PharmaSUG China 2026 - Paper AI-194

From AI-Ready Metadata to Intelligent Agents: Enabling Human-in-the-Loop Coordination in Clinical Programming

Yanan Sui, Mingliang Fan, Tingting Zeng, Yaohui Zhu and Nana Yang, BeOne Medicines

ABSTRACT

As AI become increasingly integrated into clinical programming workflows, the effectiveness of automation depends not only on code generation, but also on metadata quality, output transparency, and reliable human oversight. In ADaM development, derivation rules captured in specifications and associated programming comments are often maintained manually and written with varying levels of consistency, which limits their interpretability, reusability, and adaptability in AI-augmented environments. In addition, review and debugging remain time-consuming and cognitively demanding processes, often creating a bottleneck in AI-assisted programming workflows.

This paper proposes an integrated framework that links AI-ready metadata with human-in-the-loop automation for intelligent ADaM program development, review, and debugging. Using real-world ADaM examples from ACIRA (*ADaM Code Intelligence for Rapid Analysis*), we demonstrate how structured metadata enables derivation interpretation, code generation, and lifecycle adaptation as study requirements evolve.

Building on this foundation, the framework introduces review and debug agents to support quality assessment and issue resolution: the review agent provides structured feedback and validation outcomes to support logic verification and result evaluation, while the debug agent analyzes execution logs and generates context-aware correction suggestions.

Together, the proposed metadata-centered framework with agent-assisted mechanisms provides a scalable approach to intelligent clinical programming automation. It improves the consistency and maintainability of ADaM derivation logic, enhances reliability of automated outputs, and ensures human supervision at critical decision points. More importantly, it demonstrates that sustainable AI integration in clinical programming requires both machine-readable metadata and interactive mechanisms for continuous review, debugging, and refinement.

BACKGROUND

The traditional ADaM (Analysis Data Model) programming process is often time-intensive and laborious because of complex derivation logic and study-specific requirements. To address these challenges, the SpecMaster system provides structured and standardized specifications, while ACIRA (*ADaM Code Intelligence for Rapid Analysis*) system centralizes code metadata and supports AI-assisted code generation. Together, these systems facilitate streamline ADaM development and reduce the manual effort required for programming activities.

Despite this progress, several challenges remain during the transition toward AI-assisted ADaM programming:

- **Handling complex or implicit logic:** AI can generate SAS code based on specification content, but output quality may decline when derivation rules involve complex logic, unstated assumptions, or implicit conditions.
- **Maintaining effective human review:** Although ACIRA provides a visual code review interface, users must still examine, validate, and adjust AI-generated code to ensure accuracy and compliance with programming standards.
- **Detecting execution issues earlier:** Some problems are not identified until the debug run stage, which can delay issue resolution and increase the burden on code updates.

To address these pain points, this work aims to further optimize the end-to-end ADaM development workflow by expanding AI participation while preserving human oversight at critical decision points. The proposed approach focuses on following key capabilities: AI-ready metadata to improve the correctness and consistency of AI-generated outputs; a code review agent to support quality assessment and logic verification; and a code debug agent to submit SAS program batch run job, analyze execution logs, and

suggest targeted code modifications. The enhanced workflow connects AI-ready metadata, code generation, review, debugging, and human decision-making into a closed loop.

AI-READY METADATA

1. PURPOSE

AI-ready metadata is intended to bridge ADaM specification and SAS implementation by making derivation logic sufficiently explicit for both programmers and AI. It does not aim to turn the specification into a highly granular metadata model or a programming language. Instead, it keeps the Method and Comment fields close to existing specification practice while requiring key assumptions to be stated clearly enough to support code generation and change-aware maintenance.

2. INTERMEDIATE BRIDGE

The intermediate bridge should remain lightweight. Its role is not to decompose every method into detailed metadata fields, but to make key logic explicit enough for AI to avoid inferring critical assumptions.

- **Dependent input:** required datasets, variables, reference sources, and join keys.
- **Selection condition:** filters, population restrictions, date windows, and record-level criteria.
- **Key and merge logic:** merge grain, relationship type, duplicate handling, and unmatched records.
- **Priority rule:** how to choose among multiple matching records or candidate values.
- **Missing and fallback handling:** missing inputs, no matches, partial dates, invalid dates, fallback rules, or imputation rules.

This bridge keeps the specification concise while making the logic clear enough for automation. It reduces implicit assumptions, improves consistency between specification and code, and allows AI to identify missing information before code generation, which can lower review effort and reduce downstream rework.

3. AI-READABLE CODE DESIGN

SAS code and macros should be designed as reusable and AI-readable implementation assets. The focus is to keep the code structure clear, the logic concise, and the macro interface easy for AI and programmers to understand.

- **Clear program structure:** organize SAS programs by input preparation, record selection, merge or join, derivation, and post-processing.
- **Simple and focused logic:** keep simple derivations as readable open code, and only convert stable, repeated patterns into macros.
- **Small reusable macros:** design each macro for one derivation pattern, such as date imputation, unit conversion.
- **Explicit macro interface:** align macro parameters with explicit method information, such as input dataset, source variable, target variable, key, condition, priority rule, missing rule, unit conversion rule, and output dataset.
- **Method-to-code traceability:** preserve comments that link the generated code or macro call back to the related Method or Comment logic.

This design helps AI select suitable code patterns or macros, map Method and Comment content into macro parameters, and apply targeted updates when the specification changes. It also keeps generated SAS code easier to review, reuse, and maintain.

