

# Accelerating software development with AI: exploring the impact of ChatGPT and GitHub Copilot

Illia Solohubov<sup>1</sup>, Artur Moroz<sup>1</sup>, Mariia Yu. Tiahunova<sup>1</sup>, Halyna H. Kyrychek<sup>1</sup> and Stepan Skrupsky<sup>1</sup>

<sup>1</sup>National University "Zaporizhzhya Polytechnic", 64 Zhukovskiy Str., Zaporizhzhya, 69063, Ukraine

## Abstract

One of the directions in artificial intelligence (AI) development is the automation of the software development process. Programmers need to write a lot of code, but thanks to intelligent AI algorithms, tools are being developed that can assist with the automation of this process. In this paper an investigation is conducted into the capabilities and impact of artificial intelligence tools in program code development. The primary use cases and advantages of these tools in terms of development efficiency are examined and demonstrated. Two main tools that are currently used in program code development, Copilot and ChatGPT, are considered. A comparison is made regarding the speed of program code development, training based on certain test tasks, where a test group of developers is tasked with completing them using these tools. It is proven that the advancement of artificial intelligence technologies in the field of software development opens up new opportunities for improving productivity and efficiency in development. The obtained results confirm the advantages of using artificial intelligence as assistants in the development of program code for computer systems.

## Keywords

Copilot, ChatGPT, development, artificial intelligence, code generation

## 1. Introduction

Owing to the rapid development of AI technologies, including machine learning and deep learning, there has been significant distrust in society towards the use of these technologies [1]. This distrust is not limited to the general public but also extends directly to software developers.

The problem lies in developers having biases that artificial intelligence will not make a significant contribution and that there will always be a need to write code manually. They tend to think that AI will almost never be able to generate quality code, and developers will still have to rewrite it. However, this notion does not reflect reality, as the basic functions of artificial intelligence can easily generate code.

The aim of this study was to investigate and ascertain that artificial intelligence technologies, particularly machine learning and deep learning, will be used by all software developers in the future. The capabilities of these technologies were demonstrated, and how these technologies significantly streamline development [2].

With the rapid advancement of AI technologies, they are finding increasing applications in the field of software development. AI has a significant impact on the development process, easing the tasks of programmers and increasing productivity [3].

There are many AI tools that assist in software development:

1. Tabnine is an advanced AI-powered tool that's popular in developer communities. It leverages machine learning algorithms to autocomplete code and effortlessly converts natural language descriptions into functional code. Tabnine works as a code assistant and can learn your coding habits to predict what you intend to code and provide valuable suggestions.

CTE 2023: 11th Workshop on Cloud Technologies in Education, December 22, 2023, Kryvyi Rih, Ukraine

✉ illia.solohubov@gmail.com (I. Solohubov); arthur.official.moroz@gmail.com (A. Moroz); mary.tyagunova@gmail.com (M. Yu. Tiahunova); kirgal08@gmail.com (H. H. Kyrychek)

🆔 0009-0000-6140-3485 (I. Solohubov); 0009-0008-6742-2298 (A. Moroz); 0000-0002-9166-5897 (M. Yu. Tiahunova); 0000-0002-9437-9095 (S. Skrupsky)



© 2024 Copyright for this paper by its authors.

Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. DeepCode is a platform for static code analysis using artificial intelligence. It provides recommendations for code improvement, identifies potential errors, and suggests optimizations based on the analysis of millions of open repositories.
3. SonarQube is a continuous integration tool that provides code analysis for errors, code duplication, and other issues.
4. Github Copilot is powered by OpenAI CodeX, an AI system developed by OpenAI and trained on diverse data from multiple sources. This AI tool auto-generates and auto-completes code snippets.
5. ChatGPT is an AI-powered language model developed by OpenAI, capable of generating human-like text based on context and past conversations.

This paper has analyzed two AI tools for software development – Copilot and ChatGPT. Their combination, serving as a tool for rapid code generation/correction and as a tool for information retrieval/error correction/idea generation, provides a robust foundation for researching and uncovering the potential of AI in the future of software development and various other domains.

