Paper ML 10

From Expert Coding to Expert Steering: Zero-Shot TLF Generation with Agentic AI

Florian Wiese, Bayer, Wuppertal, Germany
André Kelly Wibawa, Bayer, Berlin, Germany
Dr. Christoph Berns, Bayer, Leverkusen, Germany
Antonia Wiebeler, AWS, Berlin, Germany
Philipp Kaindl, AWS, Zurich, Switzerland

ABSTRACT

Producing Tables, Listings, and Figures (TLFs) remains one of the most important and expert-dependent steps in clinical study reporting. In a 10-week proof of concept, AI engineers from AWS and statistical programmers from Bayer jointly build a zero-shot, multi-agent workflow that generates structured work packages, executable R code, and draft TLFs with automated checks and human review. Developed on a non-oncology study and extended to oncology with minimal rework, the approach shows study-agnostic potential while outputs largely match ground truth. The proof of concept demonstrates TLF generation in hours rather than weeks. More importantly, it shows a shift in the role of statistical programmers: from primarily authoring code to steering agents, validating outputs, and capturing re4.6usable knowledge through human feedback and persistent skills.

Rather than presenting AI as a standalone coding tool, this collaboration shows how domain expertise and technical engineering can be combined to scale statistical programming knowledge, preserve expert control, and unlock broader opportunities across the clinical reporting workflow.

1. INTRODUCTION

Clinical Trials function as highly complex, yet efficient projects. While being very costly to run, including a study team setup of Trial Nurses, Investigators, Monitors, Study Managers, Data Managers, Biostatisticians, Statistical Analysts, Medical Writers, Regulators, and others, they provide no immediate return in profit and often are inherent in failing. Success rates of clinical trials have been declining since 2001-2009 where they were ranging between 38% to 70% to around 20 to 50% in 2015-2023, with overall success rates of clinical development programs below 10% [25].

Naturally, organizations are searching for ways to become more cost-effective. Simultaneously, there is an inherent incentive to advance time to market as the driving KPI to get to ROI faster. Clinical development time for innovative drug is being estimated at 9.1 years [26].

Tables, Listings, and Figures (TLFs) form the evidentiary backbone of every Clinical Study Report (CSR) submitted to regulatory authorities. A typical Phase III trial requires 200–500 TLFs spanning demographics, efficacy endpoints, safety summaries, and laboratory analyses. Producing these outputs requires statistical analysts to interpret Statistical Analysis Plans (SAPs), write SAS, Python or R code against ADaM datasets, validate outputs against specifications, and iterate through quality review: a cycle that typically consumes weeks per section and demands deep domain expertise in both clinical trial methodology and regulatory formatting conventions.

The scale of this effort is substantial. A pharmaceutical organization running 30 to 40 studies per year devotes a significant share of its statistical programming capacity to what is fundamentally a translation task: converting unstructured specifications into executable code that produces formatted outputs.

Recent advances in large language models (LLM) have demonstrated impressive code generation capability [1, 2, 3], and the PHUSE community has responded rapidly. The 2024–2026 period alone produced over 100 papers exploring AI-assisted statistical programming [4-12]. Multi-agent decomposition, iterative self-correction, and "AI augments, not replaces" framings are now community baselines. This is also reflected in the hype cycle for platform engineering, which sees AI Agents and AI Application on the peak of inflated expectations in May 2026, looming above the “Trough of Disillusionment”. Yet, most published systems address isolated steps (single-table generation, metadata extraction, or code assistance) without tackling the full pipeline from unstructured specification to validated output. This also reflects Spec-Driven Development and Autonomous Workload optimizations are still at the beginning of the Hype Cycle.

