

CT08 – Enhancing Stats Programming Efficiency in Pharma: Harnessing Large Language Models (LLMs) for Accelerated Development and Validation

A Framework for Compliant and Efficient LLM Integration in SAS Workflows

Andy Ratcliffe | November 2025 | PHUSE EU Connect, Hamburg



Agenda

01

The Context

Understanding why the pharma industry requires enhanced LLM efficiency for statistical programming

02

LLM Fundamentals

Exploring what LLMs are and their critical role within GxP environments

03

Technical Bridge

Integrating LLMs securely with SAS via RAG and API access protocols

04

Accelerated Development

Practical LLM applications for code generation and comprehensive documentation

05

Compliant Validation

Leveraging LLMs for automated QC and independent verification scripts

06

Governance Framework

Risk mitigation strategies, trust establishment, and maintaining auditability

07

Future Directions

Summary of key takeaways and emerging opportunities in pharma programming

The Challenge: Speed, Quality, and Volume

The Pharma Data Treadmill

Increasing Complexity

Clinical trials are becoming more intricate, demanding sophisticated statistical methods and advanced programming logic to handle multifaceted data structures and regulatory requirements

Regulatory Bottlenecks

The imperative for 100% accuracy, strict GxP compliance, and manual validation processes significantly constrains turnaround times whilst maintaining the highest quality standards

The Programming Burden

Statistical programming tasks - creating SDTM, ADaM datasets, and TFLs - remain exceptionally high-effort phases with limited automation opportunities under current methodologies

The Strategic Goal

Maintain unwavering data integrity and regulatory compliance whilst drastically reducing development and validation cycle times through intelligent automation