One of the directions in AI development is the automation of the software development process. Programmers need to write a lot of code, but thanks to intelligent AI algorithms, tools are being developed that can assist with the automation of this process. One of the recently released tools is OpenAI Codex from the GPT-3 project, which is used in ChatGPT and Copilot.

OpenAI Codex is capable of generating code based on a text description of the task or a common code template. This allows programmers to efficiently create programs using only a small amount of input data. Codex understands various programming languages and can provide useful recommendations regarding program structure and optimal use of functions.

The application of AI in software development, such as ChatGPT and Copilot, greatly eases the tasks of programmers and increases the speed of development. Due to their language understanding capabilities and code generation, they can offer auto-completion, hints, and rapid implementation options that help programmers accelerate the development process.

Copilot users can receive intelligent assistance while writing code, complementing their intentions and making recommendations on possible ways to solve the problem. ChatGPT, on the other hand, can be used for communicating with a program or system, allowing for the creation of natural language task descriptions and receiving responses in code format.

However, despite everything described above, there exists a large community of people who are sceptical about the idea of using these tools. There are many arguments made from both sides regarding the feasibility and benefits of using such tools.

The proposed research aims to:

1. Study the fundamental AI tools and their capabilities in code development.
2. Analyse the capabilities of the Copilot tool, its impact on development and the evaluation of the increased efficiency in development provided by this tool.
3. Analyse the capabilities of the ChatGPT tool in development, such as generating new code, analysing existing code, fixing errors, as well as its impact on the speed of learning new technologies.
4. Investigate the future prospects of the development of these technologies, the necessity of their use, and their implementation in development.

## 2. Investigating the impact of ChatGPT and Copilot

ChatGPT and Copilot are examples of cutting-edge technologies based on artificial intelligence, developed by OpenAI. Their goal is to enhance and simplify communication and software development processes.

## 2.1. Exploring the capabilities of Copilot

Copilot is a coding assistant that offers support in writing code by generating code constructs [4, 5, 6, 7, 8]. The researchers decided to test this tool on a simple task of creating a CRUD API for products, opting for the Dart programming language and the Flutter framework for implementation. Although Copilot claims to be effective in conjunction with languages such as Python, JavaScript, TypeScript, Ruby, and Go, it can actually work with any tool. Dart was chosen as an example for this study because it is not widely used, making the experiment more intriguing. The task involved implementing functionality for retrieving all products, fetching a single product by its identifier, creating, updating, and deleting a product. It was also necessary to create a product model with corresponding fields and simple markup for displaying a product card and delete button. The detailed functionality for creating and updating the product was excluded from the experiment since the result becomes apparent during the writing of this markup.

Initially, it was necessary to implement the Product class model. With Copilot's assistance, a basic structure of the class with two JSON serialization methods was suggested. After pasting this code into the development environment and editing it by adding the necessary fields, Copilot automatically suggests making changes to the constructor and serialization methods, which takes less than 5 seconds.

Next, to implement standard CRUD operations, a service class was created that is responsible for interacting with the API server. Copilot provided the basic functionality of this class without error handling or additional logic. For demonstration, figure 1 and figure 2 show how the bot tries to suggest additional functionality for this class.

```
20 Future<bool> toggleTag({required int productId, required int tagId}) async {  
    final response = await dio.put('$baseUrl/products/$productId/tags/$tagId');  
    final updatedProduct = Product.fromJson(response.data);  
    return updatedProduct.tagIds.contains(tagId);  
}
```

**Figure 1:** Generating code for the tag switch.

Therefore, with the help of Copilot, one can quickly generate standard code for implementing a simple class. Considering that business tasks are most often trivial, this tool can automate a larger part of the development. For more complex tasks, Copilot will be less efficient in generating code, but in such a case it begins to act as an automatic code generator, which, based on previously partially written code, can suggest additions that in most cases coincide with the desired code.

From the demonstration, it is evident that Copilot can set the basic structure for any functionality if even a small context is provided.

The markup, that is, the code of the interface, is the only place where Copilot cannot anticipate the subsequent code. This is because each project has its own interface, and it looks different. Design should not be fully standardised by nature, as it is considered a bad practice in design in general. It must be unique. Thus, although Copilot can help in creating graphical components, it is more used as a help in the form of code autocompletion in places where elements often repeat, such as indents, header styles. However, for input fields and buttons where a name and function are present, Copilot can suggest code that will automatically add a function to the event handler for pressing, based on what code was initialised.

