PharmaSUG China 2025 - Paper AP-114

Accelerating Clinical Data Management Programming with Copilot AI assistance

Peng Tan, Kai Yan Sanofi Corporation

ABSTRACT

In the fast-changing area of clinical research, good data management programming is crucial to make sure that research results are accurate, dependable, and available when needed. However, traditional clinical data management methods often involve time-consuming manual processes, especially the management of clinical programs for multiple study under the same compound. This article explores the transformative potential of integrating Github Copilot AI assistance into clinical data management programming workflows. By leveraging advanced artificial intelligence algorithms, Github Copilot AI can generate optimized code snippets and provide real-time guidance to programmers. This not only speeds up the programming process, but also improves the quality and consistency of deliverables. We discuss the specific process implementation of the successful application of Copilot AI. Our hands-on results show that using AI in programming greatly cuts down the time it takes to develop data management program, leading the way for faster and stronger methods in clinical data management.

INTRODUCTION

In the field of clinical research, data management programming plays a vital role in ensuring the integrity and reliability of research results. As clinical trials become more complex, data management for multiple studies under the same compound brings more challenges. Traditional clinical data management programming methods often rely heavily on manual processes, which are time-consuming and error-prone, slowing down research progress and increasing the risk of inconsistent delivery results. Faced with the need for more efficient and accurate programming solutions, we have begun to explore innovative tools and methods such as artificial intelligence (AI). AI, in particular, has shown the potential to change clinical data management work with its ability to automate repetitive tasks and optimize processes. In this context, the integration of Github Copilot AI into the clinical data management workflow is seen as an important development. The advanced machine learning algorithms equipped with Copilot AI provide real-time assistance to programmers, allowing them to generate code with greater efficiency while maintaining high standards of quality. This article delves into the application of Copilot AI in clinical data management, showing how it solves existing challenges and simplifies programming tasks, aiming to provide readers with valuable insights into how AI-assisted programming can drive clinical trial data management to be more flexible, precise, and efficient.

GITHUB COPILOT

GitHub Copilot is an AI-powered code completion tool developed by GitHub in collaboration with OpenAI. It acts as a coding assistant, helping developers write code faster and more efficiently by providing intelligent suggestions directly in their editor.

Key Features:

- **Code Suggestions**: Copilot suggests entire lines or blocks of code based on the context of what you're writing.
- **Multi-language Support**: It supports a wide range of programming languages, including Python, JavaScript, TypeScript, Java, C#, and SAS as well.
- **Context Awareness**: It understands the context of your code, including comments, function names, and variable declarations, to provide relevant suggestions.

- **Learning from Repositories**: Copilot is trained on a vast dataset of public code repositories, enabling it to offer solutions to common coding problems.
- **Integration**: It integrates seamlessly with popular editors like Visual Studio Code, Neovim, and JetBrains IDEs.

Benefits:

- **Increased Productivity**: Automates repetitive coding tasks and reduces boilerplate code.
- **Learning Tool**: Helps developers learn new coding patterns and best practices.
- **Error Reduction**: Reduces the likelihood of syntax errors by providing accurate suggestions.