LLMs: Your New Programming Assistant

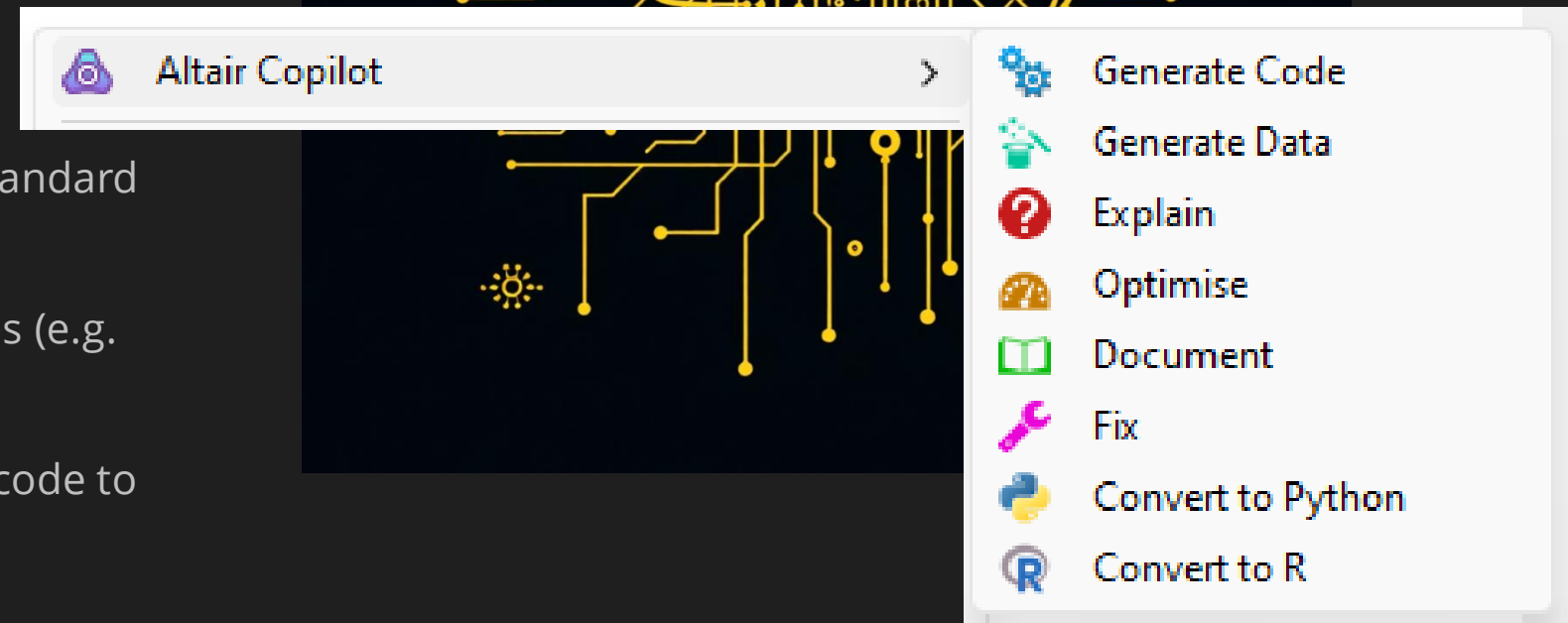
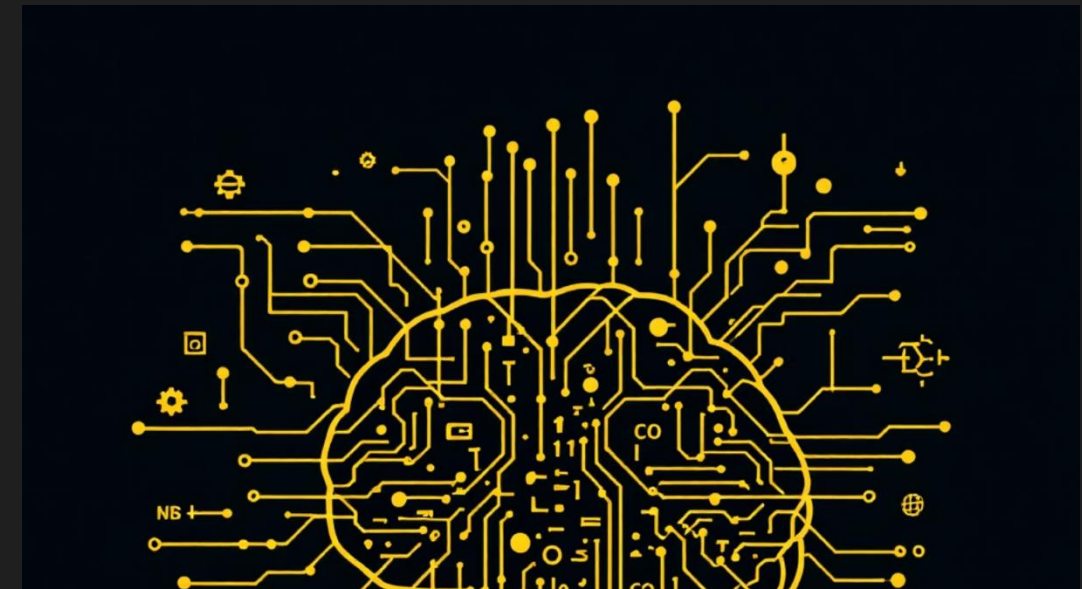
What is a Large Language Model?

Definition: Advanced algorithms such as ChatGPT, GPT-4, and Llama (built upon the Transformer architecture), representing the cutting edge of natural language processing technology.

Core Function: These models are trained on vast amounts of text to comprehensively understand, generate, and process both human language and programming code with remarkable fluency.

Key Capabilities in Programming:

- **Code Generation:** Automatically writing boilerplate code, standard functions, and procedure shells
- **Translation:** Converting code seamlessly between languages (e.g. SAS to R/Python)
- **Documentation:** Summarising or expanding upon existing code to enhance clarity and maintainability



Hi Ms: Your New Programming Assistant

Altair Copilot

Workspace1 - Untitled 1.sas - Altair Analytics Workbench

File Edit Navigate Search Project Run Altair Altair Copilot Window Help

*Untitled 1.sas X Rdemo1.sas SAS-R-Python... RUN_ALL_ALL.sas 02_DataPrep... adam_definex... »₁

Local Server

- Databases
- Libraries
- Filerefs
- Log
- Hub Server
- nlb.altair-slc-dev.com2 Server

Altair Copilot X

Copilot

Ask Altair Copilot a question

Altair Copilot uses AI. Results may not always be accurate.

