

From ChatGPT to Copilot: Evolving AI Support in SAS and Beyond

Jyoti (Jo) Agarwal, Gilead Sciences

ABSTRACT

Building on insights from PharmaSUG 2025 Paper SI-294, which demonstrated ChatGPT as a transformative assistant for SAS programming workflows, this paper expands the discussion to a broader ecosystem of AI-powered tools, focusing on GitHub Copilot and Microsoft Copilot and their practical applications in statistical programming.

As the pharmaceutical industry moves beyond prompt-based experimentation, Copilot tools are redefining how programmers write, debug, and optimize code across SAS, R, Python, and other analytical platforms. This paper presents real-world use cases where Copilot enhances productivity in clinical trial programming, including automating repetitive tasks, generating structured documentation, and enabling cross-language interoperability.

Unlike standalone conversational AI, Copilot tools are embedded within development and enterprise environments, offering real-time code suggestions, contextual error detection, and workflow-aware assistance aligned with clinical data standards such as SDTM and ADaM.

Through comparative analysis, this paper clarifies the complementary roles of ChatGPT, Microsoft Copilot, and GitHub Copilot. It also introduces refined AI interaction strategies for generating accurate, reproducible, and audit-ready outputs. Finally, it addresses critical considerations such as data privacy, model bias, and validation protocols required for deploying AI tools in regulated environments.

The transition from ChatGPT to Copilot represents a shift from asking AI questions to working alongside AI in real time. The future of clinical programming is not just faster code; it is smarter, compliant, and collaborative development powered by AI.

DISCLAIMER

The views expressed in this paper are solely those of the author and do not represent the views of the author's employer. No proprietary or confidential clinical trial data is used.

INTRODUCTION

The pharmaceutical industry continues to explore innovative technologies to enhance efficiency, quality, and compliance in clinical trial programming. Generative AI tools, such as ChatGPT, have demonstrated significant value in improving SAS programming workflows, as highlighted in PharmaSUG 2025 Paper SI-294 ("A Game Changer for Efficient SAS Programming Using ChatGPT").

However, the landscape of AI-assisted programming is rapidly evolving. The shift from standalone, prompt-based AI tools to embedded AI copilots marks a fundamental transformation in how programmers interact with technology.

This paper explores this evolution across three dimensions:

- *From prompt-based assistance -> embedded copilots*
- *From individual productivity -> enterprise workflows*
- *From code generation -> end-to-end lifecycle support*

AI is no longer just a tool. It is becoming an embedded collaborator in programming workflows across SAS,

R, and Python environments.

This paper explores how AI support has evolved, compares the strengths of ChatGPT and Copilot tools, and provides practical examples of how these technologies enhance clinical programming workflows. This paper shifts the perspective from AI as a standalone tool to AI as an embedded collaborator within clinical programming workflows.

OVERVIEW OF AI TOOLS IN PROGRAMMING

AI tools can be grouped into three functional categories:

CONVERSATIONAL AI (CHAT-BASED MODELS)

Examples: ChatGPT, Claude, Gemini

- Ideal for ideation, documentation, and generating code from natural language
- Strong at explaining concepts and producing examples
- Limited context awareness beyond the prompt

IDE-INTEGRATED AI (DEVELOPER-FOCUSED)

