Response of Apple Inc. to the Copyright Office's AI NOI Question #18

Submitted by Tom LaPerle, Senior Director
Oct. 30, 2023

18. Under copyright law, are there circumstances when a human using a generative AI system should be considered the "author" of material produced by the system? If so, what factors are relevant to that determination? For example, is selecting what material an AI model is trained on and/or providing an iterative series of text commands or prompts sufficient to claim authorship of the resulting output?

Overview:

Apple responds to this question in the context of the use of automated tools for generating computer program code, including both older legacy tools and more recent generative AI coding tools, and posits that in circumstances where a human developer controls the expressive elements of output and the decisions to modify, add to, enhance, or even reject suggested code, the final code that results from the developer's interactions with the tools will have sufficient human authorship to be copyrightable.

Under current Copyright Office guidance, the Office considers whether a work is authored by a human, with a computer or other device acting as an assisting tool to embody the human's original literary, artistic, or musical expression or selection, coordination, or arrangement. *See Copyright Registration Guidance: Works Containing Material Generated by Artificial Intelligence*, 88 Fed. Reg. 16190, 16192 (Mar. 16, 2023) [hereinafter *Registration Guidance*]. The Office states that works containing AI-generated material have sufficient human authorship to be protected by copyright in certain circumstances. For example, the Office makes clear that works continue to be protected by copyright when humans use tools, whether traditional or technological, either to create works or to recast, transform, or adapt material as expressive authorship. *See id.* at 16193; 17 U.S.C. § 101 (defining derivative works). In addition, the creative selection, coordination, or arrangement of AI-generated content by humans can result in an original work of authorship. *See* 17 U.S.C. § 101 (defining compilation). In both circumstances, the key factor is the extent of human creative control over the work's expression and literary, artistic, or musical elements. *See Registration Guidance* at 16193.

Software developers at Apple have been using automated tools to assist them in writing computer programs for decades. The capabilities and complexity of such tools have ranged across a broad spectrum, from error detection and suggested corrections to human authored code, to creation of initial drafts of a module of code for a specific function or purpose. More recently, with the advent of generative AI coding tools, the capabilities of the tools have expanded. Yet even these more recent generative AI tools can still be used to perform the less complex tasks for which legacy tools are typically used, often with better results. It is important to recognize this long legacy of use of automated tools in order to understand that generative AI coding tools are not something completely new. Rather, they are an evolution—albeit a

significant one—of tools that have long formed an integral part of computer program development by human authors.

As discussed in more detail below, human developers control, as assisting instruments, both legacy coding tools and generative AI coding tools. Use of such tools typically involves an interactive process with the human developer, who controls not only what code is produced by the tool, but also the forms, including transformations, in which it may ultimately be integrated in a computer program. In cases of automated code tool use (regardless of type), the developer must examine the code or corrections suggested by the tool for correctness, efficiency, quality, readability, and other criteria and make appropriate modifications, additions, enhancements, substitutions, or other changes, or decide to reject use of the suggested code entirely. This is especially true for generative AI tools, which sometimes produce code that is not secure, or is incorrect, nonsensical, or extraneous (a problem referred to as "hallucination"). In such circumstances, when the human developer controls the tools, examines the suggested code, and determines the form, including transformations, in which it will be used, the final, developed code will have sufficient human authorship to be copyrightable.

Nomenclature

This comment describes the range of automated tools, both legacy and generative AI, and how they can be used by programmers for the development of code. The comment will refer to the output of AI coding tools as "**suggested code**," because, in all cases, the human developer using the tool remains responsible, both for the correctness, efficiency, quality, or readability of the suggested code produced by the tool in use, and for making modifications, additions, substitutions, and enhancements to the suggested code—or the decision to reject it entirely—before making any use of it in the final code. The developer is responsible for that final code, which will be referred to in this comment as the "**developed code**." Because of the developer's control of the tool and necessary review and edits, the developed code, regardless of the tool used, has sufficient human authorship to be protected by copyright. This is true whether the suggested code is used to develop a standalone module of developed code or is incorporated, combined, or interfaced with code written entirely by the developer or by other developers. And it is true whether the tool is a generative AI tool or is a legacy, non-AI tool.

Legacy Coding Tools

Legacy coding tools of the types described in this section have been used for development of computer program code for decades. All these tools typically use some combination of templating, static analysis, and other manually written heuristics. With limited exception, they do not typically involve training the tool on data as is the case with recent generative AI tools. To the extent a user might provide a "prompt" to a legacy coding tool, it would typically be only to provide pre-specified information, such as what section of code the tool is to be run on, rather than the sort of free-form prompts that generative AI tools utilize today.