Hub: Not logged in

Default Server: Local Server

Standard Licence

ENGLISH_UNITEDSTATES

No python interpreter

Progress Outputs X

When	Source	File	Results	Server	Status	Durati...
------	--------	------	---------	--------	--------	-----------

12:03
23/10/2025

- Generate Code
- Generate Data
- Explain
- Optimise
- Document
- Fix
- Convert to Python
- Convert to R

LLMs in a High-Stakes Environment

The GxP Distinction: Stochastic vs. Deterministic

Stochastic Nature

LLMs are fundamentally **probabilistic**. They predict the "most likely" correct answer through statistical patterns, which can occasionally lead to "**hallucinations**"—outputs that appear plausible but are factually incorrect or inconsistent

Deterministic Requirement

Regulated programming under GxP mandates **deterministic** results—outputs must be perfectly consistent, reproducible, and verifiable across all executions without variation

The Human-in-the-Loop (HIL) Model

1

LLMs as Tools

Productivity assistants that augment human capability

2

Programmers as SMEs

Statistical programmers retain sole ownership of validation and compliance

3

Mandatory Review

All LLM-generated code requires rigorous review, testing, and validation



The Technical Bridge: Connecting LLMs to Your Workflow

Integrating LLMs with SAS Infrastructure

External API Access

LLMs reside outside the SAS environment, requiring secure API communication channels to facilitate interaction whilst maintaining system integrity

- **SAS language:** Utilising PROC HTTP or PROC PYTHON to send structured prompts to secure LLM endpoints (e.g. AWS Bedrock, Azure OpenAI Service)
 - “Bring Your Own LLM”
 - Governance and Explainability
 - Low-Code / No-Code Tools
- Encrypted connections ensure data protection during transmission
- Authentication protocols prevent unauthorised access



❏ Data Security is Paramount

- **NEVER** send clinical patient data to the LLM under any circumstances
- LLMs only process **metadata** (variable lists, formats), **code snippets**, or **specifications**
- Implementation must use **private LLM instances** (e.g. within a private cloud/endpoint) to maintain environment security and regulatory compliance

