Paper SD04

AI Code Generator: Automating TFL Programming with ARS Metadata

Navin Dedhia, Clymb Clinical, Seattle, WA, United States
Bhavin Busa, Clymb Clinical, Burlington, MA, United States

ABSTRACT

TFL programming has traditionally required extensive manual coding in SAS and R, consuming valuable time and resources. To address this, we developed the AI Code Generator, an LLM/GenAI-based solution that leverages CDISC Analysis Results Standard (ARS) metadata within the TFL Designer tool to automatically generate production-ready TFL code in both SAS and R. Early evaluations demonstrate more than 70% reduction in time for TFL output generation, underscoring the potential of AI-assisted programming to streamline clinical reporting workflows. The premise is simple: design and specify early in the process, with metadata and tools serving as the key drivers of efficiency. In this demo, we will present a live case study showing how table programs can be generated in minutes, quickly modified, and applied to study data to produce outputs, transforming the way statistical programmers' approach TFL development.

INTRODUCTION

Clinical trial reporting depends heavily on the accurate generation of Tables, Figures, and Listings (TFLs). Traditional TFL programming workflows are labor-intensive and are prone to variability and human error. The growing maturity of Generative AI presents an opportunity to modernize these workflows while maintaining regulatory rigor.

This paper introduces an AI Code Generator embedded within TFL Designer that leverages GenAI and CDISC ARS metadata to automate SAS/R program generation. The framework emphasizes metadata-driven automation, traceable logic, and reproducible outputs suitable for regulated clinical environments.

BACKGROUND AND MOTIVATION

Statistical programming teams must translate protocol requirements, statistical analysis plans, and shell specifications into executable code. This process involves interpretation, coding, validation, and review, often repeated across studies. While CDISC ARS provides a structured model for representing analysis intent, adoption is still evolving. Integrating ARS metadata with AI-driven reasoning allows automation to occur at a semantic level rather than simple template reuse.

The motivation behind this solution is to reduce repetitive manual coding while improving consistency, auditability, and scalability across studies.

IMPLEMENTATION WORKFLOW

The implementation of the AI Code Generator follows a structured, metadata-driven workflow designed to integrate automation with regulatory-grade controls. The process begins in TFL Designer (Figure 1), where study teams author ARS-aligned metadata describing populations, derivations, statistical methods, and presentation logic. This metadata serves as the authoritative specification layer and is programmatically extracted through controlled APIs to form the contextual foundation for code synthesis (Figure 2).

The retrieval-augmented generation (RAG) pipeline enriches these structured inputs with validated reference artifacts and organizational programming assets, ensuring that generated logic is grounded in domain-approved standards.