Examples: GitHub Copilot, Amazon Q Developer

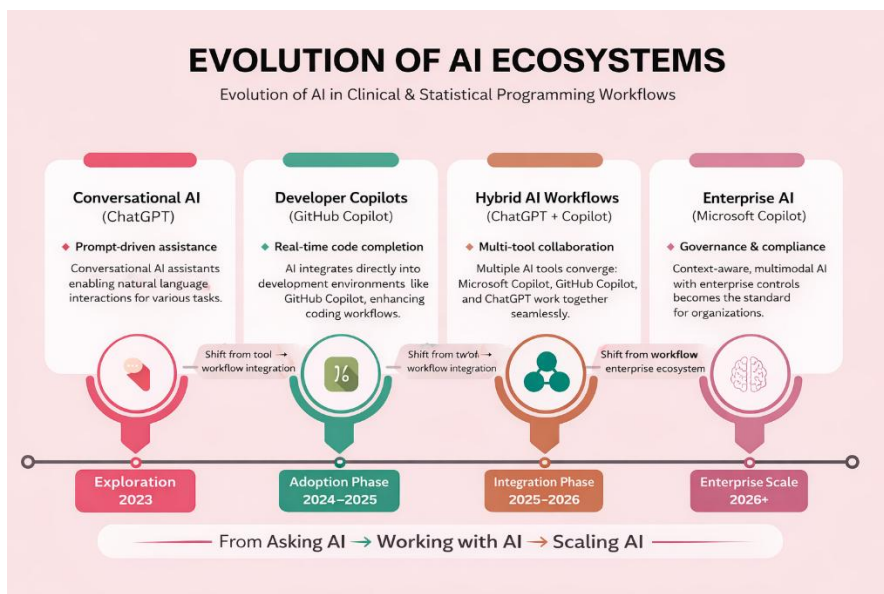
- Embedded in VS Code, PyCharm, Jupyter, and other IDEs
- Provides real-time code suggestions
- Learns from repository context, file structure, and coding patterns

ENTERPRISE-EMBEDDED AI (WORKFLOW-FOCUSED)

Examples: Microsoft 365 Copilot

- Integrated across Windows, Office, Teams, Outlook, SharePoint
- Supports documentation, analysis, summarization, and regulatory workflows
- Bridges programming tasks with enterprise communication and reporting

FIGURE 1: EVOLUTION OF AI ECOSYSTEMS (CONCEPTUAL DIAGRAM)



WHY EXPAND BEYOND CHATGPT?

While ChatGPT remains a powerful tool, several limitations emerge in production environments:

LIMITATIONS OF CONVERSATIONAL AI

- Requires manual prompting
- Limited awareness of project context or full codebase
- Higher risk of hallucinations without constraints
- Not optimized for real-time coding

ADVANTAGES OF IDE-INTEGRATED TOOLS

GitHub Copilot provides:

- Real-time code completion
- Repository-aware suggestions
- Inline error detection
- Pattern-based learning from existing code

ADVANTAGES OF ENTERPRISE-EMBEDDED TOOLS

Microsoft Copilot provides:

- Document generation (ADRG, SAP snippets, QC notes)
- Email drafting for programming clarifications
- Integration with Excel, PowerPoint, Teams
- Ability to summarize SAS logs, specifications, and outputs

THE SHIFT FROM PROMPTING -> COLLABORATING

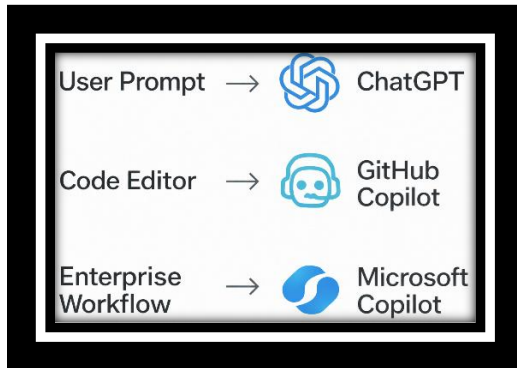
Copilot tools operate as continuous collaborators, not just prompt responders.