Retrieval-Augmented Generation (RAG) for Pharma Standards

RAG: The Compliance Guardrail

The RAG Concept

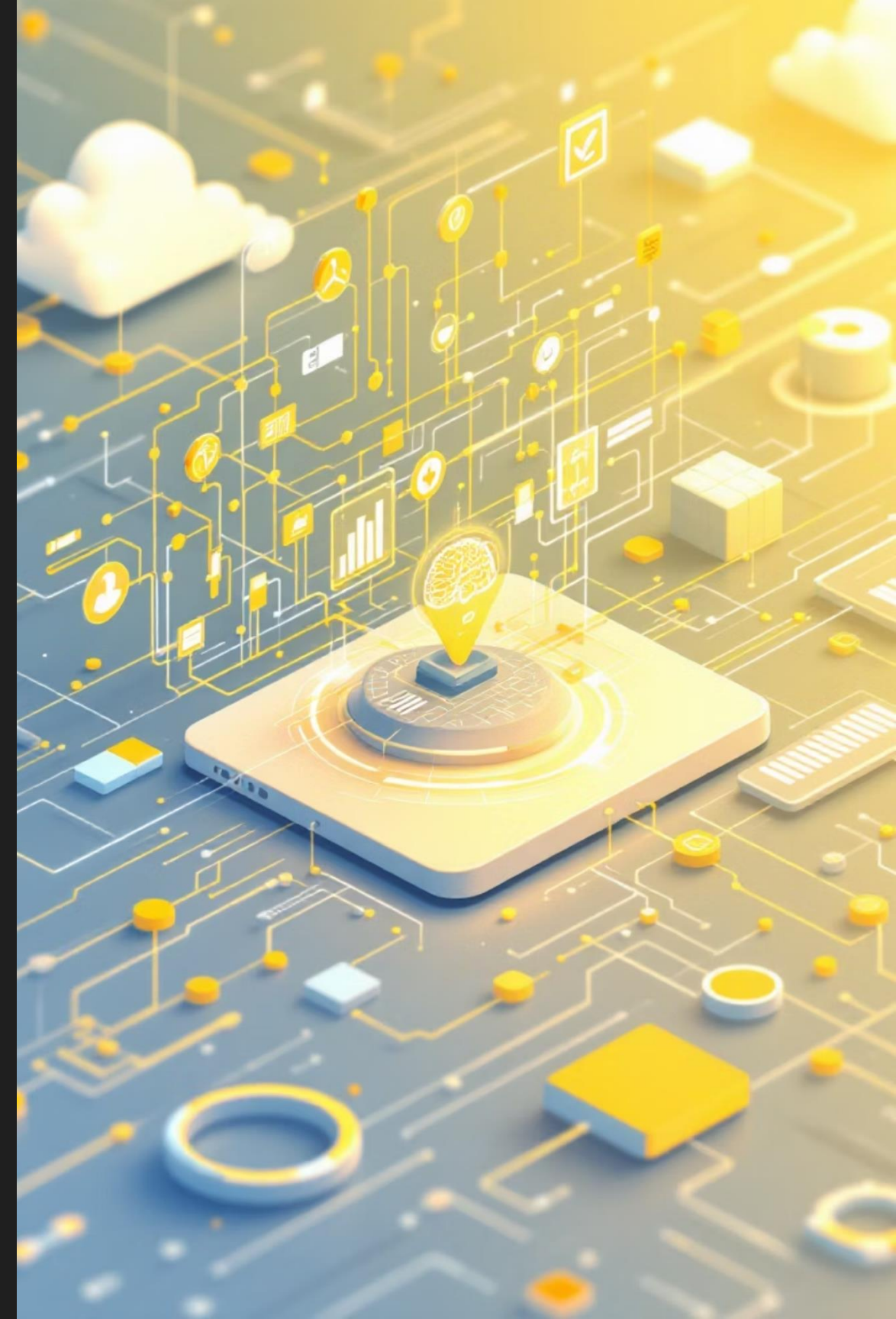
Whilst LLMs are exceptionally powerful, they remain generic without context. RAG grounds the LLM with specific, authoritative knowledge tailored to your organisational requirements.

This approach transforms a general-purpose AI into a **domain-specific programming expert** that understands your unique standards and compliance frameworks.

The Benefit: By referencing internal standards, the LLM generates code that is not merely syntactically correct, but also **institutionally compliant** and audit-ready from the outset.

Pharma RAG Knowledge Base

- Internal Standard Operating Procedures (SOPs) defining organisational best practices
- Internal Programming Guidelines and comprehensive Style Guides
- Validated SAS Macros and Utility Functions from your approved library
- CDISC Implementation Guides (SDTM/ADaM/Define-XML) ensuring industry standards



Use Case I: Accelerating Development & Reducing Boilerplate

LLMs in Accelerated Development (I) - Code Generation

40-60%

Time Reduction

Potential decrease in time spent on initial drafting of routine SAS modules

3x

Productivity Gain

Faster iteration cycles for standard programming tasks

Challenge Addressed: Writing Repetitive, Low-Variability Code

1

TFL Shells

Generating empty PROC REPORT or PROC TABULATE frameworks directly from Display Mock specifications, eliminating hours of manual template creation.

2

ADaM Logic

Creating DATA steps for standard derivations (e.g. VISIT Day/Week calculations based on protocol logic), ensuring consistency across studies.

3

Variable Formatting

Generating comprehensive PROC FORMAT code based on value-level specifications, reducing formatting errors and improving documentation quality.