It often happens that a developer, through inattention, may copy some part of the code and forget to edit variables or logic in some line of code, resulting in an inconspicuous error in the code. A key feature of Copilot in aiding code writing is also the absence of typos in the code.

Several tests were conducted with different subject areas, and approximate acceleration in code writing was derived. The data is demonstrated in figure 3, where three parts of a simple application – model, logic, interface, and the time in seconds spent writing them are depicted. People were asked to do the same task with Copilot and independently.

The evaluation may vary depending on the type of tasks, their complexity, and uniqueness. Also,

```
import 'package:copilot_test/repository/product.dart';
import 'package:dio/dio.dart';

final dio = Dio();
final baseUrl = 'https://testapp.com/api/';

class ProductService {
  Future<List<Product>> getProducts() async {
    final response = await dio.get('$baseUrl/products');
    final products = List<Product>.from(response.data.map((product) => Product.fromJson(product)));
    return products;
  }

  Future<Product> getById({required int id}) async {
    final response = await dio.get('$baseUrl/products/$id');
    final product = Product.fromJson(response.data);
    return product;
  }

  Future<Product> createProduct({required Product product}) async {
    final response = await dio.post('$baseUrl/products', data: product.toJson());
    final createdProduct = Product.fromJson(response.data);
    return createdProduct;
  }

  Future<Product> updateProduct({required Product product}) async {
    final response = await dio.put('$baseUrl/products/${product.id}', data: product.toJson());
    final updatedProduct = Product.fromJson(response.data);
    return updatedProduct;
  }

  Future<void> deleteProduct({required int id}) async {
    await dio.delete('$baseUrl/products/$id');
  }
}
```

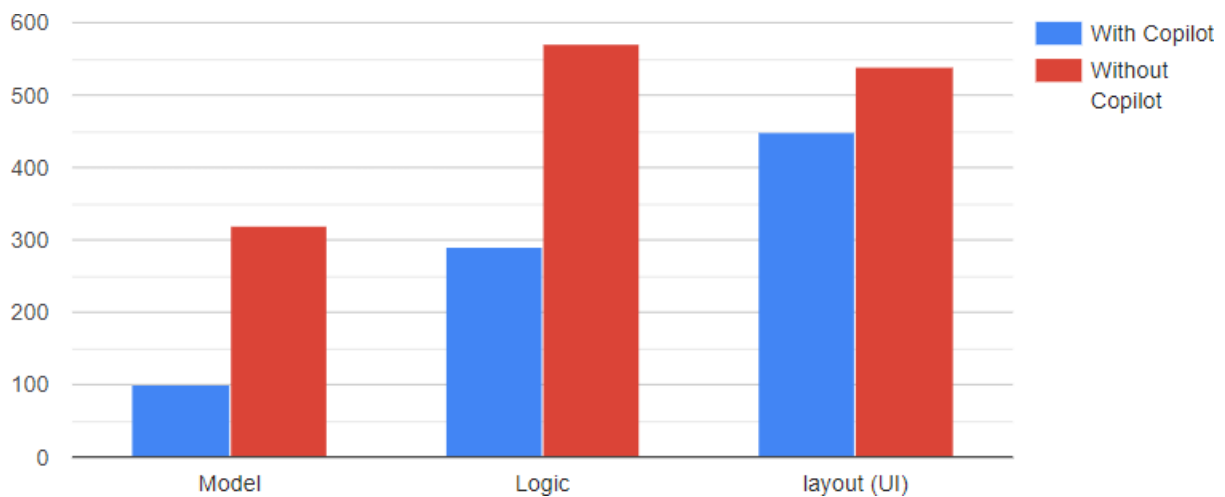
**Figure 2:** Generating Service class code for interacting with API based on initial data.

everything may depend on the initial experience of the programmer. However, on average, the results show that Copilot allows for approximately a 30% acceleration in writing the logic of the project and never adversely affects it.

