabnine embedded on eclipse IDE using Java language and the other 20 (group G) was also trained on Tabnine embedded on VS code interpreter using c# language. After coaching, the students were administered two practical-based lab tests which were written 1. on compiler with an AI-enabled tool; and 2. on a non-AI-enabled compiler. The study limited the test scope to focus on OOP questions that can be implemented either using Java or C#. For the tests, screen recording software during tests was activated and all work saved for marking. The research also got the opportunity to observe students during test writing and noted key features explained later in data analysis phase. The participants saved work and recorded work were the primary source of data for this activity. After grouping of the

artefacts, the performance data for each participant was then captured into SPSS package and descriptive statistics was adopted to attempt to answer the following research objective:
To measure the impact of GitHub Copilot on the coding efficiency

Activity 3: Interviews
A summative assessment based on the practical tests administered provided the basis for the choosing of students to interview. In this regard, 10 students from Group A and another 10from Group B were interviewed. Interviews were set to provide solutions for the last objective of the research, that is:

1. *To assess the effects of AI-powered code assistant tools on students' motivation*

Proper interview questions were drafted and time slots of 20 minutes were scheduled for each participant. However, the participants were advised that they could take as much time as they needed to express themselves. The events of each session were recorded with approval from the interviewees.

Results and Analysis
In this study, all 40 fourth-year computer science students (100%) took part in two practical test assessments and a sample of 20 students participated in interviews and the results were tabulated using the IBM SPSS Statistical package.

To address the first objective, to assess the effect of AI-powered code assistant tools on students' coding efficiency, the study administered two practical tests for both Group A and Group B. The first test was written using AI-activated code assistant platform and the second using non-AI-activated code assistant platform. Efficiency, being the ability to avoid wasting materials, energy, efforts, money and time when executing a particular activity, the research assessed efficiency basing on an evaluation of the number of students who managed to finish the examination in stipulated time, at the same time attaining at least a pass. Since two examinations were written by every participant, results were analysed on that basis. Table 1 shows collated results of the exam

written on AI-Activated code assistant IDE and Table 2 depicts results of the exam written on non-AI-Activated code assistant IDEs.

Table 1: Results of AI-activated code assistant examination

		Final AI-IDE Exam Mark		Total
		>=60 and <75	>=75	
Did the student finish AI-Enabled Exam?	N	1	3	4
	Y	2	34	36
Total		**3**	**37**	**40**

Table 2: Results of Non-AI-activated code assistant examination

		Final Non-AI-IDE Exam Mark			Total
		>=50 and <60	>=60 and <75	>=75	
Did the student finish non-AI-Enabled Exam?	N	5	20	1	26
	Y	0	9	5	14
Total		**5**	**29**	**6**	**40**

From Tables 1 and 2, it is observed that all participants (40) managed to pass for both AI and non-AI-activated code assistant platforms.

However- a closer analysis shows that in the AI activated exam, 4 (10%) did not finish the exam, while in the non AI-activated IDEs, 26 (65%) did not finish their exam, implying that AI-coding tools help to improve typing speed and also save time for a student to think of the next token during coding, since the tool quickly avails such an option for the programmer to simply click on and concentrate on the next problem.

In order to assess the effect of AI-powered code assistant tools on students' code comprehension, the research used a comparison of test performance results administered on AI-enabaled IDEs versus non-AI-enabled IDEs and the results are interpreted in Tables 1 and 2. Of interest is the quality of passes. Results from the AI-activated IDE exam reveal that a total of 37 (93%) students had passes in the range of 75 to 100, while results from non-AI-activated IDEs show that a total of 6 (15%) had passes in the range 75 to 100. From these results, can be said that high quality marks were attained on AI-activated code assistant platforms not because students comprehend the code, but the tool is intelligent enough to pick up the coder's goals and consequently present complex patterns that envision in the students' thoughts, allowing it to be able to assist coding in a quite significant way. If participants comprehend coding, then they should be able to code even better when using non-AI-enabled code assistant platforms which is not a reflection of results (see Table 1 versus Table2.

To assess the effects of AI-powered code assistant tools on students' motivation, a sample of 10 students from Group A were interviewed and another 10 in Group B and the results are shown in Table 3.

Table 3: Level of motivation of students towards use of AI-code assistant tools.