IDE Tools. Software developers typically use a structured software platform called an "Integrated Development Environment" (IDE), which provides features to streamline and simplify the process of writing code. IDEs have included built-in auto-correct, error detection, code refactoring, and code generation tools (described in the next four subsections) for many years. For example, Microsoft's Visual Basic IDE has included graphical user interface code generation since 1991, auto-complete since 1996, real-time error detection since at least 1998, and code refactoring since 2005. Other IDEs with similar features include Apple's Xcode, Netbeans, Eclipse, and IDEs from the company JetBrains.

Auto-Complete. These tools suggest ways to complete a partial line of code that a programmer has written, analogous to the auto-complete function on a smartphone keyboard that suggests how to complete a word or phrase being typed in by the user. Today some auto-complete tools make limited uses of AI to choose the ordering in which non-AI suggestions for completion are displayed.

Real-Time Error Detection and Correction. Most IDEs include tools to detect certain kinds of errors in code, such as syntax errors or missing parameters, while the developer is writing the code so that the developer can identify and correct them without needing to run the code. For certain simple errors, the tool might also suggest a fix according to a template.

Code Refactoring. These tools provide the ability to perform certain pre-defined types of tasks that modify large amounts of code. These tasks are usually conceptually simple but would be labor intensive for a developer to perform manually—for example, cutting and pasting code from one file to another and updating any other code that utilizes the moved code so that the other code looks for the moved code in the new file instead of the old one.

Code Generation. Many IDEs provide limited, non-AI code generation tools that typically perform generation based on a template. This template gets automatically filled in based on parsing code the user has already written or according to other specifications supplied by the user. These tools are limited to generating only code that they have explicitly been designed to generate. For example, a code generator might assist in producing code for a graphical user interface by allowing the developer to click and drag an element such as a button or text box to visually insert it into the interface. The code generator would then automatically generate code corresponding to that element.

Template-Based Coding Frameworks. These tools help automate the labor-intensive task of implementing "CRUD" functionality, where CRUD stands for Create-Read-Update-Delete. For example, a health records application might enable a doctor to create a new prescription entry, read and update existing prescriptions, and delete prescriptions when they are no longer needed. Template-based coding frameworks like Ruby on Rails—first released in 2004—can automatically generate code for basic CRUD functionality.

Compilers. A compiler is a software program that takes code written in one programming language and automatically translates it into a different programming language.

Most compilers give the developer some degree of control over the characteristics of the generated code. The first compilers date back to the mid-1900s. And some types of compilers such as "transpilers" generate code that can be designed to be read and edited by humans.

AI-Based Coding Tools

More recent AI-based coding tools can achieve greater efficiencies in development or reductions in development time than legacy tools by offering broader or more robust capabilities. A recent small study performed by GitHub, done on a sample of GitHub Copilot users, indicated that these users used 55% less time to complete a select programming task. (See https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/).

Examples of AI-based tools that are currently available for code development include GitHub Copilot (developed by a partnership of Microsoft and OpenAI), Amazon's CodeWhisperer, IBM's watsonx.ai, Replit's GhostWriter, Sourcegraph's Cody, Codata's Tabnine, OpenAI's ChatGPT chatbot, and Anthropic's Claude chatbot. Apple's expectation is that generative AI tools could over time become just as ubiquitous and important as legacy tools to software development projects.

Two types of generative AI tools can be distinguished:

Code-Completion Style Tools. These AI-based tools have been in use for a few years. They take as input the code from a file (or possibly multiple files) and attempt to fill in a section of the code that has been left blank. Usually, the tool user will input some plain English text at the beginning of the blank section to explain what code the user wants to fill that section.

Chat-Based Tools. Chat-based tools have been in use for only a year or less. They allow a user to "chat" with an AI. For example, the user can copy and paste code into the chat window and ask it to do things like make changes to the code, add code, or help with debugging. The user can also ask the tool to perform larger scale changes to the code, with varying degrees of success.

The user might have a back-and-forth with a chat-based tool via a sequence of prompts clarifying what the user is seeking and asking the tool to correct issues the user points out. Or, in the case of a code-completion style tool, the user might need to try variations of the prompt to find a version that works better to produce the desired result.

The Interactive Process with the Developer

In all cases of automated code tool use, whether legacy coding tools or generative AI coding tools, the developer using the tool is responsible for ensuring the generated, corrected, or completed code suggested by the tool meets their requirements for correctness, efficiency, quality, readability, or other criteria and for making appropriate modifications, additions,

enhancements, substitutions, or other changes, or deciding to reject use of the suggested code entirely. This is especially true for generative AI tools, given the possibility of hallucination, the inclusion of extraneous code that needs to be deleted, or suggested code that may require modifications for security issues.

Thus, these tools involve an interactive process with the human developer, who is the final arbiter not only of what is produced, but also in what form it may ultimately be used, if at all, in developed code. In such circumstances, when the human developer controls the use of the tool and the decision to modify, add to, enhance, or even reject the suggested code, the final code that results from the developer's interactions with the tool will have sufficient human authorship to be copyrightable.