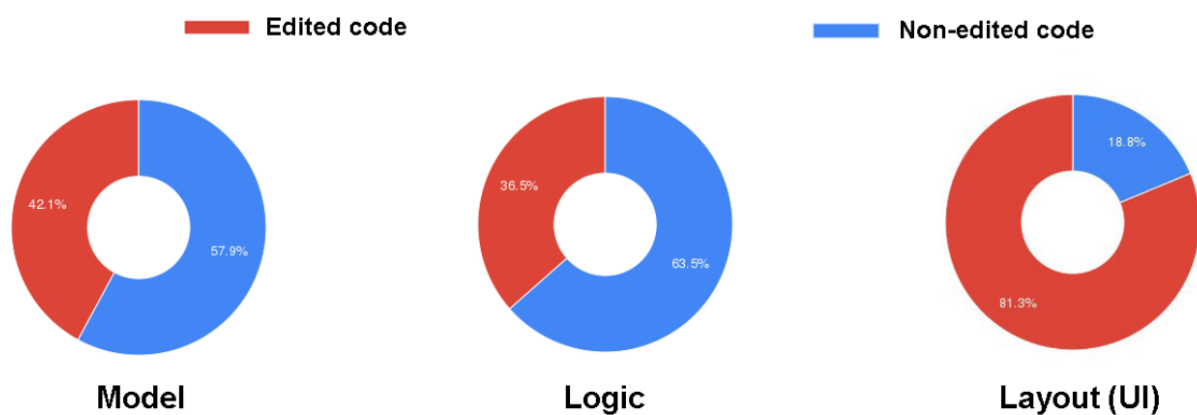
Development time, as mentioned above, is saved not only because we generate code with the bot, but also because less time is needed to check the written code, and errors in the code due to human factors are less frequent, which also saves a lot of man-hours in writing code.

It is also possible to consider the amount of code that the developer has to edit independently and correct what the bot suggests, write himself compared to how much can be left as is. For comparison, several tasks with different subject areas were also considered. The subject areas were chosen to be trivial – product display, shopping cart, data editing of an object, authorisation, simple chat. These tasks were chosen because there is some regularity in them since these tasks are solved most often. In other tasks, there is a very strong variation depending on the complexity of the task, the bot may suggest either entire independent functions or only certain pieces. The results are demonstrated in figure 4.

As can be seen, for the logical part of the project, the code largely remains as is, which is explained by the fact that often the code contains primitive actions such as storing some value of an element from



**Figure 3:** Comparison of the average time (in minutes) required to write code with AI assistance and without it.



**Figure 4:** Comparison of the amount of edited code and remaining code in three architectural layers – Domain (Model), Logic, UI (Interface).

user input, error handling, creating/editing an object which have common logic, differing only in that we know some information about an existing object, various authorisation methods that are done by template, or transitioning the application’s states like ‘loading’, ‘successful loading’, ‘error’, ‘not found’.

However, the situation remains worse for writing the interface, as the design needs to be unique. Nevertheless, some aspects of its development are simplified, as there are often repetitive elements in the code such as input fields, subheadings, margins, or styles of elements. Depending on the design, the influence of Copilot varies, but it is observed that the impact on writing the interface is minimal.

A significant advantage is also the fact that often, depending on the experience of the developer, the bot can provide a better solution than the one the developer might have suggested. Therefore, the developer can adopt the code proposed by the bot rather than their own. This is particularly beneficial for beginners and novices who are learning a new tool and are not familiar with all its capabilities.