COMPARATIVE CAPABILITIES: CHATGPT VS MICROSOFT COPILOT VS GITHUB COPILOT

COMPARISON TABLE

Feature/ Capability	ChatGPT	GitHub Copilot	Microsoft Copilot
Primary Strength	Ideation, documentation, code generation	Real-time coding, repository context	Enterprise workflows, documentation, analysis
Integration	Browser/app	IDEs (VS Code, PyCharm)	Windows, Office, Teams
SAS Support	Strong via prompts	Moderate (via extensions)	Strong for documentation, analysis
R/Python Support	Strong	Very strong	Strong
Context Awareness	Prompt-limited	File/ repository-aware	Document/system-aware
Best Use Cases	Learning, drafting, exploring	Coding, debugging, refactoring	Documentation, reporting, compliance

FIGURE 2: AI WORKFLOW INTEGRATION (DIAGRAM)



PRACTICAL USE CASES AND REAL-WORLD EXAMPLES

Modern AI copilots such as **GitHub Copilot** and **Microsoft Copilot** provide real-time, context-aware assistance directly within SAS development environments. Unlike ChatGPT, which requires copying code into a chat interface; Copilot operates *inline*, reading the surrounding code, indentation patterns, and project structure. This allows Copilot to generate cleaner, more consistent, and more workflow-aware SAS programs that align with clinical programming standards such as SDTM, ADaM, and TLF conventions.

The following examples demonstrate how Copilot enhances SAS workflows across common clinical programming tasks. Each example mirrors the structure of the author's 2025 ChatGPT paper but expands the depth, clinical relevance, and Copilot-specific capabilities.

COPILOT VS CHATGPT IN PRACTICE

While ChatGPT excels at generating standalone code examples, Copilot's strength lies in its ability to operate inside the programming environment. This allows Copilot to read the surrounding code, understand indentation patterns, and generate context-aware suggestions that align with project standards. As a result, Copilot's output is often more consistent, more maintainable, and more aligned with clinical programming workflows. The following examples illustrate how Copilot enhances SAS programming tasks beyond what conversational AI alone can provide.

PROMPT 1: REFORMATTING SAS CODE FOR BEST PRACTICES

Scenario: *"Could you reformat the code to follow good programming practices without adding anything to the code?"*

Copilot automatically reformats SAS code directly inside the editor, applying indentation, spacing, and alignment consistent with clinical programming standards. Because Copilot reads the entire file, it formats code in a way that matches the project's existing style, improving readability and QC efficiency.

```

Sas

proc sql noprint;

    /* Create patient-level dataset */
    create table patient_data as
    select *
    from clinical_trial
    order by treatment_group, visit_day;

    /* Create summary of maximum response by treatment group */
    create table max_response_summary as
    select
        a.treatment_group,
        b.visit_day,
        a.max_response_value,
        b.total_response_value
    from
        (select
            treatment_group,
            max(response) as max_response_value
        from patient_data
        group by treatment_group) as a
    left join
        (select
            treatment_group,
            visit_day,
            sum(response) as total_response_value
        from patient_data
        group by treatment_group, visit_day) as b
    on a.treatment_group = b.treatment_group;

quit;

```

Copilot's inline formatting reduces review time and improves traceability, critical in regulated environments.

SCENARIO 2: ALTERNATIVE APPROACHES WITHOUT PROC SQL

Scenario: “Could you do the same task the above code is doing without using PROC SQL?”

Copilot can translate SQL logic into DATA step and PROC-based alternatives, which is valuable for organizations that prefer non-SQL code for transparency, performance, or legacy standards.

Copilot-Generated DATA Step Equivalent

```

Sas

proc sort data=clinical_trial out=patient_data;
    by treatment_group visit_day;
run;

/* Maximum response by treatment group */
proc means data=patient_data nway noprint;
    class treatment_group;
    var response;
    output out=max_resp(drop=_) max=max_response_value;
run;

/* Total response by treatment group and visit */
proc means data=patient_data nway noprint;
    class treatment_group visit_day;
    var response;
    output out=total_resp(drop=_) sum=total_response_value;
run;

/* Merge results */
data max_response_summary;
    merge max_resp total_resp;
    by treatment_group;
run;

```

This demonstrates Copilot's ability to convert logic across SAS paradigms while preserving analytical intent.

SCENARIO 3: ENHANCING CODE WITH COMMENTS

Scenario: “Could you add some comments to the code to help me understand?”