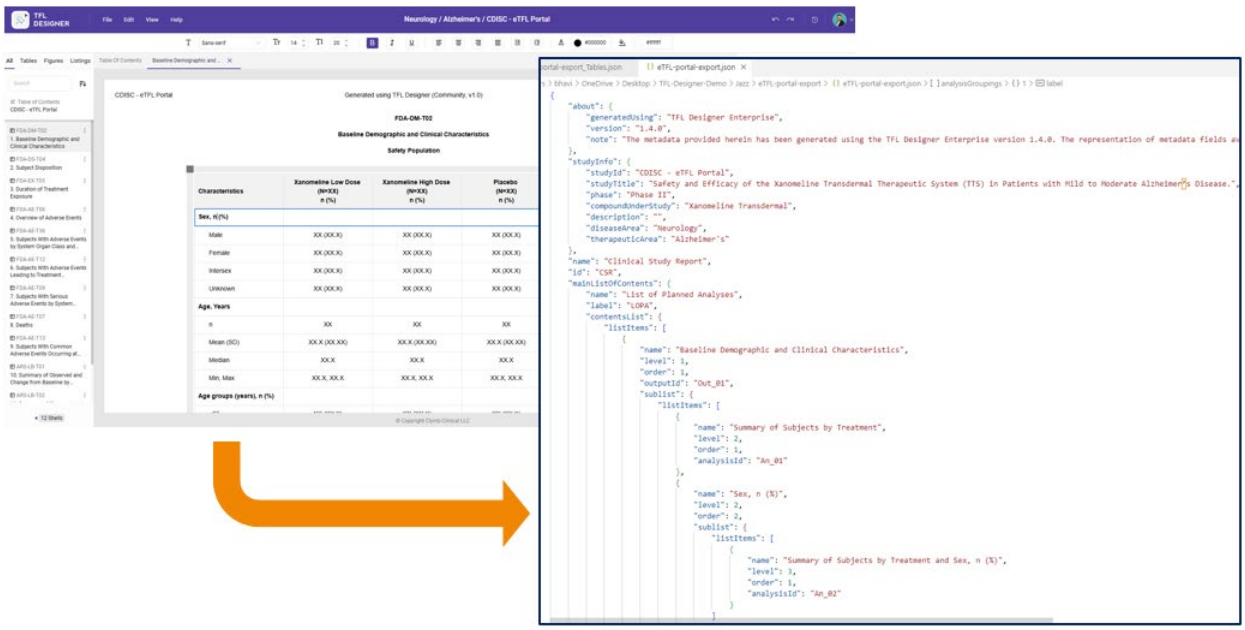

Figure 1: TFL Shells + CDISC ARS Metadata (JSON) from TFL Designer

Using this enriched context, the AI code synthesis layer produces SAS or R programs tailored to the study's TFL requirements. Generated programs are not executed directly within the AI environment; instead, they are exported to the Statistical Computing Environment (SCE), where execution occurs under established validation controls. This separation preserves the integrity of existing execution infrastructure and aligns with regulatory expectations for controlled runtime environments. Within SCE, programs are compiled, executed, and subjected to conventional quality checks, enabling organizations to leverage AI-generated logic without altering validated execution workflows.

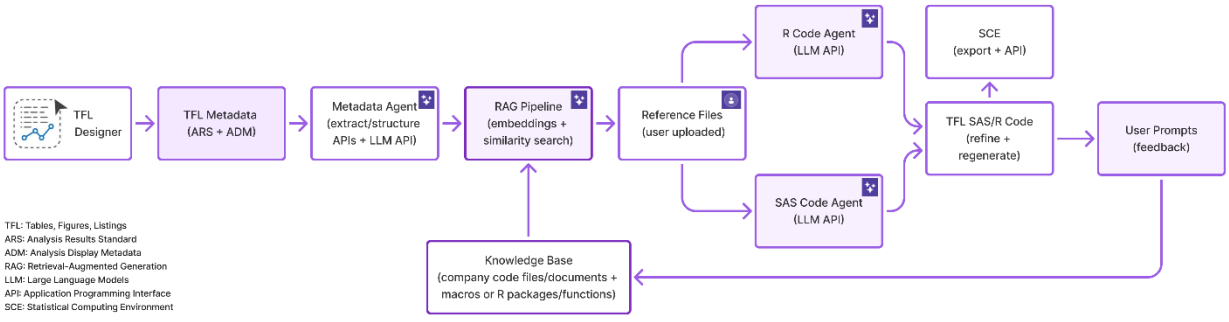

Figure 2: AI Code Generator Architecture

Validation is embedded throughout the lifecycle to support traceability, reproducibility, and audit readiness. Each generated code artifact is tagged with metadata linking it back to originating ARS definitions, enabling bidirectional traceability between specifications, code, and outputs. Automated validation checks assess structural correctness, naming conventions, dependency resolution, and conformance to programming standards before execution.

Version control mechanisms capture model selection, prompt context, regeneration history, and user refinements, forming a transparent audit trail consistent with GxP documentation expectations. Human oversight remains integral: programmers and reviewers can inspect, refine, and approve generated code prior to execution, reinforcing accountability and validation discipline.