## 2.2. ChatGPT feature research in development

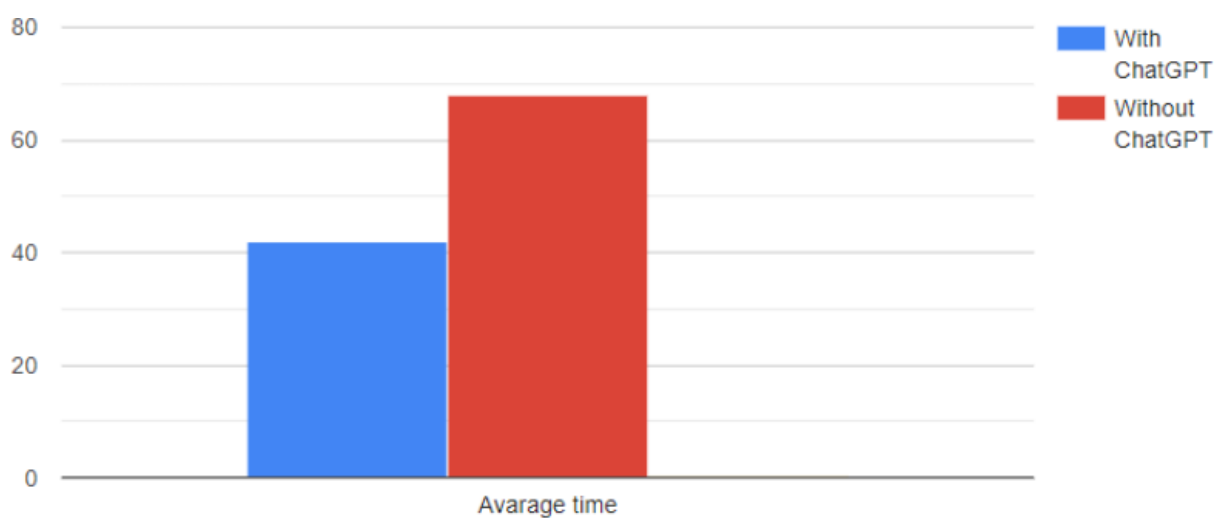
ChatGPT, in contrast to Copilot, serves as an assistant that responds to specific questions [9, 10, 11, 12]. Its use in development serves a different purpose. With its help, a developer can find solutions for their problems or quickly learn new tools.

### 2.2.1. First experiment

The first experiment of the study was aimed precisely at testing how quickly one can master a new tool if using ChatGPT. The specific time estimation is subjective as it all depends on the developer's skills and mindset, as well as the ability to learn. However, one can try to find a pattern by giving the same task to several people and asking them to accomplish certain tasks using the chat and without the chat.

As a task, several developers were asked to complete 5 programming tasks in programming languages unknown to them and to observe how quickly they could learn the basics of the language. They were only prohibited from asking for a ready-made solution, but were allowed to ask questions about the language's toolset, such as the availability of standard structures, classes, functions with their descriptions.

As can be seen in figure 5, learning on simple tasks occurs about a third faster. This result is explained by the fact that ChatGPT provides ready-made information with a detailed description of a specific tool or its parts, immediately provides the necessary libraries. Thus, the developer spends less time searching for information and can quickly start accomplishing the specific tasks set.



**Figure 5:** Comparison of average learning time independently and with the bot.

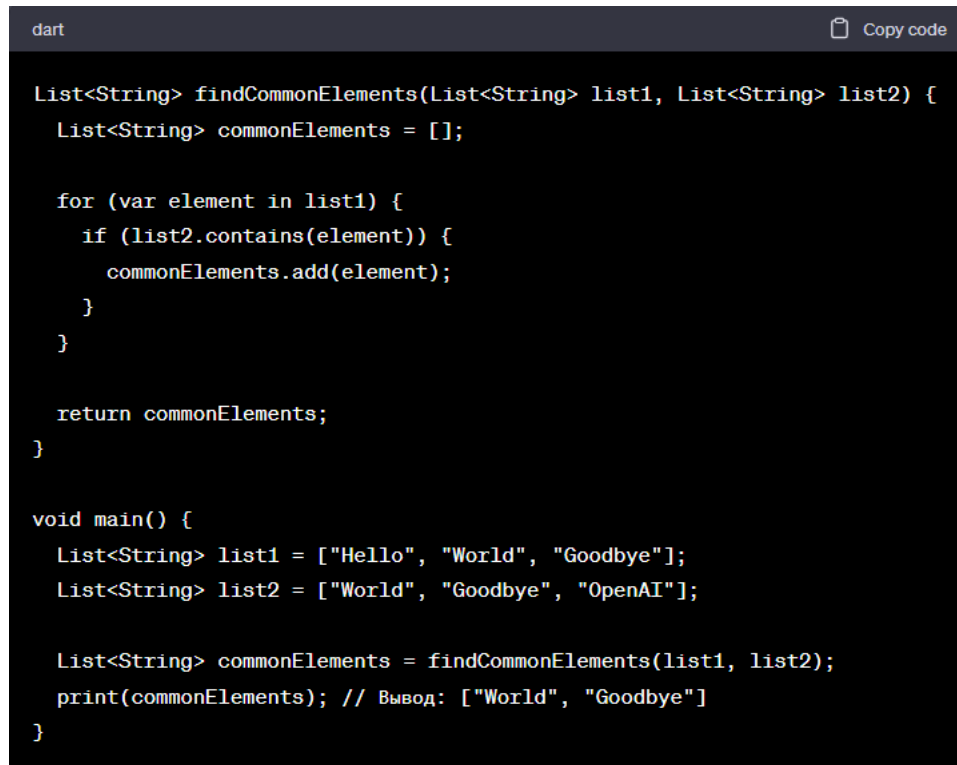
For more complex tasks such as learning new frameworks, the situation remains roughly the same, but in the long term plays a significant role as the acceleration is now measured in days or even weeks. This bot provides the necessary libraries, even those that are far down in search lists, and provides explanations, freeing one from reading extensive documentation.