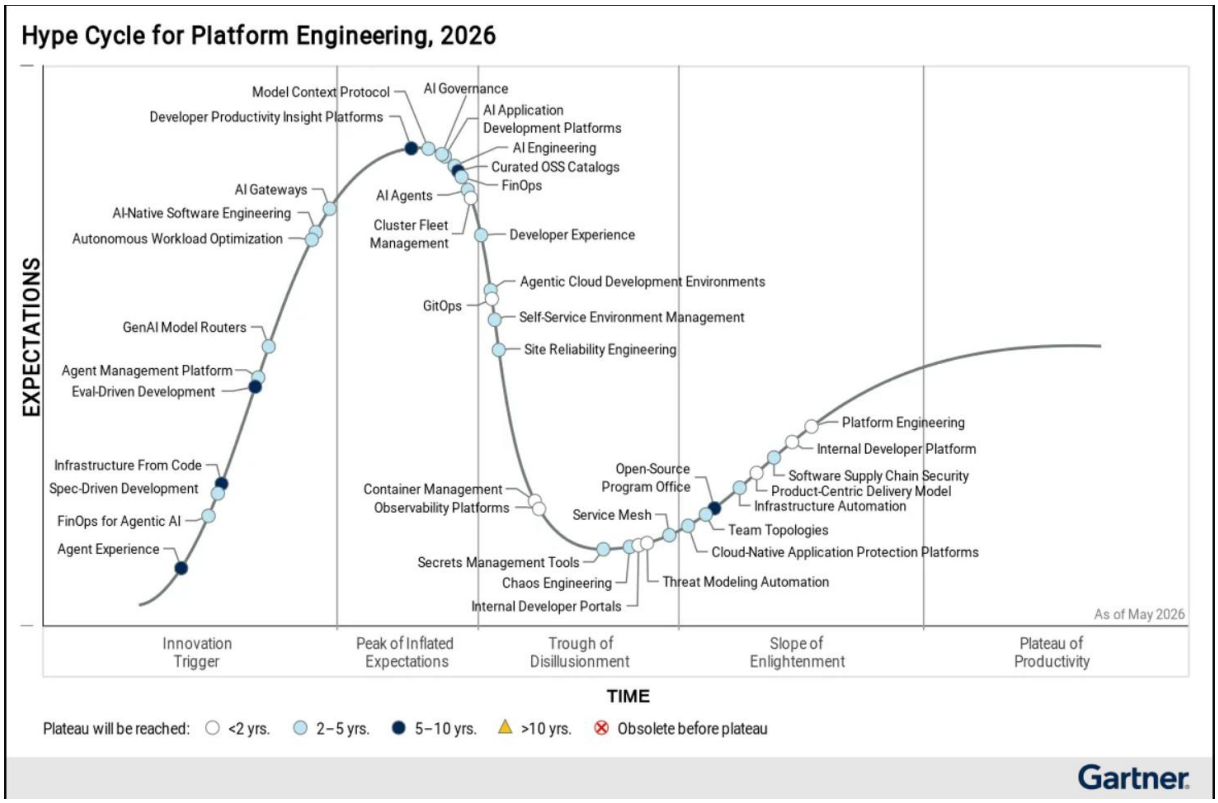


Figure 1: Hype Cycle for Platform Engineering, 2026 [17]

The pharmaceutical industry is simultaneously adopting open-source languages for statistical programming, creating an opportunity to design AI-assisted workflows from the outset rather than retrofitting them onto legacy codebases.

In this paper, we describe a 10-week proof of concept between Bayer and AWS that demonstrates a different model. Statistical analysts reaching reliable and structured outputs steering AI agents rather than writing code from scratch or using prompting to iterate through versions of code always on the hunt for bugs. We present the system architecture, the design choices that distinguish it from prior work, results across two therapeutic areas, and most importantly, what changes for the analysts who use it.

2. THE WORKING MODEL

The proof of concept was structured as a Bayer-AWS Generative AI Innovation Center (GenAIIC) engagement: a 10-week, time-boxed collaboration where AWS AI scientists work alongside the customer's domain experts to build and validate a production-oriented prototype.

AWS scientists owned system architecture, agent design, prompt engineering and infrastructure, and iterated the multi-agent pipeline itself. Bayer's statistical analysts brought domain expertise and ground truth, evaluating work packages, code output, and rendering TLF against production standards. The two groups stayed in the same room rather than working across a vendor's handoff and followed a weekly sprint cadence: AWS shipped incremental capabilities each week. Bayer analysts reviewed the outputs, surfaced failure modes, and fed feedback back into the design within the same cycle. This loop was not a formality - it directly produced two of the system's defining features, the corrected work-package contract and the persistent-skills mechanism, neither of which had been planned upfront.

Neither group could have built this alone. AI engineers without continuous domain input risk optimizing for code that runs rather than code that produces factual artifacts; domain experts without engineering capacity know exactly what "correct" looks like but rarely have the capacity to build agent architectures and safety gates around that knowledge.

Two studies tested how well this working model generalized across therapeutic areas. Study A, an ophthalmology trial with standard efficacy and safety TLF requirements, was the primary development target: the system was designed, trained via skills, and validated here over the engagement's first seven weeks. Study B, an oncology trial with materially different endpoints, populations, and table structures, served as the transfer target in weeks 8-10. The key question was not whether the system could produce *one* correct table, but whether the same pipeline and accumulated knowledge could transfer across therapeutic areas with minimal human rework.

The proof of concept operated within deliberate boundaries. Code generation targeted R only, using pharmaverse packages including rtables, dplyr, ggplot2, and r2rtf. TLF scope focused on Sections 8.1 (Demographics) and 8.3 (Safety), the highest-volume table families in a typical CSR, with layout perfection, listings, and figures formally out of scope. The pipeline

nonetheless showed it could extend beyond tables without structural change: a Hy's Law scatter plot was generated for the oncology study and closely matched the SAS ground truth. Formatting gaps remained in presentation details such as page breaks, spanning headers, and footnote pagination, which required manual post-processing. These constraints were intentional. The goal was to prove the core pipeline - specification interpretation, code generation, and cross-study transfer - before extending to the full breadth of a CSR deliverable package.