Altair Copilot



- Generate Code
- Generate Data
- Explain
- Optimise
- Document
- Fix
- Convert to Python
- Convert to R



Use Case II: Debugging, Translation, and Documentation

LLMs in Accelerated Development (II) – Beyond Code Generation



Debugging & Error Resolution

Feed a SAS log error message *and* the relevant code snippet to the LLM. The model suggests targeted fixes, citing potential causes based on its extensive training data or RAG-fed SOPs, dramatically reducing troubleshooting time.



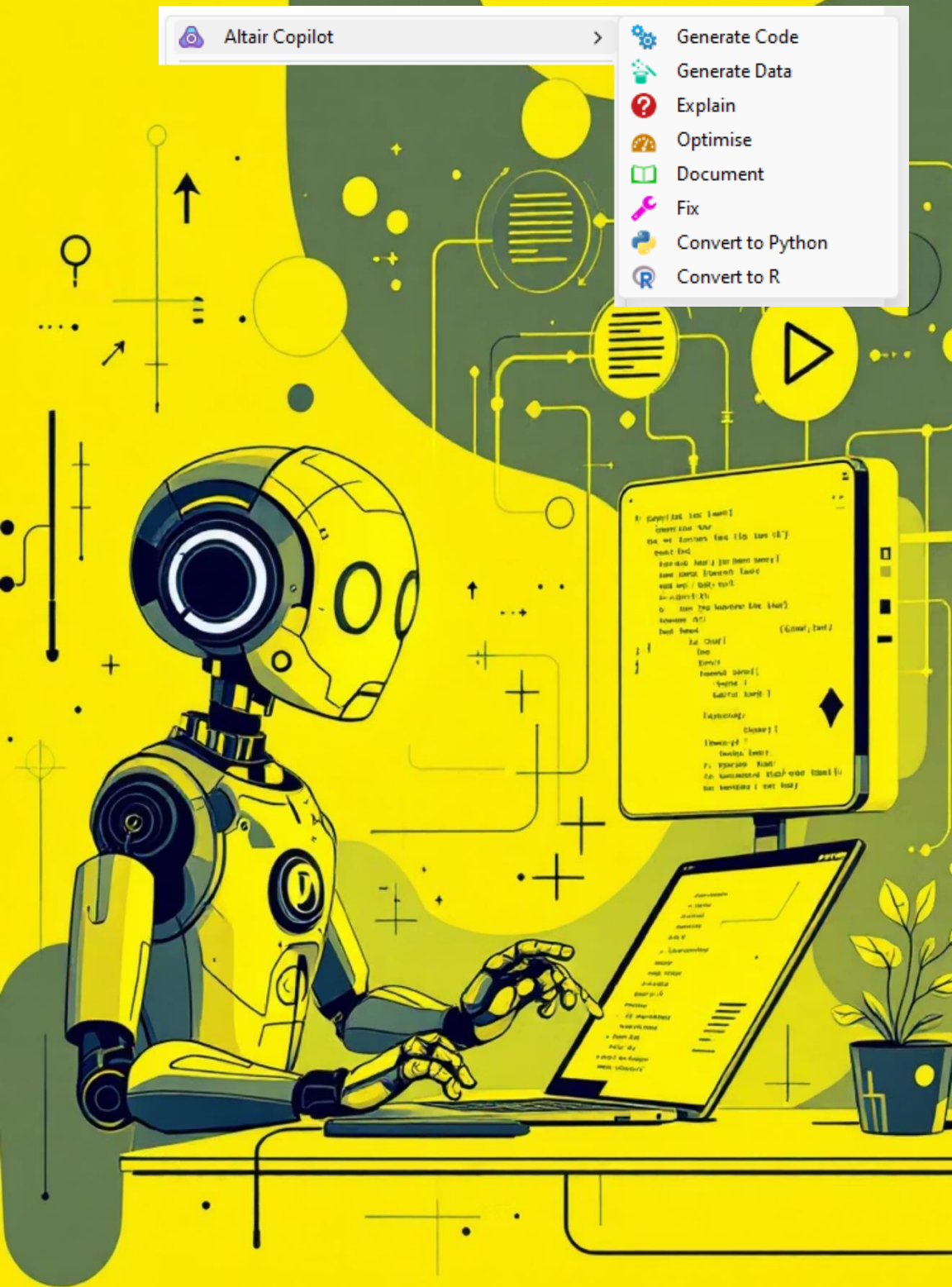
Code Translation & Migration

Quickly convert legacy SAS procedures to modern open-source equivalents (R/Python) for hybrid systems. Translate complex SQL logic into SAS Data Steps for consistent data flow across platforms.