### 2.2.2. Second experiment

This experiment is an investigation of ChatGPT's capabilities regarding code generation. Sometimes in development, there are moments when it is necessary to write some code or use an unknown library only for a specific task, which is specific to a particular project. For this, the bot was asked to generate code to build a model of Mars using the three.js library – a specific task where the client desired one model of a planet, another task was algorithmic – finding duplicates in two arrays.

This experiment is of particular interest because there are often situations where a specific task needs to be solved or a library's capabilities need to be utilized, but only about 1% of its functionality is required. In such cases, it can sometimes be more rational to find a ready-made solution. This is where ChatGPT can be useful – it can provide a ready-made solution with detailed explanations of how the code works, saving the programmer time searching for information. On the other hand, this can be considered a harmful habit if used too frequently. However, it often simply serves as a time-saving measure.

As can be seen in figure 6 and figure 7, ChatGPT allows for easy generation of working code, thus the developer can spend less time on research and learning the library and execute specific tasks. It also allows for quickly solving tasks in a more optimized way. Of course, more often than not this is considered bad practice, as the developer should learn and gain experience independently, but sometimes tasks need to be solved on a “here and now” basis, so this chatbot allows for quickly solving the given problem.

A screenshot of a code editor showing Dart code. The code defines a function `findCommonElements` that takes two lists of strings and returns a list of their common elements. It uses a `for` loop to iterate over the first list and checks if each element is in the second list. A `main` function is also shown, which creates two lists: `list1 = ["Hello", "World", "Goodbye"]` and `list2 = ["World", "Goodbye", "OpenAI"]`. It then calls `findCommonElements` and prints the result, which is `["World", "Goodbye"]`. The code is written in a dark-themed editor with a "Copy code" button in the top right corner.

```
dart Copy code  
  
List<String> findCommonElements(List<String> list1, List<String> list2) {  
  List<String> commonElements = [];  
  
  for (var element in list1) {  
    if (list2.contains(element)) {  
      commonElements.add(element);  
    }  
  }  
  
  return commonElements;  
}  
  
void main() {  
  List<String> list1 = ["Hello", "World", "Goodbye"];  
  List<String> list2 = ["World", "Goodbye", "OpenAI"];  
  
  List<String> commonElements = findCommonElements(list1, list2);  
  print(commonElements); // Вывод: ["World", "Goodbye"]  
}
```

**Figure 6:** Generated code for finding common elements with a complexity of  $O(N\log N)$  (best solution).

### 2.2.3. Third experiment

This experiment is associated with testing the capabilities in error correction by the chatbot. Let’s assume there is some code in which subtle logical errors have been made, figure 8. At first glance, it’s not noticeable that one of the vowels is missing in the array and that in the if block, the symbol `=` is used instead of `==`. Such errors may not occur often, but there are many other similar errors that can be found in the code and not be noticeable to the eye. Developers can spend hours searching for such errors.