3. SYSTEM OVERVIEW

3.1 END-TO-END WORKFLOW

The system runs as a two-agent pipeline connected by a structured contract. Rather than one agent attempting interpretation, code generation, and validation in a single pass, the pipeline splits at the point where a human needs to intervene: after interpretation, before code generation.

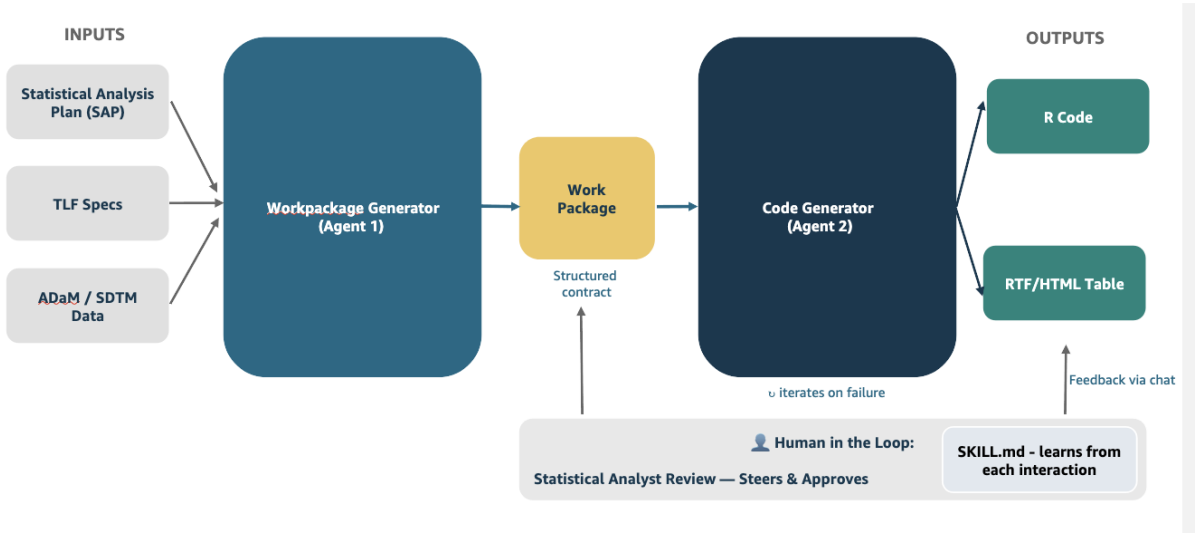


Figure 2. Data2TLF end-to-end workflow. Two specialized agents are connected by a structured work package that serves as both machine-readable input and human-reviewable contract. Human oversight gates ensure expert control at each stage, Skills are learned from the human feedback and persist across runs

3.2 THE WORK-PACKAGE GENERATOR (AGENT 1)

The Work-Package Generator reads unstructured study documents - SAP PDFs, TLF specification templates as images, ADS specifications, and ADaM dataset headers - and produces a structured work package specifying exactly what the second agent must build: the analysis requirements (population, endpoints, statistics, groupings), the dataset grounding (which ADaM datasets and variables are needed, validated against live .sas7bdat headers rather than assumed from SAP text alone), the display specification (column structure, row groupings, decimal precision, footnotes), and any clarifications or assumptions the agent had to resolve along the way.

The work package is human-reviewable before code generation begins - the point of the architecture. A programmer catches a misread SAP clause, a wrong population assumption, or a missing subgroup before any R code exists, rather than during code review or output Quality Control.

3.3 THE CODE GENERATOR (AGENT 2)

The Code Generator takes the approved work package plus template images and produces executable R code. It generates code informed by the TLF specification template images, so the model sees the target layout directly rather than inferring it from text; runs pre-execution safety gates that screen for clinical-programming errors before anything runs; executes in an isolated sandbox; evaluates the rendered output via an LLM-as-judge that compares it against the template image; and iterates automatically if the judge finds the structure doesn't conform.

3.4 HUMAN OVERSIGHT GATES

Four checkpoints keep a programmer in control, each catching a different class of error at the point where correction is cheapest: **discovery**, where the programmer confirms the system found the right target TLF and source documents; **work-package approval**, where the programmer signs off on the specification before any code is written; **code review**, where the programmer inspects the generated R code; and **output sign-off**, where the programmer validates the final rendered TLF.

At any gate, a programmer's feedback does double duty - it steers the next iteration, and where it reflects a general rule rather than a one-off fix, it is captured in the persistent skill repository, so a correction made once can prevent the same error from recurring on the next TLF, the next study, or the next programmer's first attempt.