Documentation Generation

Automatically draft GxP-compliant programming specification documents from high-level analysis plan text, ensuring consistency between analysis intent and code implementation whilst reducing documentation burden.



LLMs in Compliant Validation (I)

Use Case III: Validation & Quality Control (QC)

The Validation Goal: Validation shifts from purely checking execution to ensuring *compliance with standards*.

LLM-Assisted Automated QC

- The LLM acts as a "smart" reviewer by ingesting the human-written code
- It checks the code against the RAG-fed **Internal Style Guide and SOPs**
- **Output:** A checklist flagging inconsistencies (e.g. missing header comments, non-standard variable names, non-compliant macro usage)





LLMs in Compliant Validation (II)

Use Case IV: Creating Independent Verification Scripts

The Golden Rule of Validation: Verification code must be written *independently* from the development code

1

Prompt 1 (Development)

Programmer writes code with human-in-the-loop oversight, ensuring logic aligns with analysis specifications

2

Prompt 2 (Validation)

QA/Reviewer uses the *original analysis specification* to prompt the LLM to generate a **separate, single-purpose R or SAS script**

3

Verification Task

This script reproduces a key output (e.g. calculating N or Mean of an endpoint) using an alternative method or language

4

Audit Trail

Generate comparison documentation (Diff Reports) between the main programme's output and the LLM verification script's output



Risk Mitigation and Governance

Governing LLM Use: Trust, Traceability, and Training

1

Hallucination Management

Implement rigorous testing to proactively identify and block illogical outputs (e.g. testing with negative/illogical prompts).

2

Audit Trail & Traceability

All LLM interactions must be logged, including the exact prompt, the model used, and the raw output, to support regulatory audits.

3

Training & Prompt Literacy

Programmers and QA must be trained on **Prompt Engineering** - how to write clear, constrained, and effective prompts that minimise ambiguity and risk.

4

Continuous Monitoring

Establish metrics to continuously monitor the accuracy and compliance rate of the LLM in production environments.

Conclusion

Summary: The New Paradigm

Component	Traditional Workflow	LLM-Augmented Workflow
Development	Manual coding, low automation	Accelerated drafting, boilerplate reduction
Code Quality	Dependent on manual review	Automated QC against RAG-fed SOPs
Validation	100% human-driven dual programming	LLM assists in generating independent verification scripts
Speed	Linear scale with complexity	Exponentially reduced cycle times

Key Takeaway

LLMs are powerful accelerators, but success is contingent on secure integration, robust compliance guardrails (RAG), and a commitment to the Human-in-the-Loop model.