		AI tools motivate me to code			**Total**
		Strongly agree	**Agree**	**Not sure**	
Group A or B	A	6	2	2	10
	B	7	2	1	10
Total		**13**	**4**	**3**	**20**

Results from Table 3 hint that 13(65%) of the participants strongly believed that AI-coding tools motivates them to love programming, whereas 4 (20%) were also in agreement and 3 (15%), quite a small percentage, indicated that they were not so sure. From the results in Table 3, it can be said that AI-powered code assistant tools play a significant role towards motivating a student to love to code.

From the observations, the study established that most students looked uneasy as they were writing the non-AI-enabled code assistant platform exam, whereas the same students, when exposed to the AI-enabled code assistant platform exam, looked quite confident and engaged. It was also observed that the AI coding plugins are generally not aware of what the programmer is trying to achieve and, therefore, at best, it can recommend mutual constructs like it has seen before, possibly directed by the structure of a method.

Conclusion

This study analysed the effectiveness of AI-powered code assistant tools in learning programming and a sample of 40 final-year computer science students was used. Two main compilers were chosen (eclipse and VS code) and Tabnine (an AI coding tool) was installed in order to try and influence students' coding efficiency, comprehension and motivation. After training them on the AI tool, an experiment was carried out whereby all 40 students were given two tests, one on AI-enabled IDE and the other on non-AI-enabled IDE and results were tabulated. Results indicate that AI-coding tools significantly increase students' coding efficiency and motivation to coding and that AI code assistant tools do not affect student's code comprehension. From the results, it is recommended that AI-code assistant tools must be incorporated to aid novice programmers to learn to write code faster and, with repeated exposure to such tools, they may end up comprehending code.

For future research, this study proposes an increase of the scope research to include more compilers and AI tools, especially the one recently developed by Microsoft termed as Copilot which, at the time of writing, was not yet available for experiments .There is also need to look

into the issue of copyrights, since under AI-assisted coding, one will be using other developer's codes extracted by algorithms from code bases such as Github and also to look into the issue of security of one's code, since the coder will be coding under monitoring of AI plugins installed on the coding platform running in the background.

REFERENCES

Alanazi, M., Soh, B., & Samra, H. (2025). PyChatAI : Enhancing Python Programming Skills — An Empirical Study of a Smart Learning System.

Bruch, M. et al. (2018). Learning from Examples to Improve Code Completion Systems.toHAL Id : hal-01575348 Learning from Examples to Improve Code Completion Systems.

Choi, I. C. (2025). Exploring the Impact of CodeCombat Python Programming Curriculum on Student Motivation at Primary School Exploring the Impact of CodeCombat Python Programming Curriculum on Student Motivation at Primary School. July. https://doi.org/10.1145/3719487.3719521

Ciniselli, M. et al. (2022). An Empirical Study on the Usage of Transformer Models for Code Completion. IEEE Transactions on Software Engineering, 48(12), 4818-4837. https://doi.org/10.1109/TSE.2021.3128234

Ì L. R. and Ì I. P. (2024). Factors that Influence Computer Programming Proficiency inin Higher Education : A Case Study of Information Technology Students. 36(July), 40-75.

Khomokhoana, P. J. and Nel, L. (2020a). Decoding Source Code Comprehension: Bottlenecks Experienced by Senior Computer Science Students. Communications in Computer and Information Science: Vol. 1136 CCIS (Issue January). https://doi.org/10.1007/978-3-030-35629-3_2

Munisamy, M. (2024). Code, Click, Learn : A Systematic Review of Online Assessment Tools in 21st Century Programming. March. https://doi.org/10.35631/IJMOE.620027

Saini, D. A., Sharma, M. D. and Tripathi, M. K. (2025). Artificial Intelligence (AI) in Education: Using AI Tools for Teaching and Learning Process. Ijireeice, 13(2). https://doi.org/10.17148/ijireeice.2025.13206

Tüfekci, A.and Köse, U. (2013). Development of an Artificial Intelligence-based Software System on Teaching Computer Programming and Evaluation of the System. Journal of Education), 28(2), 469-481.

Warth, A. (2008). Experimenting with Programming Languages. 0639876, 0–122. http://www.vpri.org/pdf/tr2008003_experimenting.pdf

Zhou, W. et al. (2022). Improving Code Autocompletion with Transfer Learning. Proceedings - International Conference on Software Engineering, 161–162. https://doi.org/10.1109/ICSE-SEIP55303.2022.9793983