4. KEY DESIGN CHOICES

The PHUSE community has reached consensus on the fundamentals of AI-assisted TLF generation: multi-agent architecture, iterative self-correction, human-in-the-loop review, and time savings measured in hours rather than weeks are now community baselines. Our design choices sit at the contract, safety, and memory layers around these baselines, and each one emerged directly from collaboration with Bayer's statistical programmers rather than from architectural preference.

4.1 THE WORK PACKAGE AS AUDITABLE CONTRACT

Prior systems typically decompose SAPs directly into code with no intermediate reviewable artifact [6] or require pre-existing structured metadata such as CDISC ARS [7] or Pinnacle 21 [5] exports as a precondition. We generate a structured markdown contract directly from raw SAP PDFs and TLF specifications instead, with no precondition that the study team has adopted a particular metadata standard. That single document serves three audiences at once: machine-readable input for the Code Generator, a human-reviewable specification a programmer approves before any code is written, and a regulatory audit trail of what was interpreted and how ambiguities were resolved.

4.2 SCHEMA GROUNDING WITHOUT DATA EXPOSURE

Some published systems inject preview values of ADaM columns directly into prompts, exposing patient-level data to external LLMs. We validate dataset and variable existence against .sas7bdat headers only - names, types, and labels - so no patient-level data ever reaches the LLM.

4.3 TEMPLATE-AWARE MULTIMODAL GENERATION

Table specifications carry layout information - column structure, row groupings, indentation, header hierarchy - that is largely lost when reduced to text. A prompt can say: "nested by AE preferred term under system organ class" and still leave the model guessing at indentation depth or where a total row belongs. We inject the TLF specification image directly into the code-generation prompt, so the model sees the target layout before writing a line of code, rather than only discovering a mismatch during judging.

4.4 PRE-EXECUTION CLINICAL SAFETY GATES

Most comparable systems rely on runtime-error retry loops, discovering a problem only once code executes and either crashes or, more dangerously, runs cleanly and produces a plausible but wrong result. We run static analysis before execution instead, checking for clinical-programming-specific error classes - missing required packages, work-package/schema contract violations, and references to datasets or variables that don't exist - and block code with error-severity findings from running at all. This matters because the most dangerous errors here don't crash anything: a wrong dataset feeding a denominator or a missing occurrence flag both execute cleanly and look correct.

4.5 TEMPLATE-IMAGE LLM-AS-JUDGE

Confirming that a generated table structurally matches its specification - headers, columns, row groupings, footnotes - has traditionally meant a human comparing the two line by line. We use a multimodal LLM judge instead, comparing the rendered output and the specification template as images and flagging structural non-conformance to trigger another generation pass automatically. It is a different problem from cross-table consistency checking: our judge evaluates a single output against its own template and makes no claim about whether the numbers themselves are right.

4.6 PERSISTENT SKILLS FROM HUMAN FEEDBACK

Every correction a programmer makes is institutional knowledge that, in most systems, evaporates once the session ends. We capture it instead: a correction such as "adverse event tables at Bayer always use the occurrence flag from X, not Y" becomes a reusable, human-reviewable skill, stored in a persistent repository and automatically retrieved for relevant future work packages. Skills compound - the system processing of Study B benefited from everything learned on Study A with no architectural change, and a new team member inherits the accumulated conventions rather than rediscovering them from scratch.

5. RESULTS

The system was developed and iterated on Study A (ophthalmology) over the first 7 weeks of the engagement. In weeks 8–10, it was applied to Study B (oncology) with no architectural changes. Only the study-specific inputs (SAP, specifications, datasets) were exchanged.

Outputs were evaluated by Bayer statistical analysts against production-quality ground truth. Assessment was qualitative (correct/partially correct/incorrect) across dimensions:

- **Statistical correctness:** Denominators, counting logic, summary statistics
- **Structural conformance:** Column layout, row groupings, nesting hierarchy
- **Formatting:** Decimal precision, footnotes, titles, page breaks
- **Code fidelity:** Correctness and comprehensiveness of generated R programs

The work-package contract generalized across therapeutic areas without structural modification, Skills accumulated on Study A (e.g., formatting conventions or denominator logic) transferred directly to Study B.

Where Study B introduced novel requirements (oncology-specific analyses, domain-specific table layouts), the system produced reasonable first drafts that required programmer refinement rather than complete rewrites.

Outputs from Study B largely matched ground truth with minimal additional skill curation. Given sufficient documentation, the agents provided numbers and table structures that aligned with the ground truth. Edge cases were often inferred correctly by the system.

Issues in generated TLFs were traceable to implicit knowledge, not documented in the TLF specifications or the SAP. An example is different available variables covering similar purposes with slightly different derivation logic.

Structural conformance contained the most gaps. Replicating the exact indentations and breaks was often not done on first iterations.