GITHUB CODESPACE

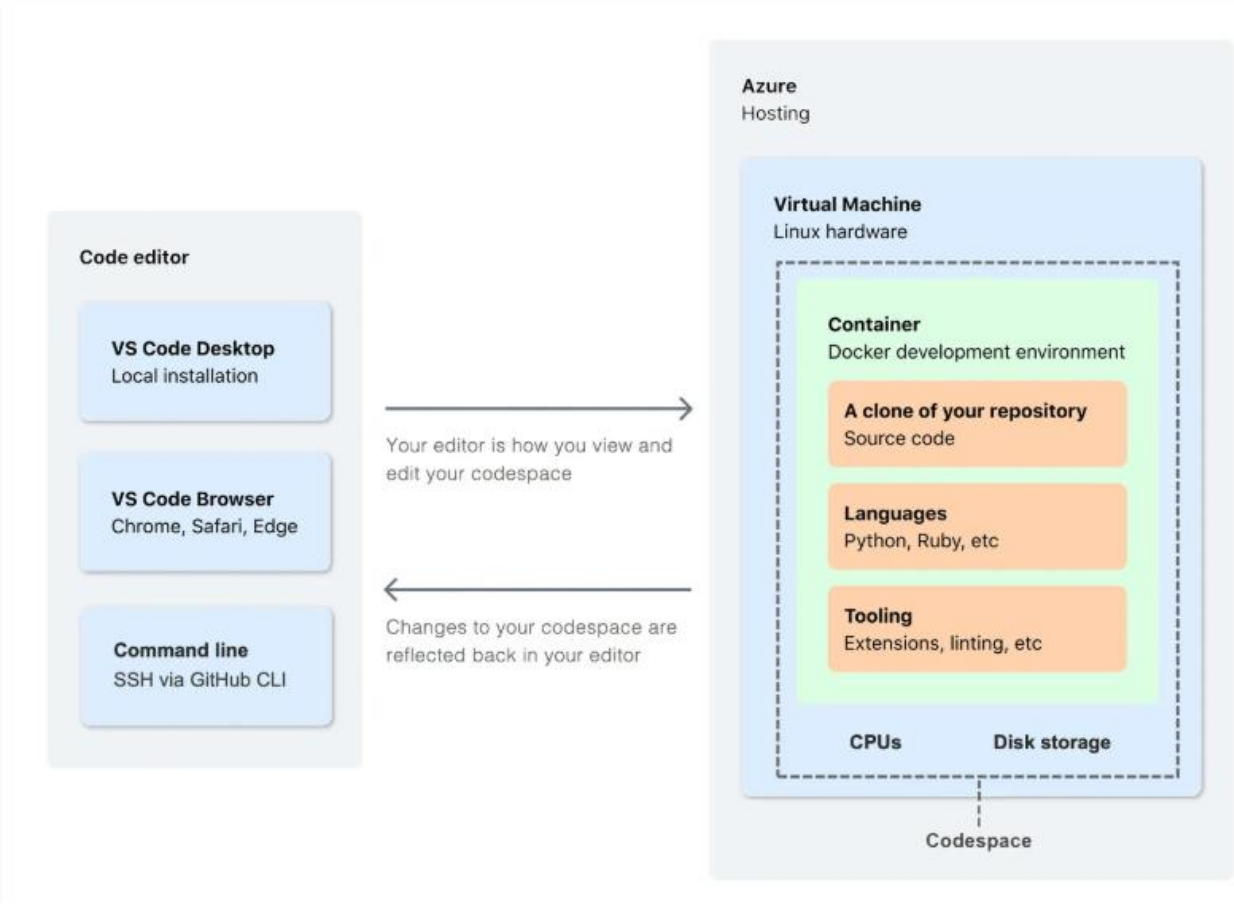
A Codespace is a development environment that's hosted in the cloud. You can customize your project for GitHub Codespace by committing configuration files to your repository (often known as Configuration-as-Code), which creates a repeatable Codespace configuration for all users of your project.

Each Codespace you create is hosted by GitHub in a Docker container, running on a virtual machine. You can choose from a selection of virtual machine types, from 2 cores, 8 GB RAM, and 32 GB storage, up to 32 cores, 64 GB RAM, and 128 GB storage.

By default, the Codespace development environment is created from an Ubuntu Linux image that includes a selection of popular languages and tools, but you can use an image based on a Linux distribution of your choice and configure it for your particular requirements. Regardless of your local operating system, your Codespace will run in a Linux environment. Windows and macOS are not supported operating systems for the remote development container.

You can connect to your Codespace from your browser, from Visual Studio Code, or by using GitHub CLI. When you connect, you are placed within the Docker container. You have limited access to the outer Linux virtual machine host.