Copilot can automatically generate clear, audit-friendly comments aligned with CDISC programming expectations. Comments are contextual, based on variable names, dataset structure, and common ADaM derivation patterns.

```
Sas

/* Sort input data by subject and visit */
proc sort data=adlb;
  by usubjid visitnum;
run;

/* Derive baseline and change from baseline */
data adlb_der;
  set adlb;
  by usubjid;

  /* Capture baseline value (VISITNUM = 0) */
  if visitnum = 0 then base = aval;
  retain base;

  /* Carry baseline forward for post-baseline visits */
  if visitnum > 0 then chg = aval - base;
run;
```

This improves clarity for QC reviewers and future programmers.

SCENARIO 4: IMPLEMENTING LOCF (LAST OBSERVATION CARRIED FORWARD)

Scenario: “Could you write a program on how to implement LOCF?”

Copilot recognizes LOCF as a common imputation method in ADaM BDS datasets and generates a clean implementation.

```
Sas

proc sort data=adslab out=lab_sorted;
  by usubjid visitnum;
run;

data lab_locf;
  set lab_sorted;
  by usubjid;

  retain lastval;

  /* Update last non-missing value */
  if not missing(aval) then lastval = aval;

  /* Carry forward when missing */
  if missing(aval) then aval = lastval;
run;
```

This aligns with ADaM guidance for handling intermittent missing values.

SCENARIO 5: IMPLEMENTING LOCF IN MULTIPLE LANGUAGES

Scenario: “Write a SAS program that demonstrates how to implement LOCF in SAS, Python, R, and

SQL.”

Copilot supports cross-language workflows, enabling programmers to validate logic across platforms.

SAS

```
Sas Copy  
aval = coalesce(aval, lag(aval));
```

Python (pandas)

```
Python Copy  
df['aval'] = df.groupby('USUBJID')['aval'].ffill()
```

R

```
R Copy  
library(dplyr)  
df %>% group_by(USUBJID) %>% tidyr::fill(aval, .direction = "down")
```

SQL (Window Function)

```
Sql Copy  
LAST_VALUE(aval IGNORE NULLS) OVER (PARTITION BY usubjid ORDER BY visitnum)
```

This supports multi-platform validation and reproducibility.

SCENARIO 6: CONVERTING LINUX EGREP FUNCTION TO SAS

Scenario: “Write a SAS program to convert the `egrep` Linux function.”

Copilot uses PRX functions to replicate pattern-matching behavior.

```
Sas  
  
data matches;  
  set rawtext;  
  if prxmatch('/error|warning|critical/i', text_line);  
run;
```

This is particularly useful for log scanning and automated QC.

SCENARIO 7: CREATING AN SGPLOT FOR MEDIAN (Q1, Q3) BY TREATMENT AND VISIT

Scenario: “Generate SGPlot code to display Median (Q1, Q3) for CHG by TRT and Visit.”

```
Sas Copy
proc sgplot data=lab_summary;
  title "Median (Q1, Q3) Change from Baseline by Treatment and Visit";

  series x=visit y=median / group=trt lineattrs=(thickness=2);
  highlow x=visit low=q1 high=q3 / group=trt transparency=0.2;

  yaxis label="Change from Baseline";
  xaxis label="Visit";
run;
```

Copilot also suggests color palettes and layout improvements.

SCENARIO 8: CREATING A BOXPLOT IN SAS

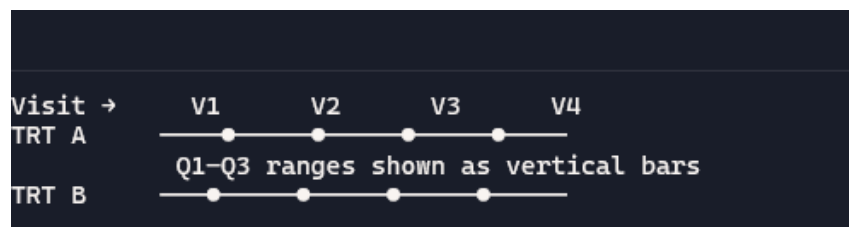
Scenario: “Generate a boxplot of age by actual arm.”

```
Sas
proc sgplot data=adsl;
  title "Distribution of Age by Actual Arm";
  footnote "Boxplot generated using Copilot-assisted SAS code";

  vbox age / category=actarm fillattrs=(color=cx7BAFD4);
run;
```

SCENARIO 9: GENERATING A MOCK-UP GRAPH

Scenario: “Copilot can generate ASCII mock-ups for early design discussions.”