In figure 9, it is demonstrated how the bot can find errors and correct them; as can be seen, the bot easily found these subtle errors and resolved the task set before it. This also greatly simplifies development and allows for quickly remedying the code, even if the developer did not write this code and received it for further development or correction.

## 3. Conclusions

Having analysed the two artificial intelligence tools as assistants for software development, it can be asserted that their emergence is a revolutionary new phase in development as a whole. Thanks to these tools, programmers can focus more on solving specific tasks without dwelling too much on their precise implementation and spending less time on code testing or error hunting, especially those that are due to human factors, such as inattentiveness or otherwise.



```
html Copy code  
  
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8">  
    <title>Mars Model</title>  
    <style>  
      body { margin: 0; }  
    </style>  
  </head>  
  <body>  
    <script src="https://threejs.org/build/three.js"></script>  
    <script>  
      // Initialize the scene  
      var scene = new THREE.Scene();  
  
      // Initialize the camera  
      var camera = new THREE.PerspectiveCamera(75, window.innerWidth / win  
      camera.position.z = 5;  
  
      // Initialize the renderer  
      var renderer = new THREE.WebGLRenderer();  
      renderer.setSize(window.innerWidth, window.innerHeight);  
      document.body.appendChild(renderer.domElement);  
  
      // Create geometry for Mars  
      var geometry = new THREE.SphereGeometry(1, 32, 32);  
  
      // Load the texture for Mars  
      var textureLoader = new THREE.TextureLoader();  
      var marsTexture = textureLoader.load('https://example.com/path-to-yo  
  
      // Material for Mars  
      var material = new THREE.MeshBasicMaterial({ map: marsTexture });  
  
      // Create the Mars model  
      var mars = new THREE.Mesh(geometry, material);  
  
      // Add Mars to the scene  
      scene.add(mars);  
  
      // Create an animation function  
      var animate = function () {  
        requestAnimationFrame(animate);  
  
        // Rotate Mars  
        mars.rotation.x += 0.005;  
        mars.rotation.y += 0.005;  
  
        renderer.render(scene, camera);  
      };  
  
      // Call the animation function  
      animate();  
  
      // Handle window resize  
      window.addEventListener('resize', function () {  
        var newWidth = window.innerWidth;  
        var newHeight = window.innerHeight;  
        camera.aspect = newWidth / newHeight;  
        camera.updateProjectionMatrix();  
        renderer.setSize(newWidth, newHeight);  
      });  
    </script>  
  </body>  
</html>
```

Figure 7: Generated code for displaying the model of Mars.



```
java Copy code

public class VowelCounter {
    public static void main(String[] args) {
        String input = "Hello, World!";
        int vowelCount = countVowels(input);

        // Display the vowel count.
        System.out.println("Number of vowels: " + vowelCount);
    }

    public static int countVowels(String input) {
        // Array of all vowels in lowercase.
        char[] vowels = {'a', 'e', 'o', 'u'};

        // Convert the input string to lowercase.
        input = input.toLowerCase();

        int count = 0; // Initialize the vowel count.

        // Iterate through each character in the string.
        for (int i = 0; i < input.length(); i++) {
            char currentChar = input.charAt(i);

            // Check if the current character is a vowel.
            for (char vowel : vowels) {
                if (currentChar == vowel) {
                    count++;
                    break; // Exit the loop if a vowel is found.
                }
            }
        }

        return count;
    }
}
```

Figure 8: Code with an error (for clarity, corrected in HTML).

The combination of these tools and the range of options they provide – code generation based on past context (Github Copilot) and fast information retrieval, error checking, and idea generation for development (ChatGPT) – pave the way for a new era where humans gradually move away from

```
java Copy code

public class VowelCounter {
    public static void main(String[] args) {
        String input = "Hello, World!";
        int vowelCount = countVowels(input);

        // Display the vowel count.
        System.out.println("Number of vowels: " + vowelCount);
    }

    public static int countVowels(String input) {
        // Array of all vowels in lowercase.
        char[] vowels = {'a', 'e', 'i', 'o', 'u'};

        // Convert the input string to lowercase.
        input = input.toLowerCase();

        int count = 0; // Initialize the vowel count.

        // Iterate through each character in the string.
        for (int i = 0; i < input.length(); i++) {
            char currentChar = input.charAt(i);

            // Check if the current character is a vowel.
            for (char vowel : vowels) {
                if (currentChar == vowel) {
                    count++;
                    break; // Exit the loop if a vowel is found.
                }
            }
        }

        return count;
    }
}
```

Figure 9: Code corrected by chatbot.

specificity and transition towards abstract thinking. This applies not only in the field of software development but also beyond.