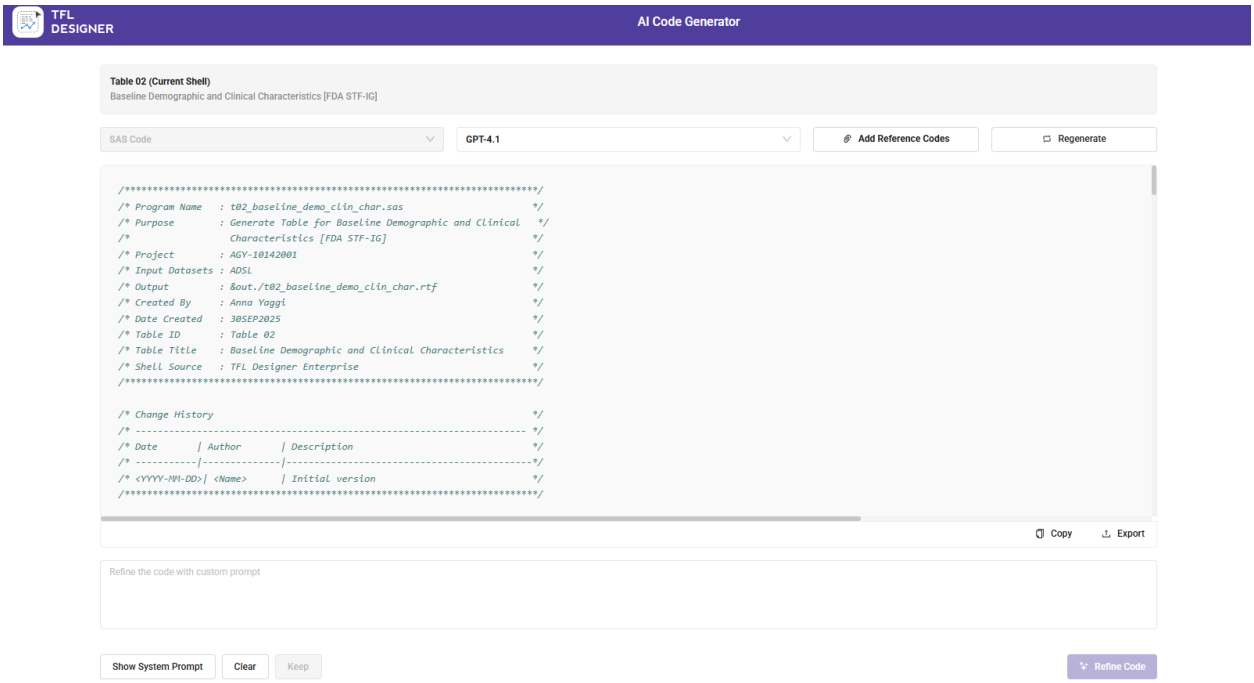

Figure 3: AI assisted SAS code generation using ARS and analysis display metadata

CASE STUDY / BENEFITS AND IMPACT

Background

To quantify the efficiency gains and practical impact of AI-assisted code generation, a controlled case study was conducted to compare traditional statistical programming workflows with the AI Code Generator integrated into TFL Designer (Figure 3). The evaluation focused on standard clinical Table, Figure, and Listing (TFL) development using SAS programming, reflecting common production practices in regulated clinical trial reporting environments. The objective was to assess whether metadata-driven, GenAI-assisted code generation could reduce development time while preserving consistency, predictability, and output quality.

This case study intentionally focused on primary programming activities only, excluding independent validation, QC programming, and formal review cycles, in order to isolate the impact on initial code development effort.

Testing Approach

The study evaluated the generation of 25 summary tables representing standard safety and efficacy outputs, including both comparative and inferential statistical analyses, under a controlled setup using three distinct programming approaches noted below. These tables reflect commonly produced clinical trial deliverables and encompass a range of analysis complexity, allowing for a representative comparison of programming effort, consistency, and efficiency across traditional and AI-assisted workflows.

- **Traditional Programming with Existing Company Macros**, where programmers reviewed TFL shells and manually developed SAS programs using established macro libraries, setup files, and standard PROC REPORT logic.
- **AI Code Generator without Macros**, in which the AI Code Generator produced SAS programs using ARS-aligned Clymb JSON metadata as the primary specification layer, without leveraging company-specific macro libraries.
- **AI Code Generator with Clymb Macros**, where the AI Code Generator combined ARS-aligned Clymb JSON metadata with existing, validated macro libraries and standard RTF style macros.

Across all approaches, titles and footnotes were sourced from TFL Designer exports, and minor post-processing was performed where needed to align final formatting. Generated code was exported to the Statistical Computing Environment (SCE) and executed under standard controls. The evaluation emphasized consistency of structure, maintainability, and suitability of generated code for production use.