The quality of first-pass code was sound and given its proper structure and commenting, adaptable by steering from the domain expert. The quality of the generated code was directly tied to provided context and tools. Standardized R packages available for the system improved the overall output quality. Compared to ground truth SAS programs, code was more verbose. Due to internal formatting standards as well as additional checks integrated by the AI, heavily standardized table code was significantly longer than the internally produced (e.g. Code for a demographics table had 618 rows in R vs 189 rows in SAS). The difference in less standardized tables was reduced (e.g. Adverse events table 495 vs 283).

Overall system demonstrated credible automated TLF generation for standard table types. Tables were created in a matter of minutes, complete sections in hours. More complex tables with multiple denominators, or complex layouts needed guidance by the analysts. Ambiguity in specifications translated into higher probability of incorrect results.

6. FROM CODING TO STEERING: THE CHANGING ROLE OF THE STATISTICAL ANALYST

6.1 THE TRADITIONAL WORKFLOW

In conventional clinical programming, the statistical analyst's day-to-day work includes:

- Reading and interpreting SAP, Protocol, TLF specifications and other study documentation
- Setting up Analysis Dataset specifications and transformations
- Developing R, Python or SAS code to implement analysis logic
- Debugging code against dataset structures
- Formatting outputs to match specifications
- Iterating through QC and review cycles
- Documenting derivation logic
- Interaction with different stakeholders (i.e. the study team)
- Project planning by matching timelines for deliverables to the study flow

The analyst value comes from both their domain expertise (knowing what a table should contain) and their technical skill (implementing it in code). The role depends on the capability to integrate statistical, clinical, technical, and operational knowledge. In the traditional workflow, these capabilities are tightly coupled with the same person.

Domain expertise covers statistical knowledge as well as the medical background and expertise on standards for clinical data. The focus of the statistical analyst lies in understanding the data structure and translating the teams' questions into data queries. As such, they act as a connection between medical science, data analytics, data engineering and traditional statistics.

Technical skills typically apply to translating this expertise through coding. A constraint for analysts in most pharmaceutical companies up to today remains the dependency on SAS, a proprietary system that is not often taught in universities.

The combination of this special domain and technical expertise is a rare skill set. As such, analysts often come into the industry with a variety of backgrounds, including mathematics, statistics, biology, pharmaceuticals or broader education with an emphasis on data analysis, such as economics.

6.2 THE EMERGING WORKFLOW

With an agentic AI workflow, such as the one designed during the PoC, the analyst's role shifts from manually translating specifications into code toward supervising a structured, agent-assisted production process including work packages, code, and draft outputs.

Activity	**Traditional**	**With Agentic AI**
Specification interpretation	Manual reading + mental model	Agent-generated work package for review
Code authoring	Primary activity (hours/days)	Agent-generated, analyst reviews
Debugging	Analyst identifies and fixes issues	Pre-execution gates catch errors; agents iterate; analyst steers correction via feedback
Formatting	Manual coding adjustments to match specification	Multimodal template understanding; judge checks structural conformance
QC	Second analyst re-implements and/or code review	LLM-as-judge supports structural checks; analyst performs final QC and sign-off
Knowledge capture	Informal (tribal knowledge)	Explicit skill artifacts in persistent repository

Routine activities such as generating first-pass code, resolving simple execution issues ("bug-hunting"), and aligning outputs with visual templates can increasingly be handled by the system. But, as shown in the PoC, the system needs the correct context. The analyst's focus therefore shifts to other areas of their work. Reviewing whether the work package correctly reflects the SAP, protocol, dataset structure, and TLF specification before code is generated. Validating whether the generated code and outputs implement the intended analysis correctly and are suitable for regulatory use. This makes domain understanding, critical review, and the ability to articulate feedback more important than manual coding speed alone.

6.3 THE “AI-ENABLED ANALYST”

Instead of working and making assumptions and decisions independently, the AI-enabled analyst interacts with new partners in the form of Agents. The proof of concept showed that statistical analysts need a refined skill profile to get the most out of this cooperation. While some skills remain unchanged, some emerge, some become more important, and some become less central.

6.3.1 AGENT STEERING

Interacting with the agentic system requires the analyst to be able to evaluate intermediate AI-generated artifacts, understand how they were created and guide the system through clear, actionable feedback while avoiding to fall victim to prominent biases such as blindly trusting the output (“Automation Bias”) [23].

Analysts must be able to identify and translate issues in a way that the system can act on. They must learn the “language of the system”. The content of the feedback has direct implications on the quality of the returned result. With general statements, AI might come up with different, even wrong assumptions or hallucinations. Effective steering therefore requires analysts to translate domain judgment into explicit instructions. This turns feedback into an operational input rather than an informal review comment.

6.3.2 KNOWLEDGE CURATION

Structuring corporate knowledge in processes is a key component in every company. For analysts using an agentic system, this means transforming individual corrections into reusable skill artifacts. Doing so enhances efficiency of the system as it allows direct application of learned conventions in future iterations. When handled correctly, expert feedback turns into structured memory.

