International Journal of Leading Research Publication (IJLRP)
E-ISSN: 2582-8010 ● Website: www.ijlrp.com ● Email: editor@ijlrp.com

Automating the Code: Exploring AI-Powered Development Tool Github Copilot

AzraJabeen Mohamed Ali

Independent researcher, California,USA
Azra.jbn@gmail.com

Abstract

As the demand for faster and more efficient software development grows, artificial intelligence (AI) has emerged as a transformative force in automating various aspects of coding. This paper explores the integration of AI-powered development tool Github Copilot that streamlines the coding process, enhance productivity, and assist developers in solving complex programming challenges. We examine key AI-driven technologies such as code completion, bug detection, automated refactoring, and intelligent code generation, highlighting their capabilities and applications. Additionally, we analyze the role of machine learning and natural language processing in improving software development workflows, reducing human error, and enabling better decision-making. Through case studies and real-world examples, this paper assesses the benefits and limitations of AI-powered tools in coding environments, while considering ethical implications and the future of AI in programming. Ultimately, we present a comprehensive view of how AI is revolutionizing the coding landscape, paving the way for a more efficient and innovative approach to software development.

Keywords: Artificial Intelligence, Refactoring, Copilot, Machine Learning, Natural Language Processing

1. Introduction

Artificial intelligence (AI) has made significant strides in transforming various industries, and software development is no exception. Traditionally, coding has been a manual, time-consuming task that requires immense skill and attention to detail. However, with the rapid development of AI technologies, the coding process is undergoing a paradigm shift. AI is now being utilized to enhance, automate, and streamline various aspects of software development, from code generation to debugging and optimization.

AI in coding encompasses a wide range of applications, including intelligent code completion, bug detection, automated testing, and even generating complex code based on high-level specifications. By harnessing the power of machine learning, natural language processing, and deep learning, AI is enabling developers to write better, more efficient code with greater ease. These AI-powered tools are reducing the cognitive load on developers, allowing them to focus on more creative problem-solving tasks while automating repetitive and error-prone elements of the coding process.

"Pairing up with AI" reflects a shift from traditional coding practices to a more dynamic and intelligent partnership between humans and machines. There are several tools and platforms that developers can use to "pair up" with AI in coding, each designed to enhance the development process by leveraging AI's capabilities. Some of the most notable AI-powered tools used in modern software development are GitHub Copilot, Tabnine, Kite, Intellicode, DeepCode, Codota, Amazon CodeWhisperer, Refactor.ai

Github Copilot:

Developed by GitHub and OpenAI, GitHub Copilot acts as an AI-powered code completion tool that assists developers by suggesting entire lines or blocks of code based on the context of what is being written.

Key Features of GitHub Copilot:

- **Code Autocompletion:** GitHub Copilot can predict and suggest entire lines or blocks of code based on the context in which a developer is working. As the developer types, Copilot offers recommendations that match the coding pattern and intent, significantly reducing the need to manually write out repetitive code or search through documentation.
- **Context-Aware Suggestions**: Copilot uses machine learning to understand the surrounding code context and suggests the next appropriate lines or functions. It can predict the usage of variables, function names, and parameters based on what has already been written, providing seamless assistance across languages and frameworks.
- **Support for Multiple Programming Languages**: GitHub Copilot supports a wide variety of programming languages, including but not limited to Python, JavaScript, TypeScript, Ruby, Go, Java, and C#. This allows developers to use it across a range of projects, regardless of the programming language in use.
- **Natural Language to Code Translation:** One of Copilot's standout features is its ability to generate code from natural language descriptions. Developers can write comments in plain English describing what they want a function to do, and Copilot can automatically generate the corresponding code. This is especially useful for quickly implementing complex functions or algorithms without needing to memorize syntax.
- **Unit Test Generation**: GitHub Copilot can help write unit tests by generating test functions based on the code being written. This is invaluable in maintaining code quality and improving test coverage without the need to manually write repetitive or complex tests.
- **Documentation Assistance**: Copilot can also assist in generating documentation comments. By analyzing the code, it can create docstrings or comments that describe the functionality of functions, classes, and methods. This helps ensure that code is well-documented and understandable for others.

Steps to Install Github Copilot:

- GitHub Copilot requires Visual Studio 2022 or newer as a prerequisite.GitHub Copilot subscription is needed. If we don't have it yet, we can sign up for a trial or a paid plan at GitHub Copilot's official page.
- Open Visual Studio on a computer.Make sure to use Visual Studio 2022 or later for compatibility

with GitHub Copilot.