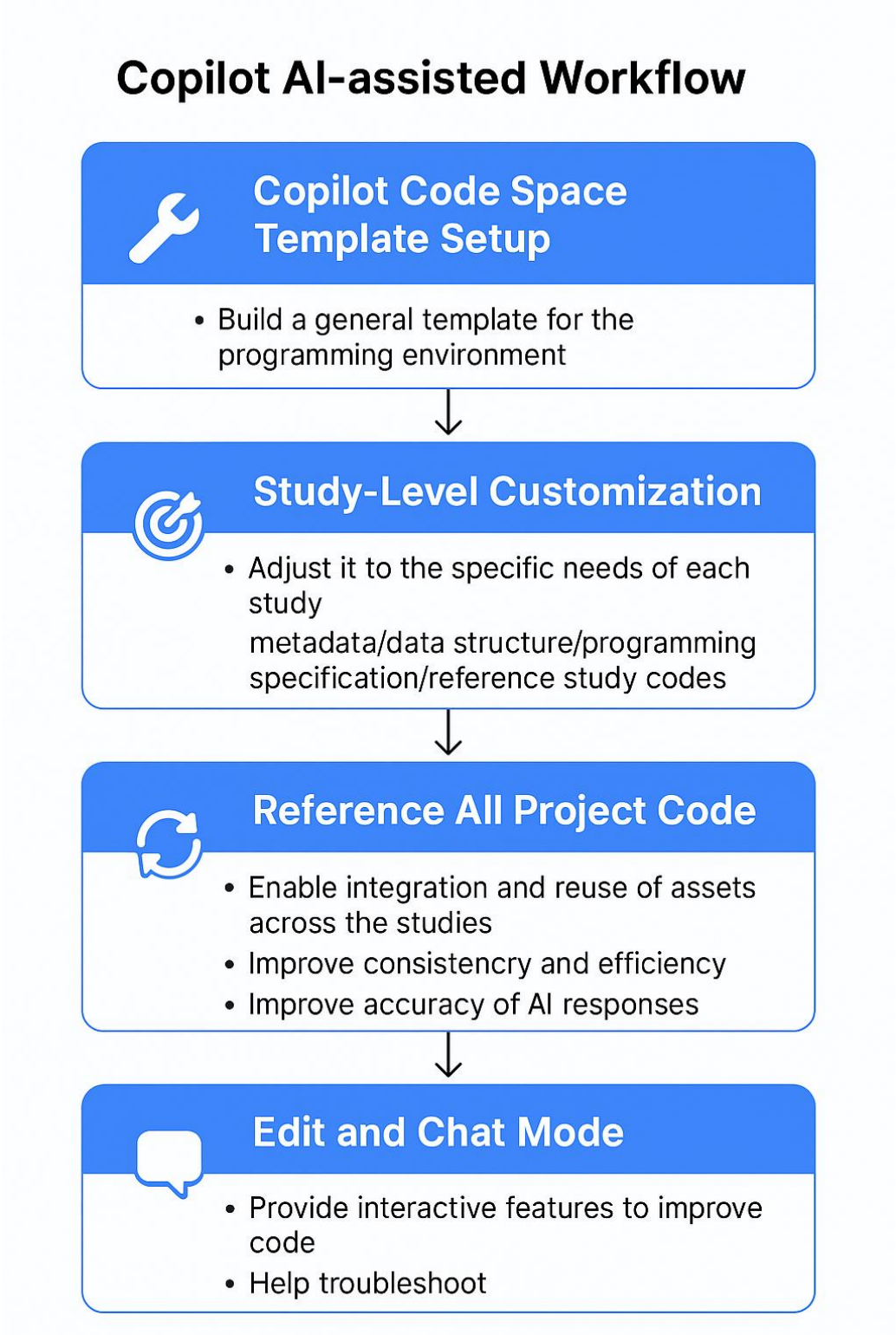
Figure 1, A diagram showing the relationship between a code editor and a codespace running on an Azure virtual machine.:



COPILOT AI ASSISTANCE WORKFLOW

For the Copilot AI-assisted workflow, we will break down the process in detail. First is the Copilot code space template setup, which is used to build a general template for the basic programming environment. Next is the study-level customization, focusing on how we adjust it to the specific needs of each study/analysis to ensure a tailored solution which will include metadata/data structure/programming specification/reference study codes. The "Reference All Project Code" section shows how AI facilitates fully integration and reuse of assets across the studies, thereby improving consistency and efficiency and improving the accuracy of AI responses. Finally, the "Edit and Chat Mode" section shows how Copilot AI empowers programmers through interactive dynamic features to help them improve their code and troubleshoot.