Analysts need to implement a workflow, where they not only reflect changes on a single output, but routinely include assessment on whether the correction represents a general rule, a study-specific exception, or an organizational standard during development. Shaping a reusable knowledge base that governs TLF generation becomes a responsibility of every analyst.

6.3.3 DOMAIN EXPERTISE

The need for Statistical analysts remains to understand what a table, listing, or figure is intended to communicate and whether the output reflects the study design, endpoint definition, analysis population, visit structure, subgroup logic, and statistical methodology. This knowledge cannot be reduced to code syntax alone. Even if an agent produces executable code, the analyst must judge whether the analysis itself is appropriate. For example, the analyst still needs to recognize whether the statistical method applied is correctly implemented, whether treatment-emergent events were counted according to the SAP or whether the correct clinical endpoints were used in the analysis.

6.3.4 EDGE-CASE RESOLUTION

Studies frequently contain ambiguities, protocol-specific decisions, non-standard derivations, special populations, data anomalies, or late changes that cannot be solved by generic automation alone. These cases require judgment, discussion with statisticians or clinical colleagues, and sometimes a deliberate decision between multiple acceptable interpretations. Agentic AI may help surface the ambiguity or propose an implementation, but resolving the ambiguity still depends on human reasoning and cross-functional alignment.

6.3.5 CODE REVIEW

The analyst must be able to critically assess whether an AI-generated work package fully and correctly reflects the design of the study through SAP and protocol. This builds on traditional validation competencies through independent code review. However, it differs from independently developing and reviewing code. When creating a program from scratch, missing assumptions or ambiguities often surface naturally during implementation. When reviewing an AI-generated work package, the analyst must deliberately look for what may be missing, incorrectly inferred, or insufficiently precise before code generation begins. The quality of this review directly determines the quality of the downstream code and output.

6.3.6 OUTPUT VALIDATION

Even if structural checks, execution gates, and LLM-as-judge components reduce manual effort, the analyst remains the primary quality gate for statistical correctness and regulatory suitability. Analysts must be able to compare rendered outputs against the specification, identify whether denominators, counts, percentages, confidence intervals, subgroup structures, and footnotes are implemented correctly, and distinguish cosmetic deviations from substantive analytical errors. This is particularly important because plausible-looking AI-generated outputs may still contain subtle mistakes that require expert review to detect.

6.4 IMPLICATIONS

Introducing Agentic AI does not replace the integrated expertise of the analyst but suggests changes in where and how it is applied in the emergent workflow. The shift mirrors transitions seen in other engineering disciplines when automation was introduced. The introduction of computer-aided-design (CAD) did not eliminate architects. It freed them from drafting to focus on design. The introduction of automated testing did not eliminate software engineers. It freed them to focus on system design and edge cases.

Similarly, the future process suggested here is not a purely autonomous model, but a hybrid one. Analysts steer agentic systems, maintain sufficient code literacy to assess their outputs, and use modern coding assistants or integrated development environments to preserve the connection between statistical intent, data structure, and executable implementation. While programming skills might no longer be as relevant as they were in the past, the ability to understand, challenge and validate the implementation of code remains crucial, especially because the generated code remains part of a regulated deliverable. Relying on Agents to solve the problem can lead to a lack of expertise in the coding area as already experienced by many ([20], [21], [22]). Analysts and organizations must find a way to keep this critical skill set.

AI-assisted workflows must remain traceable. Organizations need to document how AI systems operate, how intermediate artifacts were created, what assumptions were made, and how final outputs were reached, in line with regulatory expectations such as FDA's draft guidance on the responsible use of AI [24]. Organizations need to shape their workflows and train their personnel on emerging workflows to stay compliant. Failing to do so can lead to rejection by the authorities and significant impact for the companies, as underscored by FDA's 2026 warning letter to Purolea Cosmetics Lab, which included a dedicated section on "Inappropriate Use of Artificial Intelligence in Pharmaceutical Manufacturing" [24]. The agency did not object to AI-assisted document creation as such. Rather, it criticized failure to review AI-generated specifications, procedures, and production records for accuracy and CGMP compliance before use. In that context, the FDA stated "If you use AI as an aid in document creation, you must review the AI generated documents to ensure they are accurate and compliant with CGMP. Your failure to do so is a violation of 21 CFR 211.22(c)." [24]. For AI-assisted clinical reporting, this reinforces why the work package must serve not only as machine-readable input, but also as a human-reviewable and auditable control point before downstream code generation and output production. [19] "Human-in-the-Loop" becomes the critical factor to enable AI assisted workflows.

The implication therefore is not "fewer programmers" but "analysts, focused on higher-value work": Quality oversight, knowledge management, specification quality, and regulatory strategy. Organizations that invest in this transition will be better positioned as regulatory agencies increase expectations for speed, consistency, and quality of submissions. Faster translation of clinical questions into analytical answers enables faster decision making through accelerated delivery of data insights. Softening the unique coding expertise means having a broader resource pool available.