- In the top menu, click Extensions > Manage Extensions.
- In the Manage Extensions window, click on the Online tab on the left side.
- In the search box, type GitHub Copilot. Find GitHub Copilot in the search results.
- Click the Download button next to the extension.
- After the extension is installed, it will be prompted to restart Visual Studio for the changes to take effect. Click Restart Now.
- After restarting, GitHub Copilot should be ready to use. However, it is necessary to sign in to GitHub to authenticate.
- If GitHub is not signed in yet, Visual Studio will prompt to log in to GitHub account.
- Click Sign in and follow the steps to authenticate using GitHub account. It is to ensure GitHub Copilot subscription is active, as it needs a valid subscription to use the service.

Steps to Implement Code Completion Using GitHub Copilot:

1. Sign into Github Copilot:

After installation, it will be prompted to sign in to GitHub account. GitHub Copilot requires a valid GitHub Copilot subscription.

- Sign in via GitHub or use GitHub Enterprise credentials.
- After signing in, GitHub Copilot will be activated and ready to help with code completion.

2. **Start Coding with GitHub Copilot:**

- Open any project in Visual studio/ VS Code.
- Once code typing begins, GitHub Copilot will automatically provide suggestions. These suggestions are based on the context of the code which are written.
- For example, While writing a function, if comment is typed describing the function's purpose (in plain English), GitHub Copilot will try to generate the relevant code.

3. Using Code Completion in GitHub Copilot:

- **Basic Code Completion:**During typing, GitHub Copilot will offer code suggestions, such as variable names, function signatures, and snippets of code. These will appear in grayed-out text. To accept a suggestion, Press Tab or Enter to accept the suggestion and auto-complete the line.
- **Function Suggestions:** When typing function names or code inside functions, Copilot will suggest complete function bodies or code snippets based on the current context. To cycle through suggestions: If the first suggestion is not what is expected,then by pressing Ctrl+Space (or Cmd+Space on macOS) we can see additional suggestions or use Alt + [or Alt +] to navigate through multiple suggestions.

4. **Natural language to Code Translation**: We can write comments that describe the functionality we need, and Copilot will generate code to implement it. For example:Comment: // Function to calculate the factorial of a number. Copilot will suggest the corresponding function to calculate the factorial as per Fig-1.

Fig-1:

```
// Function to calculate the factorial of a number
function factorial(n) {
  if (n === 0) return 1;
  return n * factorial(n - 1);
}
```

5. Fine-Tuning GitHub Copilot Suggestions:

Sometimes, GitHub Copilot's suggestions may not be exactly what we are looking for. We can fine-tune Copilot's suggestions by:

- Modifying the code or comment and seeing how Copilot adapts to the new context.
- If a suggestion is not helpful, press Escape to dismiss it and continue coding manually.
- Copilot will often learn from our preferences, helping improve its suggestions over time.

6. Using GitHub Copilot for Code Snippets and Templates:

GitHub Copilot is very effective for generating repetitive code snippets or boilerplate code. For example:

- **Creating a class**: Type class and the class name, and Copilot will provide a template for the class with constructor methods, getter/setter methods, and other boilerplate code.
- **Creating loops**: If we start writing a loop, Copilot will often provide suggestions for common loop patterns (e.g., for loops, while loops).

Refactoring Code using Github Copilot:

Here are a few C# refactoring examples where GitHub Copilot can suggest improvements or optimizations.

Example1 : Refactoring a Method with Repeated Code: Here is the original code with repetitions.In this Fig-2 code, the two methods are performing simple arithmetic operations (add, subtract). While the code works fine, it has redundancy due to the repeated a and b parameters in each method.

Fig-2:

```
public class Refactoring_Github
{

        public double Add(double a, double b)
        {
            return a + b;
        }

        public double Subtract(double a, double b)
        {
            return a - b;
        }

}
```

Refactored Code (with helper method): GitHub Copilot might suggest consolidating the common logic into a single method. Here, Fig-3 GitHub Copilot refactored the code by creating a generic method PerformOperation, which takes a Func<double, double, double> (a function that takes two doubles and returns a double). Each operation (add, subtract, etc.) is now a lambda expression passed to the helper

method, reducing repetition and improving code maintainability.

Fig-3:

```
public class Refactoring_Github
{
    private double PerformOperation(double a, double b,
        Func<double, double, double> operation)
    {
        return operation(a, b);
    }

    public double Add(double a, double b)
    {
        return PerformOperation(a, b, (x, y) => x + y);
    }

    public double Subtract(double a, double b)
    {
         return PerformOperation(a, b, (x, y) => x - y);
    }

}
```

Example 2: Refactoring Complex Conditional Logic: Original Code (complex if-else block) Fig-4:This method contains a series of if-else statements. Although functional, it could be simplified.

