Paper CT08

Enhancing Stats Programming Efficiency in Pharma: Harnessing Large Language Models (LLMs) such as ChatGPT for Accelerated Development and Validation

Andy Ratcliffe, Altair, London, England

ABSTRACT

This paper demonstrates the significant advantages of integrating cutting-edge natural language processing models such as ChatGPT into statistical programming workflows within the pharmaceutical industry. By strategically leveraging a model's capabilities, this approach offers substantial gains in development speed and resource optimisation, leading to faster project timelines and reduced operational costs. Beyond mere automation, this paper showcases practical applications and innovative techniques that empower statistical programmers with intelligent assistance, streamlining complex tasks, improving code quality, and ultimately increasing their productivity and impact. This exploration highlights how embracing AI tools like ChatGPT can augment human expertise, transforming challenges into opportunities for enhanced efficiency and innovation in pharmaceutical development.

INTRODUCTION

The pharmaceutical industry faces persistent pressure to accelerate clinical trial reporting while maintaining stringent GxP (Good Practice) compliance standards. Statistical Programming, reliant on deterministic code generation (primarily using the SAS language), represents a significant bottleneck. This paper proposes a compliant framework for integrating Large Language Models (LLMs), such as the GPT-4 and Llama families, into the SAS ecosystem. This framework centres on **Human-in-the-Loop (HIL)** governance and **Retrieval-Augmented Generation (RAG)** grounded in corporate standards. We demonstrate how LLMs can drastically reduce boilerplate code generation time (estimated at 40–60% for routine tasks) and, critically, how they can be strategically utilised to support automated quality control (QC) and the generation of independent verification scripts, ensuring accelerated development does not compromise regulatory integrity.

The development and submission of clinical trial data - specifically the creation of Study Data Tabulation Model (SDTM) datasets, analysis datasets (ADaM) and Tables, Listings, and Figures (TFLs) - is highly governed. This process mandates **deterministic** programming, meaning code must execute predictably every time. Conversely, Large Language Models (LLMs) are **stochastic** (probabilistic) by nature, trained on vast amounts of text to generate the statistically most probable output. This distinction has historically been the primary barrier to their adoption in regulated GxP workflows.

This paper advocates for an operational shift: recognising the LLM not as a replacement for the statistical programmer, but as a sophisticated co-pilot designed to automate repetitive, low-value tasks. The successful integration strategy focuses on governance, security, and using the LLM's dynamic capabilities to complement, rather than undermine, existing quality systems.

THE LLM-AUGMENTED GXP FRAMEWORK

For LLMs to function securely within pharmaceutical statistical programming, two foundational pillars are required: a robust governance model and a secure technical architecture.

A. THE GOVERNANCE MODEL: HUMAN-IN-THE-LOOP (HIL)

In a GxP environment, the **Human-in-the-Loop (HIL)** model is non-negotiable. The LLM serves as an intelligent drafting tool, but the human programmer remains the Subject Matter Expert (SME) who owns the code and assumes responsibility for its final, validated state.

- **Responsibility:** The programmer validates, executes, and signs off on LLM-generated code.
- **Mitigation:** HIL directly addresses the LLM's primary weakness—the risk of "hallucination"—by mandating human review and confirmation of every line of code proposed by the model.

The HIL model can be misunderstood, and might be better understood as AI-in-the-loop. It depends on who one perceives as managing and controlling the cycle (aka loop). For the near future, it seems likely that the human will be seen as managing the loop, and using AI at certain points within that loop, so maybe we should use the term AI-in-the-loop.

B. TECHNICAL INTEGRATION WITH SAS

The deployment strategy for LLMs typically involves integrating external models into the SAS language environment. The SAS language does not primarily create foundational LLMs, but rather acts as the secured orchestration layer.

- **API Calling:** Integration is executed via REST API calls. In the SAS language, this is achieved through the **PROC HTTP** procedure (for direct calls) or the **PROC PYTHON** procedure (leveraging robust Python SDKs for models like GPT-4, Gemini, or Llama 3).

- **Security:** To maintain data sovereignty, organisations must utilise private cloud LLM deployments (e.g. AWS Bedrock and Azure OpenAI Service with private endpoints) or self-host open-access models (like the Llama family) to ensure proprietary programming specifications and clinical data **never** exit the secure organisational network.

THE COMPLIANCE GUARDRAIL: RETRIEVAL-AUGMENTED GENERATION (RAG)

Standard LLMs are trained on general internet data, which includes public-domain programming knowledge (e.g. Stack Overflow). This is insufficient for GxP code. **RAG** is essential to ground the model in organisational truth. RAG involves feeding the LLM not just the user prompt, but also relevant, trusted internal documents (SOPs, CDISC guides, validated macro libraries). This ensures that LLM-generated code adheres to the company's specific style, macro usage, and regulatory guidelines, turning a generic model into a highly specialised, compliant programming assistant.

ACCELERATED DEVELOPMENT USE CASES

The primary benefit of LLMs lies in automating the cognitive overhead associated with translating analysis specifications into working code.

A. DYNAMIC BOILERPLATE REDUCTION

While shared SAS macro libraries already automate much of the boilerplate, LLMs offer **dynamic customisation** that macros cannot.