7 LIMITATIONS AND FUTURE WORK

The proof of concept was tested on a limited number of studies and selected table families. The testing readout showed that the workflow can run end-to-end, and that chat-based correction is promising, but also that robustness varies by section, table complexity, and the quality of available context. Further testing, expanding this view by involving study experts across oncology, cardiovascular and radiology, further strengthening broad generalizability across all therapeutic areas, study designs, endpoints, and reporting conventions came to the same conclusions. In evaluating the system further, the importance of context became clear. If not sufficiently informed, sections were parsed incorrectly; duplicate or additional outputs were generated, and not all tables were discovered on the first draft. A controlled head-to-head study was not part of the POC scope.

The current results should be interpreted as evidence of feasibility and transfer potential, not yet as proof of production readiness. Moving from a successful proof of concept to a scalable trial-data platform requires more than improving model prompts. It requires industrialization of the workflow, systematic validation across studies, integration into the existing statistical computing environment, and a governed operating model for human-in-the-loop use.

The question on whether to produce R, Python or SAS code or even something completely different is a central one in clinical trial analysis across many years. As authorities are more accepting of formats different than SAS and cross-industry shared open-source codebases (e.g. pharmaverse) are available, the advantages of open-source languages (cost, availability, dynamic) start to outweigh missing company skills and standards.

A caveat remains for the assessment of the efficiency of agentic vs human development workflows. This needs further, quantified evaluation as analysts rarely code a table completely from scratch. Typically, there are templates, macros and code snippets available, developed over decades, that support the process. Increased review load mitigates time won over programming tasks. At the same time, costs of AI coding could increase significantly in the coming years. According to a Gartner forecast, they are even expected to exceed the average salary of a software developer by 2028. [18]

8. CONCLUSION

This paper presents a 10-week Bayer–AWS proof of concept demonstrating that agentic AI can generate TLFs close to production level quality in hours rather than weeks. The system's architecture (structured work-package contracts, multimodal template understanding, pre-execution safety gates, LLM-as-judge validation, and persistent skills) provides the governance layers needed to make AI-generated outputs trustworthy in a regulated context.

But the most important finding is about people, not technology. The proof of concept demonstrates a tangible shift in how statistical analysts work: from primarily authoring code to steering agents, validating outputs, and capturing reusable knowledge. This shift preserves the expert's domain knowledge and regulatory judgment while automating the routine translation work that currently dominates their time.

Although the workflow changes, the core accountability of the statistical analyst remains unchanged. Agentic AI can assist with interpretation, code generation, execution, formatting, and first-level checks, but it does not replace the analyst’s responsibility to understand the clinical question, the statistical intent, the data structure, and the regulatory context of the output. The system may accelerate the path from specification to draft TLF, but the decision whether a result is correct, meaningful, and fit for purpose remains with the human. Therefore, the shift in the working model is not from expert work to automated work, i.e. a replacement of the statistical analyst, but from manual execution toward expert oversight.

The collaboration model (AI scientists and statistical analysts co-designing a system through iterative feedback) proved essential. The system's most distinctive features (the work package, the persistent skills) emerged from analysts’ feedback, not from architectural invention. This suggests that successful AI adoption in clinical programming requires not just better technology but genuine partnership between AI practitioners and domain experts.

Similarly, the analyst’s role as a connector between disciplines strengthens. Statistical analysts continue to translate between clinical intent, statistical methodology, data availability, programming implementation, and reporting requirements. In many cases, this interdisciplinary role becomes even more visible. The analyst must ensure that an AI-generated artifact correctly represents the expectations of statisticians, clinicians, data managers, regulatory colleagues, and reporting teams.

As the pharmaceutical industry faces growing pressure to accelerate submissions while maintaining quality, agentic AI offers a path forward, one that amplifies expert capability rather than replacing it.

REFERENCES

This paper has been reviewed and created with the help of Anthropic (2026) Claude Opus 5.5 [Large Language Model] https://www.anthropic.com/

[1] Chen M., et al. "Evaluating Large Language Models Trained on Code." *arXiv:2107.03374*, 2021.

[2] Rozière B., et al. "Code Llama: Open Foundation Models for Code." *arXiv:2308.12950*, 2023.

[3] DeepSeek AI. "DeepSeek-Coder: When the Large Language Model Meets Programming." *arXiv:2401.14196*, 2024.