## References

- [1] S. O. Semerikov, T. A. Vakaliuk, I. S. Mintii, V. A. Hamaniuk, V. N. Soloviev, O. V. Bondarenko, P. P. Nechypurenko, S. V. Shokaliuk, N. V. Moiseienko, V. R. Ruban, Development of the computer vision system based on machine learning for educational purposes, *Educational Dimension* 5 (2021) 8–60. doi:10.31812/educdim.4717.
- [2] K. Osadcha, V. Osadchyi, S. Semerikov, H. Chemerys, A. Chorna, The Review of the Adaptive Learning Systems for the Formation of Individual Educational Trajectory, in: O. Sokolov, G. Zholtkevych, V. Yakovyna, Y. Tarasich, V. Kharchenko, V. Kobets, O. Burov, S. Semerikov, H. Kravtsov (Eds.), *Proceedings of the 16th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops*, Kharkiv, Ukraine, October 06-10, 2020, volume 2732 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 547–558. URL: <https://ceur-ws.org/Vol-2732/20200547.pdf>.
- [3] M. Y. Tiahunova, H. H. Kyrychek, Y. D. Turianskyi, The system for testing different versions of the PHP, in: T. A. Vakaliuk, S. O. Semerikov (Eds.), *Proceedings of the 3rd Edge Computing Workshop*, Zhytomyr, Ukraine, April 7, 2023, volume 3374 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 112–129. URL: <https://ceur-ws.org/Vol-3374/paper09.pdf>.
- [4] P. Denny, V. Kumar, N. Giacaman, Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language, in: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023*, Association for Computing Machinery, New York, NY, USA, 2023, p. 1136–1142. doi:10.1145/3545945.3569823.
- [5] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, E. A. Santos, Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation, in: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023*, Association for Computing Machinery, New York, NY, USA, 2023, p. 500–506. doi:10.1145/3545945.3569759.
- [6] B. Zhang, P. Liang, X. Zhou, A. Ahmad, M. Waseem, Practices and Challenges of Using GitHub Copilot: An Empirical Study, in: *Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering, SEKE2023*, KSI Research Inc., 2023. doi:10.18293/seke2023-077.
- [7] S. Imai, Is GitHub copilot a substitute for human pair-programming? an empirical study, in: *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, ICSE '22*, Association for Computing Machinery, New York, NY, USA, 2022, p. 319–321. doi:10.1145/3510454.3522684.
- [8] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, Z. M. J. Jiang, GitHub Copilot AI pair programmer: Asset or Liability?, *Journal of Systems and Software* 203 (2023) 111734. doi:10.1016/j.jss.2023.111734.
- [9] M. A. Haque, S. Li, The Potential Use of ChatGPT for Debugging and Bug Fixing, *EAI Endorsed Transactions on AI and Robotics* 2 (2023). doi:10.4108/airo.v2i1.3276.
- [10] M. M. Rahman, Y. Watanobe, ChatGPT for education and research: Opportunities, threats, and strategies, *Applied Sciences* 13 (2023) 5783. doi:10.3390/app13095783.
- [11] M. Aljanabi, M. Ghazi, A. H. Ali, S. A. Abed, ChatGpt, *ChatGpt: Open Possibilities*, *Iraqi Journal For Computer Science and Mathematics* 4 (2023) 62–64. doi:10.52866/20ijcsm.2023.01.01.0018.
- [12] A. V. Riabko, T. A. Vakaliuk, Physics on autopilot: exploring the use of an AI assistant for independent problem-solving practice, *Educational Technology Quarterly* 2024 (2024) 56–75. doi:10.55056/etq.671.