Figure 2, A diagram showing Copilot AI-assisted Workflow:



COPILOT CODESPACE TEMPLATE SETUP

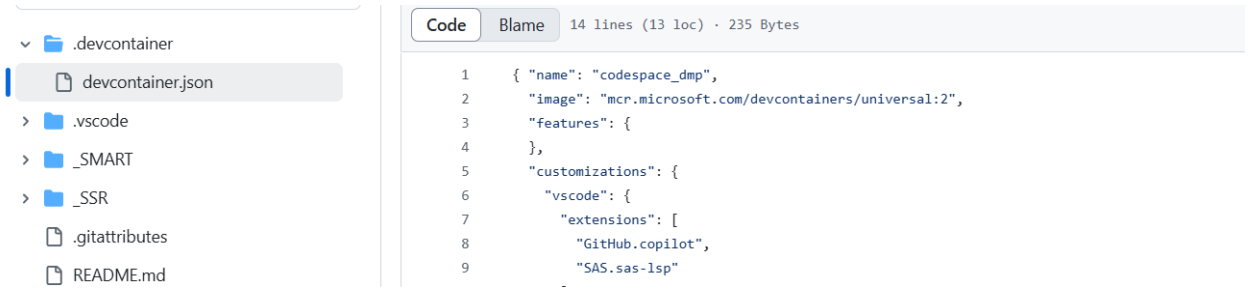
To ensure a unified and scalable programming environment across multiple studies, we first developed a Copilot Codespace template. This serves as a standardized, development environment where Copilot AI is pre-configured and ready to assist. The setup includes predefined libraries, metadata loaders, and folder structures, designed specifically for clinical data management.

We created a .devcontainer configuration folder within the GitHub repository, which includes:

- devcontainer.json to specify settings such as extensions (e.g., GitHub Copilot, SAS-lsp extension), environment variables, and post-creation scripts.
- Default study template folders, such as /data/raw, /pgm, and /output, which promote consistency across study setups.
- Integrated SOP-based code snippets and macro libraries, enabling Copilot AI to generate SOP-conformant suggestions from the start.

Once the templates are finalized, any new study can launch a GitHub Codespace using this image. Copilot AI immediately becomes functional within this shared workspace, accelerating setup and reducing human configuration errors.

Figure 3, A snapshot showing a simple devcontainer.json file setup:



STUDY LEVEL CUSTOMIZE

After setting up the cloud-hosted Codespace, the next step is customizing it for each specific clinical study. This includes:

- Loading study specific metadata, such as data structure, and programming specifications.
- Importing reference codes and study specific metadata from other studies to improve AI context alignment.

Copilot AI now leverages both the base templates and new study metadata to produce more precise suggestions.

Figure 4 A snapshot showing Copilot AI generating more helpful code with using @workspace:

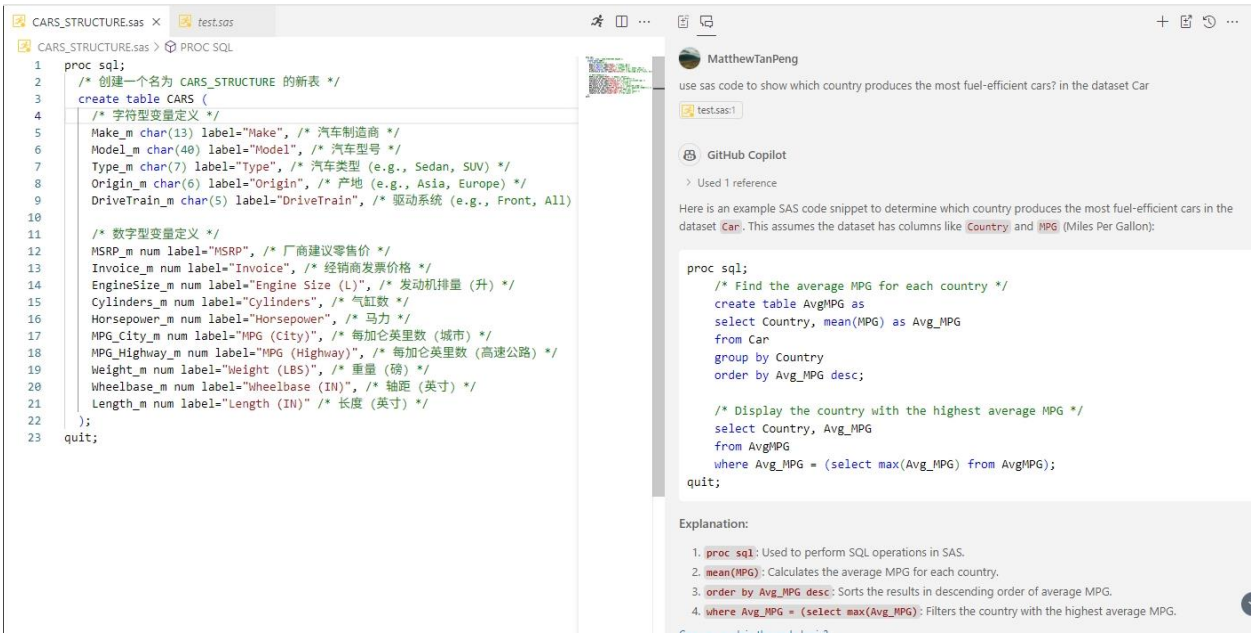
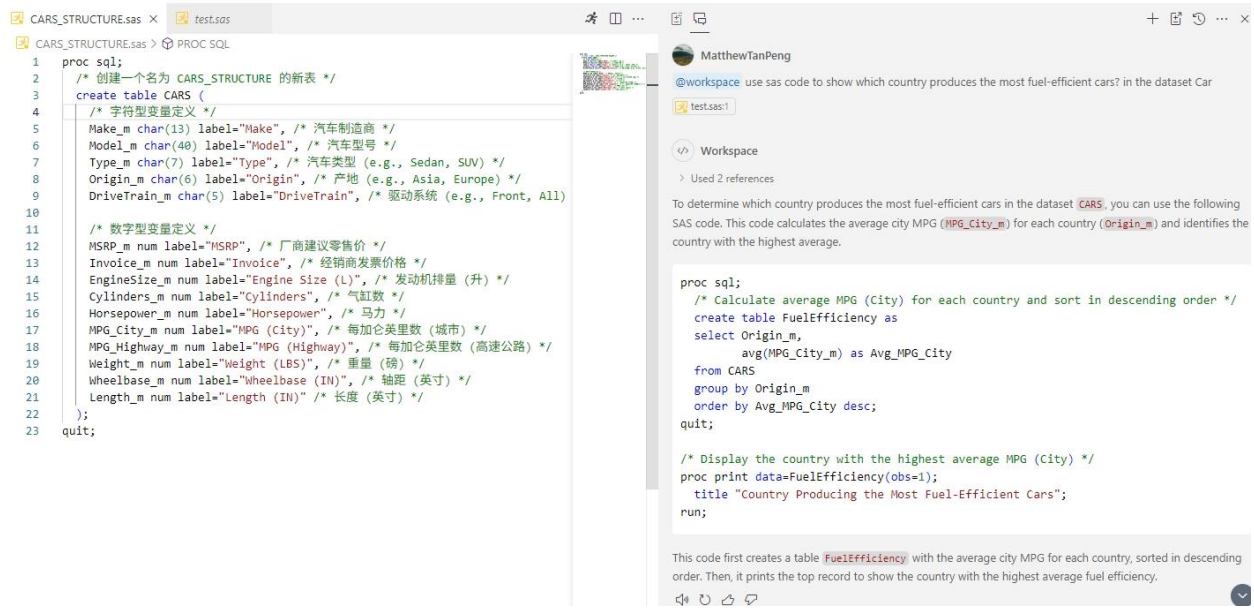


Figure 5 A snapshot showing Copilot AI to generating less helpful code without using @workspace:



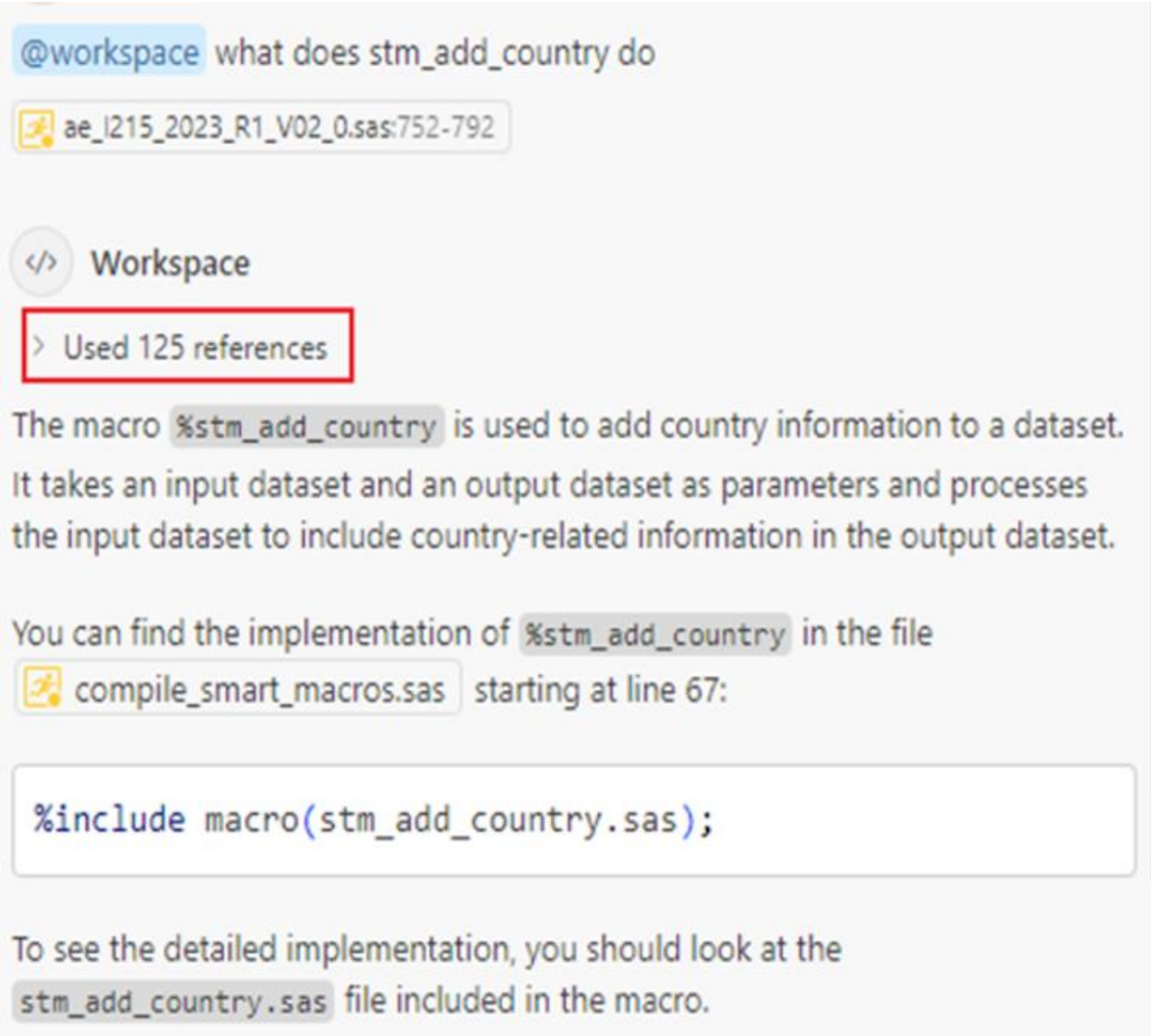
REFER ALL PROJECT CODE

To maximize reuse and maintain consistency, we centralized historical study assets into a shared Copilot-accessible project folder. This includes:

- Validated clinical data management programs (e.g., data validation program, medical listing) from prior studies.
- Custom macros and functions used repeatedly across studies.

By embedding these assets into the Copilot environment, AI can 'learn' from prior successful implementations. It can suggest standard programming logic, match previous definitions, and reduce redundancy.

Figure 6, A snapshot showing how to use @workspace letting copilot to search the whole project folder to give more accurate answer



EDIT & CHAT MODE

Once code is generated or imported, programmers enter the Edit and Chat Mode — the interactive phase where Copilot AI becomes a real-time collaborator. In this mode:

- Users can highlight a block of SAS code and type a natural-language command such as "Explain this macro" or "Optimize this merge logic".
- Copilot provides explanations, suggests refactors, or adds inline documentation automatically.
- Errors encountered during testing can be debugged by asking Copilot to "Help fix data type mismatch" or "Trace missing values in TRTSDT".