Table 1: Comparison of SAS Programming Effort for 25 Standard Summary Tables

Programming Approach	Description	Time to Complete 25 Outputs (Hours)	Time Reduction vs. Traditional
Traditional Programming with Existing Company Macros	Manual development using established macro libraries, setup files, and standard PROC REPORT logic	**135**	Reference
AI Code Generator (No Macros)	AI-generated SAS code using ARS-aligned Clymb JSON metadata without company macro libraries	**60**	~56%
AI Code Generator (With Clymb Macros)	AI-generated SAS code using ARS-aligned Clymb JSON metadata integrated with existing macro libraries	**32**	~76%

As shown in Table 1, use of the AI Code Generator resulted in substantial reductions in SAS programming time for 25 clinical outputs. Outputs generated using all three programming approaches were reviewed for accuracy to ensure consistency and correctness of the analytical results.

Compared with traditional programming using existing company macros, the AI Code Generator reduced development effort by approximately 56% without macro integration and by 76% when combined with established macro libraries. These gains were achieved while maintaining consistent output structure and formatting. The results highlight the additive value of combining metadata-driven AI code generation with existing, validated programming assets to deliver predictable efficiency gains in regulated statistical programming workflows.

Key Benefits

- Predictable and controlled outcomes compared with ad-hoc AI-generated code, supported by metadata-driven specifications and standardized programming assets.
- Ability to handle complex analyses, including safety and efficacy outputs as well as inferential and comparative statistical methods.
- Efficient debugging and long-term maintainability through seamless integration with existing, validated macro libraries.
- Configurable to Sponsor-specific programming conventions, including headers, setup files, macro usage, and PROC REPORT logic.
- Progressive learning of code styling and implementation patterns over time based on the organization’s standard programming library.

Impact

- Accelerates statistical programming deliverables, reducing development timelines for large TFL packages.
- Frees statistical programmers for higher-value activities, such as complex analysis development, review, and interpretation.
- Introduces scalable automation and intelligence into statistical programming workflows, while preserving traceability, reproducibility, and regulatory alignment.

- Generated code suitable for production, validation, or QC purposes, enabling reuse across multiple stages of the programming lifecycle.

CONCLUSION

This paper demonstrates how combining Generative AI with CDISC ARS metadata can significantly improve the efficiency of TFL programming while maintaining the controls required in regulated clinical research environments. By using metadata as the primary driver and keeping execution within established statistical computing environments, the AI Code Generator enables faster and more consistent generation of SAS programs without compromising traceability, transparency, or compliance.

While the effectiveness of the approach depends on the quality and completeness of ARS metadata and continues to require appropriate human review, the results show clear benefits for standard TFL development. The solution functions as an assistive capability that reduces repetitive coding effort and allows statistical programmers to focus on higher value analytical and review activities.

Overall, the AI Code Generator represents a practical and scalable step toward modernizing statistical programming workflows. As ARS adoption continues to mature and supporting tools evolve, metadata driven and AI assisted programming has the potential to play an increasingly important role in delivering faster, more consistent, and more transparent clinical reporting.

REFERENCES

1. Busa Bhavin. The AI Reckoning: Why Statistical Programmers Must Evolve Now. PHUSE US Connect, March 2026.
2. CDISC Analysis Results Standard (ARS), Version 1.0, April 2024. https://cdisc-org.github.io/analysis-results-standard/
3. CDISC Analysis Results Standard User Guide, Version 1.0. https://wiki.cdisc.org/display/ARSP/Analysis+Results+Standard+User+Guide+v1.0
4. TFL Designer: https://clymbclinical.com/tfl-designer/
5. AI Code Generator: https://clymbclinical.com/ai-code-generator/

ACKNOWLEDGMENTS

We thank the contributors involved in AI Code Generator development, validation, and data processing teams at Clymb Clinical for their dedication and contributions. Their efforts have been instrumental in building an innovative solution that advances our vision of enhancing efficiency and quality in code generation.

CONTACT INFORMATION

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

Navin Dedhia
Head of Software Engineering,
Clymb Clinical
ndedhia@clymbclinical.com

Bhavin Busa
Principal & Co-founder,
Clymb Clinical
bhavin@clymbclinical.com