[4] Xu H., Rowsell T. "Real-Time AI-Driven TLF Generation with GAutoTLF." PHUSE US Connect 2026, ML18. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_ML18.pdf

[5] Ehmann A., Offenbecker K. "Utilizing AI for Automated Conversion of ADaM Specifications to R Code." PHUSE US Connect 2026, ML12. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_ML12.pdf

[6] GSK. "Streamlining Python and R Code Generation from SAP Using Multi-Agent LLMs." PHUSE APAC Connect 2026, CT02. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/APAC/Hyderabad/PAP_CT02.pdf

[7] Busa B., et al. "AI Code Generator: Automating TFL Programming with ARS Metadata." PHUSE US Connect 2026, SD04. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_SD04.pdf

[8] Vaidya G., Lysenko V. "SpectoSAS: AI-Powered Instant SAS Code Generation from Study Specs." PHUSE US Connect 2026, ML19. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_ML19.pdf

[9] Genentech. "Using Large Language Models for Consistency Checking Tables, Listings, and Graphs." PHUSE US Connect 2025, SM14. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PAP_SM14.pdf

[10] Novo Nordisk. "Automated Mock TFL Using an RAG LLM Setup." PHUSE EU Connect 2025, ML01. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/EU/Hamburg/PAP_ML01.pdf

[11] Zhang Q. "AI-Driven Automation of Statistical Analysis Plans." PHUSE US Connect 2026, ML01. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_ML01.pdf

[12] Li C., Li D. "AutoProgramming: Agents Are All You Need." PHUSE US Connect 2026, ML03. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_ML03.pdf

[13] Qubeck B., Chhugani R. "Metadata-Driven TLF Generation: End-to-End Automation from Protocol to Output." PHUSE US Connect 2026, DS18. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_DS18.pdf

[14] Patel U., et al. "Using ChatGPT for AI-Assisted Quality Control of Clinical Trial Tables and Figures." PHUSE US Connect 2026, ML24. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_ML24.pdf

[15] Papasani M., Pyle E. "Enhancing CSR Authoring with Generative AI: TLF Formatting and Metadata in Merck's iRAP Tool." PHUSE US Connect 2026, ML27. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_ML27.pdf

[16] Zloty E., et al. "From Tables to Text: Revolutionizing CSR Development with Generative AI." PHUSE US Connect 2026, ML30. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2026/Connect/US/Austin/PAP_ML30.pdf

[17] Gartner, Hype cycle for platform engineering 2026 https://www.truefoundry.com/gartner-2026-hype-cycle-for-platform-engineering

[18] Gartner STAMFORD, Conn., June 24, 2026 “Gartner Predicts AI Coding Costs Will Surpass Average Developer’s Salary by 2028 as Token Consumption Surges https://www.gartner.com/en/newsroom/press-releases/2026-06-24-gartner-predicts-ai-coding-costs-will-surpass-average-developer-salary-by-2028-as-token-consumption-surges

[19] FDA „WARNING LETTER - Purolea Cosmetics Lab” April 02, 2026 https://www.fda.gov/inspections-compliance-enforcement-and-criminal-investigations/warning-letters/purolea-cosmetics-lab-722591-04022026

[20] Sang, T. Đức. (2025, December 26). When ai makes you forget how to code. DEV Community. https://dev.to/mahou_anisphia/when-ai-makes-you-forget-how-to-code-5cii

[21] How ai made me forget how to code — and what I’m doing about it | by Ankit Singh | Medium. (n.d.-e). https://medium.com/@ankitsingh1583/how-ai-made-me-forget-how-to-code-and-what-im-doing-about-it360aa3a07e06

[22] Ai killed my coding brain but I'm rebuilding it | by | run.it.bare | medium. (n.d.-d). https://medium.com/devlink-tips/aikilled-my-coding-brain-but-im-rebuilding-it-8de7e1618bca

[23] Schiel, Zischka, Erdman Human-in-the-Loop in AI-driven Processes https://www.ecv.de/beitrag/pharmind/Human-in-the-Loop_in_AI-driven_Processes

[24] FDA Considerations for the Use of Artificial Intelligence to Support Regulatory Decision-Making for Drug and Biological Products https://www.fda.gov/regulatory-information/search-fda-guidance-documents/considerations-use-artificial-intelligence-support-regulatory-decision-making-drug-and-biological

[25] Zhou Ying e.a., "Dynamic clinical trial success rates for drugs in the 21st century". https://www.nature.com/articles/s41467-025-64552-2.pdf

[26] Brown, Dean G e.a. Clinical development times for innovative drugs, https://pmc.ncbi.nlm.nih.gov/articles/PMC9869766/

CONTACT INFORMATION

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

Author Name: Antonia Wiebeler

Company: AWS EMEA SARL

Address: Oskar-vin-Miller-Ring 20, 80333 Munich, Germany

Email: schuant@amazon.de

Website: https://aws.amazon.com/

Author Name: Florian Wiese

Company: Bayer AG

Address: Aprather Weg 18a, 42113 Wuppertal, Germany

Email: florian.wiese@bayer.com

Website: https://www.bayer.com

Brand and product names are trademarks of their respective companies.