This mode transforms static editing into a dynamic conversational loop, allowing programmers to troubleshoot, improve, and document their code without leaving the development environment.

Additionally, junior programmers benefit from this as an on-demand learning tool, gaining insights into best practices directly within their workflow.

CURRENT LIMITATIONS AND POTENTIAL FUTURE IMPROVEMENTS

While the integration of Copilot AI into clinical data management programming has shown clear benefits, several limitations still exist that must be acknowledged to guide future optimization.

CURRENT LIMITATIONS

Limited Domain-Specific Understanding

Although Copilot AI can understand general programming logic, its clinical domain awareness—especially for niche therapeutic areas or sponsor-specific standards—is still developing. It may not fully grasp context-specific requirements, such as specific population flags or protocol-driven derivations, without significant user input.

Data Privacy and Regulatory Constraints

Due to strict compliance requirements (e.g., GxP, 21 CFR Part 11), sensitive clinical data cannot always be shared with external AI models. Although GitHub Copilot does not retain or learn from private code, cautious organizations may impose usage restrictions in regulated environments, limiting its utility.

Lack of Native Support for SAS-Specific Constructs

While Copilot has improved in parsing and generating SAS code, it sometimes struggles with macro-heavy workflows, PROC SQL nuance which are still common in clinical programming pipelines.

Overreliance Risk

There is a risk that programmers—especially less experienced ones—may overtrust AI-suggested code without rigorous validation. This could lead to silent errors if outputs are not carefully reviewed against specifications and regulatory expectations.

POTENTIAL IMPROVEMENTS

Domain-Specific Fine-Tuning

Fine-tuning Copilot models on validated, de-identified clinical codebases could significantly improve accuracy and relevance. This would allow it to better understand CDISC standards, common derivation logic, and sponsor-specific conventions.

Expanded Native SAS Support

Deep integration with SAS environments—such as support for SAS logs, macro debugging, or direct QC comparisons—would enhance utility for the many organizations still relying heavily on SAS. The SAS Viya Copilot seems to be a good choice in the future.

AI Validation Assistants

A future enhancement could include Copilot extensions that validate output datasets or code against specifications (e.g., dataset-level checks, variable-level consistency, automated define.xml generation), offering a full AI-assisted development and validation pipeline.

On-Prem Deployment Options

For highly regulated environments, offering a self-hosted or on-prem version of Copilot with restricted model access could satisfy compliance teams while retaining the benefits of AI assistance.

CONCLUSION

As clinical trials continue to grow in complexity and scale, the demand for more efficient, accurate, and consistent data management programming becomes increasingly urgent. GitHub Copilot, as an AI-powered coding assistant, offers a transformative opportunity to rethink traditional workflows in clinical programming. By integrating Copilot within GitHub Codespace environments, teams can accelerate study setup, and improve code quality through intelligent suggestions and real-time interaction.

Our implementation demonstrates that AI-assisted programming not only shortens development timelines but also enhances programming consistency across studies. Through structured templates, reference code reuse, and dynamic chat-based support, Copilot enables programmers to focus more on logic validation and study-specific nuances, rather than boilerplate coding tasks.

While challenges remain—such as limited SAS-specific intelligence, data privacy considerations, and domain sensitivity—the potential for future enhancement is significant. With continuous development, deeper integration into regulatory frameworks, and improved domain training, Copilot AI can evolve into a trusted co-developer in clinical research settings.

Ultimately, Copilot represents more than a tool—it is a step toward the future of AI-enabled, agile clinical data management programming, where speed, accuracy, and compliance can coexist within a collaborative digital environment.

REFERENCES

Github Copilot. https://github.com/features/copilot

Github Codespace https://docs.github.com/en/codespaces

ACKNOWLEDGMENTS

I would like to express my gratitude to the people who helped me to write this paper: Weina Jia for the encouragement and inspiration. Kai Yan for the review and comment upon the strategy.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Peng Tan
Sanofi
39F, Building 3, No. 1199 North Section of Tianfu Avenue Chengdu High-tech District, Chengdu City, Sichuan, China
Matthew.tan@sanofi.com