Future Directions & Q&A

The Next Steps

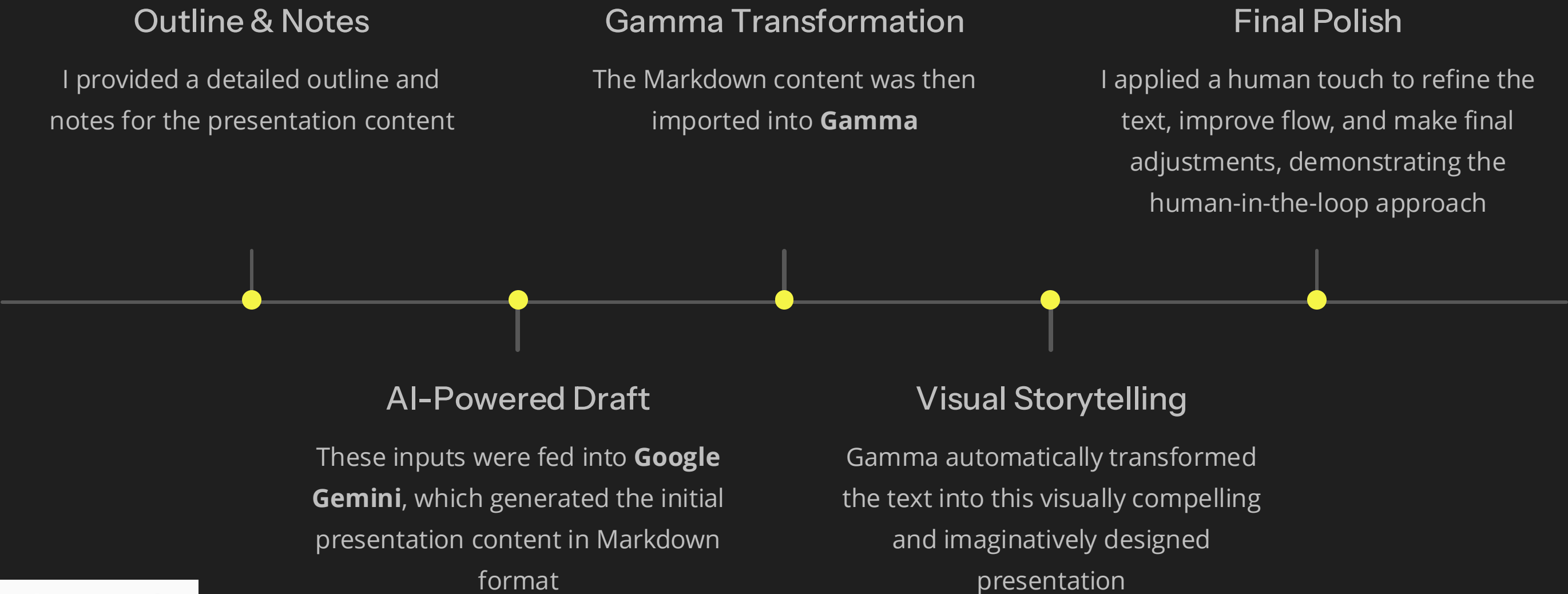
- 1 **Deeper Integration**
Moving LLM assistance directly into IDEs (Integrated Development Environments) for real-time coding suggestions
- 2 **CDISC Automation**
Full LLM-driven generation of Define-XML and annotated CRF (aCRF) from protocols
- 3 **Continuous Accuracy Monitoring**
Developing sophisticated systems to track and retrain domain-specific models



The Future: AI will not replace the statistical programmer, but the programmer who uses AI will replace the one who doesn't

Appendix: How This Presentation Was Created

This presentation itself is a demonstration of how AI can enhance efficiency and creativity in content generation...



Thank You



Andy Ratcliffe
Snr Solution Architect
Altair

Andrew.Ratcliffe@altair.com

<https://www.linkedin.com/in/andrewratcliffeuk/>

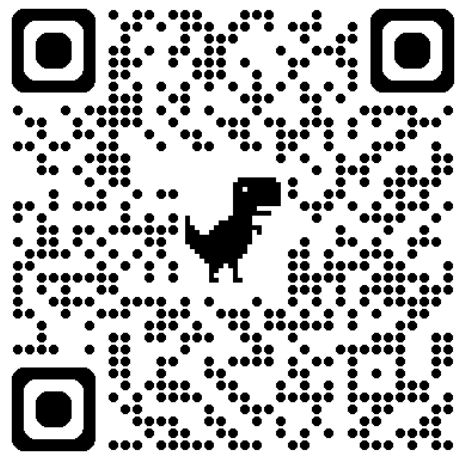


Andrew Ratcliffe

Passionate, delivery-focused
life sciences leader, across techn...



MeetUp



Independent SAS Language Community

<https://www.linkedin.com/company/suguki/>

Networking events for the SAS Language Community. We hold meetups for anyone interested in SAS - be that programming, administration, analytics, modelling or just for fun.

<https://www.meetup.com/SUGUKI/>

LinkedIn