- **Macro Limitations:** Macros provide a static, pre-written solution. Any unique customisation (e.g. adding a specific WHERE clause based on an uncommon drop-out criterion) requires the programmer to manually write or modify logic.

- **LLM Enhancement:** LLMs automate the tedious step of **translation and adaptation**. A programmer can input a complex, narrative prompt (e.g. "Generate the PROC REPORT table shell for Table X, but filter for subjects with an initial BMI greater than 30 and stratify by sex") and the LLM instantly generates the full, customised SAS code, including the necessary WHERE statements and macro parameter values. This dynamic generation drives the reported **40–60% efficiency gain** in initial drafting time.

Emerging tools such as SASBuddy[1] demonstrate the separation of execution from intelligence and the provision of real AI-assisted coding. Its RAG-augmented approach means it can assist the programmer with creating SAS language code that makes use of local resources such as common, corporate macros.

B. DEBUGGING AND DOCUMENTATION

LLMs significantly reduce time lost to troubleshooting and administrative tasks:

- **Intelligent Debugging:** Feeding SAS log errors and adjacent code snippets to an LLM via the secure RAG system allows the model to instantly suggest precise fixes, leveraging its vast training on code syntax and error patterns.

- **Specification Drafting:** LLMs excel at structuring complex text. By providing the LLM with a paragraph from the Statistical Analysis Plan (SAP) or Protocol, the model can automatically generate a detailed, GxP-compliant **Programming Specification Document**, saving programmers administrative time.

VALIDATION AND QUALITY ASSURANCE (QA)

Integrating LLMs into the GxP framework requires proving they enhance, not detract from, validation standards.

A. AUTOMATED QUALITY CONTROL (QC) AGENT

LLMs can be leveraged to automate the initial layer of code review, freeing human QA resources for scientific checks.

- **Style and Compliance Checks:** The LLM acts as an automated QC agent, reviewing programmer-written code against the RAG-fed **Style Guide** and **Programming SOPs** (e.g. checking for necessary comments, mandatory variable naming conventions, and approved macro calls).

- **Focus Shift:** This automation elevates the human QA reviewer's role from spotting syntax errors to focusing exclusively on the **statistical logic and scientific interpretation** of the code and output.

B. INDEPENDENT VERIFICATION SCRIPTS

The **Golden Rule of Validation** requires that verification code be written and executed **independently** from the development code. LLMs provide a powerful method to streamline this critical step:

- **Verification Generation:** A QA reviewer can input the specific analysis requirement (the "specification") into the LLM and prompt it to generate a **separate, single-purpose verification script** (e.g. a simple Data Step or R script) to reproduce a key calculation in the primary program's output.

- **Audit Trail:** The LLM can also be prompted to generate the necessary **Audit Trail documentation**, providing a traceable, side-by-side comparison of the developer's output, the verification script's output, and the LLM's role in generating the latter.

CONCLUSION AND FUTURE DIRECTIONS

The integration of LLMs marks a profound shift in statistical programming efficiency. By adopting a **Human-in-the-Loop** governance model and securing the LLM environment with **RAG**, organisations can leverage generative AI for accelerated development without compromising regulatory compliance.

The LLM-augmented workflow transforms the statistical programmer's role from manual typist and debugger into a high-level reviewer and data strategist. The industry's next steps will involve deeper integration into Integrated Development Environments (IDEs), full LLM-driven automation of CDISC regulatory documents (like Define-XML), and the establishment of industry-wide benchmarks for LLM accuracy and performance in GxP contexts. The future of statistical programming lies in the collaboration between human expertise and machine speed.

The key lesson to learn is that AI will not replace the statistical programmer, but the programmer who uses AI will replace the one who doesn't. The phrase echoes the sentiment from the period when mastering Google search was a crucial professional skill because both scenarios highlight a similar theme: the essential role of adopting a powerful new tool to maintain professional relevance and competitive edge. In the early 2000s, the Google-savvy professional could access information, find solutions, and learn new skills much quicker than those relying on traditional, slower methods (e.g. physical libraries, phone calls). Today, the "AI-enabled" programmer is faster, more efficient, and can handle more complex tasks than their non-AI-using counterpart.

A statistical programmer who can use an AI assistant to rapidly prototype code, debug errors, or synthesize documentation is a significantly more efficient unit of labour than one who writes every line from scratch. The AI-using programmer essentially raises the bar for productivity, making the non-user competitively disadvantaged. Being a great programmer no longer just means writing elegant code; it means knowing how to prompt and manage the AI to generate elegant code, knowing which tasks to delegate to the AI, and critically validating its output. The ability to prompt is the new meta-skill.

In essence, the phrase "AI will not replace the statistical programmer, but the programmer who uses AI will replace the one who doesn't" serves as a modern-day warning: adapt or be replaced by an adapted version of your own profession.

REFERENCES

1. Tarap, K., Morgan, D., Ghangare, P., 2025. SASBuddy: Enhancing SAS Programming with Large Language Model Integration. Available at https://www.lexjansen.com/phuse-us/2025/ml/PAP_ML06.pdf.

CONTACT INFORMATION

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

Author Name: Andy Ratcliffe
Company: Altair
Address: London, England
Email: aratcliffe@altair.com
Website: www.altair.com