Fig-4:

```
public double CalculateDiscount(double amount)
{
    if (amount >= 10000)
    {
        return amount * 0.40; // 40% discount
    }
    else if (amount >= 5000)
    {
        return amount * 0.20; // 20% discount
    }
    else if (amount >= 1000)
    {
        return amount * 0.10; // 10% discount
    }
    else
    {
        return 0;
    }
}
```

Refactored Code (using switch expression): GitHub Copilot might suggest using a switch expression or a dictionary-based approach to improve readability.In this refactor Fig-5, GitHub Copilot suggests using a switch expression for cleaner, more concise conditional logic. This improves the readability and efficiency of the code.

Fig-5:

```
public double CalculateDiscount(double amount)
{
    return amount switch
    {
        >= 10000 => amount * 0.40, // 40% discount
        >= 5000 => amount * 0.20,  // 20% discount
        >= 1000 => amount * 0.10,  // 10% discount
        _ => 0
    };
}
```

Example 3: Refactoring Nested Loops:

Original Code (using nested loops): The function Fig-6 multiplies two matrices using three nested loops, which can be hard to maintain and inefficient for large matrices.

Fig-6:

```
public int[,] Multiply(int[,] matrix1, int[,] matrix2)
{
    int rows = matrix1.GetLength(0);
    int cols = matrix2.GetLength(1);
    int[,] result = new int[rows, cols];

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            for (int k = 0; k < matrix1.GetLength(1); k++)
            {
                result[i, j] += matrix1[i, k] * matrix2[k, j];
            }
        }
    }
    return result;
}
```

Refactored Code (optimizing the loops): GitHub Copilot might suggest optimizing the loop structure or using parallel processing for better performance, especially when working with large matrices.In this refactored version Fig-7, GitHub Copilot suggests using parallel processing (Parallel.For) to divide the work across multiple threads. This can drastically improve performance when dealing with large matrices, particularly in multi-core systems.

Fig-7:

```
public int[,] Multiply(int[,] matrix1, int[,] matrix2)
{
    int rows = matrix1.GetLength(0);
    int cols = matrix2.GetLength(1);
    int[,] result = new int[rows, cols];

    Parallel.For(0, rows, i =>
    {
        for (int j = 0; j < cols; j++)
        {
            for (int k = 0; k < matrix1.GetLength(1); k++)
            {
                result[i, j] += matrix1[i, k] * matrix2[k, j];
            }
        }
    });

    return result;
}
```

Example 4: Simplifying Function with LINQ:

Original Code (using traditional loop): This function manually iterates through a list and adds even

numbers to a new list Fig-8.

Fig-8:

```
public List<int> GetEvenNumbers(List<int> numbers)
{
    List<int> evenNumbers = new List<int>();
    foreach (var number in numbers)
    {
        if (number % 2 == 0)
        {
            evenNumbers.Add(number);
        }
    }
    return evenNumbers;
}
```

Refactored Code (using LINQ): GitHub Copilot might suggest using LINQ (Language Integrated Query) to make the code more concise.In this refactored version Fig-9, GitHub Copilot simplifies the code by using LINQ's Where method, which filters the list for even numbers, followed by ToList to convert the result back into a list.

Fig-9:

```
public List<int> GetEvenNumbers(List<int> numbers)
{
    return numbers.Where(n => n % 2 == 0).ToList();
}
```

Customizing GitHub Copilot Settings (Optional):

We can customize GitHub Copilot's behavior through settings in VS Code. Some common settings include:

- **Enabling/Disabling Copilot suggestions**: We can toggle suggestions on or off by going to Settings > Extensions > GitHub Copilot.
- **Controlling suggestion frequency**: We can adjust how frequently Copilot offers suggestions.
- **Choosing specific languages or frameworks**: We can limit Copilot's suggestions to certain languages or frameworks we work with regularly.

Best Practices for Using GitHub Copilot to Refactor Code:

- **Provide Clear Context:** When code has to be refactored, clear comments should be provided or the code has to be adjusted indicatingclear goals (e.g., "simplify this loop", "optimize for performance") will help GitHub Copilot generate more accurate suggestions.
- **Accept Suggestions and Review:** Always review the suggestions provided by Copilot. While it's powerful, we should validate the refactored code for correctness, style consistency, and performance.
- **Iterative Refactoring**: Refactoring is often an iterative process. Apply one set of changes, review them, and continue making improvements as necessary.
- **Test Refactored Code**: After applying refactoring suggestions, always test the code to ensure it works as expected and doesn't introduce any regressions.