This accelerates communication with statisticians and medical writers.

SCENARIO 10: SUMMARY STATISTICS AND EXPORT TO EXCEL

Scenario: “Generate SAS code to compute summary statistics and export to Excel.”

```
Sas
proc means data=adsl n mean median std;
  var age weight height;
  output out=summary_stats;
run;

proc export data=summary_stats
  outfile="summary_stats.xlsx"
  dbms=xlsx
  replace;
run;
```

Copilot can also generate Excel formatting instructions or PowerPoint-ready tables.

COPILOT STRENGTH HIGHLIGHT

Copilot recognized the BDS structure and automatically generated a baseline-carry-forward pattern consistent with ADaM conventions.

These examples demonstrate how Copilot supports not only code generation but also documentation, visualization, and cross-language interoperability. By operating within the programming environment, Copilot provides workflow-aware assistance that enhances productivity, improves code quality, and accelerates clinical deliverables.

PROMPT ENGINEERING AND BEST PRACTICES

EFFECTIVE PROMPT DESIGN

- Provide dataset structure
- Specify CDISC standards
- Include constraints (e.g., “no PROC SQL”)
- Request comments or QC notes

HUMAN OVERSIGHT

- Validate assumptions
- Review logic for compliance
- Ensure reproducibility

AUDIT-READY OUTPUTS

- Require versioned prompts
- Document AI involvement
- Store generated code in repositories

COMPLIANCE, GOVERNANCE, AND REGULATORY CONSIDERATIONS

AI adoption in pharma requires:

DATA PRIVACY

- No patient-level data in prompts
- Use enterprise-approved AI tools

BIAS MITIGATION

- Validate outputs against specifications
- Avoid relying solely on AI-generated logic

AUDIT TRAILS

- Maintain version control

- Capture prompt history

MODEL VALIDATION

- Treat AI as a tool requiring qualification
- Establish SOPs for AI-assisted programming

WORKFLOW INNOVATION ROADMAP & FUTURE VISION

HYBRID PROGRAMMING ENVIRONMENTS

- SAS + R + Python workflows
- AI-assisted metadata-driven programming

PREDICTIVE QC

- AI-flagged anomalies
- Automated log review

AI-AUGMENTED COLLABORATION

- Copilot summarizing team discussions
- Automated documentation generation

CONCLUSION

The evolution from ChatGPT to Copilot represents a shift from isolated AI assistance to integrated, workflow-level intelligence. SAS programmers now benefit from contextual, multimodal, and cross-platform support that enhances productivity, improves code quality, and accelerates analysis.

The future of statistical programming is not just about writing code faster; it is about building smarter, safer, and more collaborative workflows powered by AI. The future is not AI replacing programmers, but AI elevating them, enabling statistical programmers to focus on scientific insight, quality, and innovation.

REFERENCES

Agarwal, J. 2025. "SI-294 SAS Programming Efficiency." PharmaSUG 2025 Proceedings.
OpenAI. (2025). ChatGPT and AI in Programming. OpenAI. Available at <https://openai.com>.
Statista Research. (2025). AI Adoption in Software Development. Statista.
GitHub Copilot Documentation
Microsoft Copilot Documentation

ACKNOWLEDGMENTS

The author acknowledges the evolving AI ecosystem and the insights gained through practical applications of ChatGPT and Copilot tools.

CONTACT INFORMATION

Comments and questions are valued and encouraged. Contact the author at:
Jyoti (Jo) Agarwal
www.linkedin.com/in/joagarwal/