Challenges:

- **Code Quality and Accuracy:** While GitHub Copilot generates useful suggestions, it is important to verify the correctness of the code. Since Copilot is trained on public code, it may occasionally recommend suboptimal solutions or outdated practices.
- **Security Risks**: The AI model is trained on a wide range of public code repositories, some of which may contain security vulnerabilities. Developers must exercise caution and review the code thoroughly before using it in production.
- **Dependence on AI**: Developers must be cautious not to become overly reliant on GitHub Copilot. While it's a helpful tool, it's essential to understand the underlying principles of the code being written to ensure long-term maintainability and control.

Benefits of Using GitHub Copilot:

- **Increased Productivity:** By automating code generation and reducing the need for manual lookups, developers can focus more on solving the core logic of the application.
- **Smarter Coding**: Copilot provides insights and best practices, suggesting code optimizations and solutions that might not be immediately obvious to the developer.
- **Faster Prototyping**: Copilot allows developers to rapidly prototype applications, quickly generating boilerplate code and simple functions, which speeds up the development process.
- **Improved Learning**: For less experienced developers, GitHub Copilot serves as a tutor, guiding them through the process of writing efficient code, learning best practices, and understanding complex coding patterns.

Conclusion:

GitHub Copilot represents a transformative advancement in the way developers approach coding, offering a seamless blend of artificial intelligence and software development. By leveraging OpenAI's powerful language model, GitHub Copilot significantly accelerates the development process, making it possible to write high-quality code more efficiently, with fewer errors, and with less manual effort.Through features like auto-completion, code refactoring, documentation assistance, and problem-solving suggestions, GitHub Copilot helps developers focus on higher-level problem solving rather than routine coding tasks. Whether it's suggesting entire functions, providing real-time documentation, or proposing optimizations, GitHub Copilot acts as an intelligent assistant that adapts to a developer's style and preferences, offering context-aware recommendations.However, while GitHub Copilot is a valuable tool, it is not without limitations. Developers must still actively review the suggestions provided by Copilot to ensure accuracy, avoid potential security risks, and maintain the project's consistency. Additionally, AI-generated code requires human oversight to ensure it meets the unique needs of the project and adheres to best practices.

In conclusion, GitHub Copilot is an invaluable asset in the modern development landscape, enhancing productivity and reducing the time spent on repetitive tasks. As AI-driven tools continue to evolve, their role in shaping the future of software development will only grow, creating a more efficient and accessible coding environment for developers of all experience levels. By combining the strengths of AI with human creativity and expertise, GitHub Copilot exemplifies the potential for technology to augment and amplify the software development process.

References

[1] Microsoft, "Improving GitHub Copilot Completions in Visual Studio for C# Developers" https://devblogs.microsoft.com/dotnet/improving-github-copilot-completions-in-visual-studio-for-csharp-developers/(Sep 18, 2024)

[2] Visual Studio, "GitHub Copilot Completions"https://marketplace.visualstudio.com/items?itemName=GitHub.copilotvs(Jun25, 2024)

[3] Marek Sirkovský"GitHub Copilot and others: Test of five Code Assistants for C#" https://mareks-082.medium.com/github-copilot-and-others-test-of-five-code-assistants-for-c-8ce252786b81 (Apr6, 2024)

[4] Robert Nubel"Refactoring with GitHub Copilot" https://rnubel.hashnode.dev/refactoring-with-github-copilot(Feb 14, 2023)

[5] Visual Studio Code "C# Quick Actions and Refactorings" https://code.visualstudio.com/docs/csharp/refactoring(Jun06, 2023)

[6] Ivan Stove "Dependency Injection Lifetimes in .NET" https://levelup.gitconnected.com/cdependency-injection-lifetimes-in-net-566830a633ca(May24, 2022)

[7] Michael D. Callaghan"P-AI-R Programming: How Al tools like GitHub Copilot and ChatGPT Can Radically Transform Your Development Workflow"P-AI-R Programming Publication (Mar17, 2023)

[8] Rob Botwright "Github Copilot For Developers: Smart Coding With AI Pair Programmer" Pastor Publishing Ltd (Apr11, 2024)

[9] Mariot Tsitoara"Beginning Git and GitHub: Version Control, Project Management and Teamwork for the New Developer Second Edition" Apress Publication(Mar 15, 2024)

[10] Chris Minnick"Coding with AI" For dummies publication (Mar26, 2024)

[11] Christoffer Noring, Anjali Jain, Marina Fernandez, Ayşe Mutlu, Ajit Jaokar"AI-Assisted Programming for Web and Machine Learning: Improve your development workflow with ChatGPT and GitHub Copilot" PacktPublishing (Aug30, 2024)