REVIEW AGENT

1. PURPOSE AND POSITIONING

With AI-ready metadata providing a structured representation of complex derivation logic, AI-generated SAS code can become more reliable and consistent. However, human review remains essential to verify whether the generated code is technically sound, aligned with the intended derivation, and suitable for downstream use. The ACIRA Review Agent is therefore designed as an auxiliary reviewer and risk-screening mechanism for SAS code associated with ADaM specifications. Its role is to help answer three practical questions:

- Does the code contain obvious technical or structural issues?
- Does the code align with the business intent expressed in the specification?
- If a problem exists, what correction or direction should the reviewer consider?

The Review Agent is not intended to automatically approve code for production use. Instead, it supports reviewers by identifying high-risk outputs, reducing first-pass review effort, providing structured evidence, and enabling human confirmation before any code changes are adopted.

2. TECHNICAL FRAMEWORK AND REVIEW MECHANISM

The Review Agent combines deterministic rule validation with AI-based semantic evaluation to assess SAS code from both technical and business perspectives.

The deterministic validation tools are designed according to business requirements and ACIRA system design principles. They focus on issues that can be checked reliably, including syntax and structural validity, macro parameter usage, common SAS anti-patterns, and other predefined rule-based checks. The results are standardized as evidence records, such as syntax-checker results and macro-checker results.

The AI component focuses on semantic alignment between the generated SAS code and the intended derivation logic. It evaluates whether the code correctly reflects the rules defined in the specification, including cases in which the code is syntactically valid but semantically incorrect. It can also consolidate multiple findings into a natural-language summary and suggest corrected code for reviewer consideration.

The final review output is therefore not a single model opinion. Instead, it provides a structured synthesis of AI-generated conclusions, tool-generated evidence, identified findings, and risk interpretation to support both technical assessment and reviewer judgment.

- **Summary:** overall assessment, code score, specification match score, and risk level.
- **Issues:** specific findings, explanations, and source tools where available.
- **Evidence:** tool results, source references, and supporting details.
- **Suggested correction:** AI-suggested corrected code for reviewer evaluation and possible adoption.

Based on the provided information, reviewers determine whether to accept the AI-generated assessment and suggested correction, reject the recommendation, or revise the code manually. Acceptance is appropriate when the supporting evidence is sufficient and the suggested correction is consistent with the specification and study context. Rejection is appropriate when the AI interpretation is incomplete, inconsistent with business logic, or not applicable to the specific study context.

By embedding human decision-making after AI summarization, the Review Agent strengthens—not replaces—the reviewer's role. This design allows AI to accelerate issue detection and evidence organization, while human reviewers retain responsibility for interpretation, acceptance, rejection, and final code readiness decisions.

3. BUSINESS EVALUATION

The Review Agent provides business value in following main areas, which strengthens human review rather than replacing it:

Risk screening. The agent can surface high-risk code, obvious specification mismatches, missing critical logic, and issues that are likely to affect results. This helps reviewers focus earlier on records most likely to require attention.

Review efficiency. Reviewers no longer need to start from a blank page. They can begin with a structured issue list, concise summary, risk indicator, and suggested correction. This reduces the cognitive effort required for first-pass review, especially when many variables or complex derivations must be checked.

Structured review data. Issues, risks, scores, suggested corrections, and tool evidence can be analyzed over time. These data support defect pattern analysis, identification of high-risk derivation types, evaluation of prompt and rule effectiveness, and continuous improvement of metadata quality.

DEBUG AGENT

1. PURPOSE AND POSITIONING

Even when SAS code passes the initial review, it may still fail during execution, generate warnings, or produce unexpected log messages. While the Review Agent focuses on pre-execution quality assessment, the Debug Agent addresses execution-time issues identified through SAS logs. It supports the full debugging cycle by submitting programs for execution, diagnosing log issues, generating targeted fixed suggestions, enabling human acceptance or rejection, supporting code-difference review, and allowing re-run verification after approved changes are applied. The agent is positioned as an evidence-based debugging assistant rather than an autonomous code-modification mechanism.

2. DEBUG WORKFLOW

The ACIRA debug workflow follows a multi-step pipeline:

- Set up a controlled sandbox environment for debugging runs with appropriate access control.
- Generate SAS programs and submit to batch run jobs.
- Collect SAS logs and parse them into structured event types, such as errors, warnings, notes, and macro messages.
- Summarize log events by macro, target variable, and affected code area.
- Apply deterministic debug rules to classify and prioritize actionable issues before invoking AI-based repair.
- Send actionable log events, affected code context, and related information to the AI fixed step.
- Persist structured results and present AI-generated fix suggestions to reviewers for decision-making.

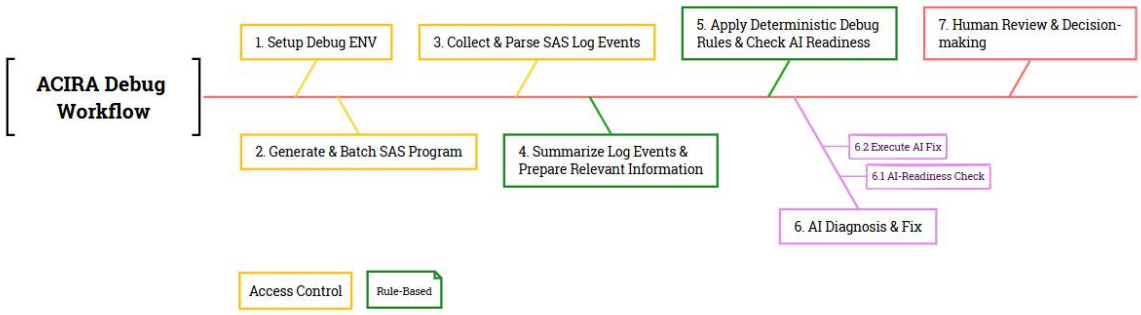

Figure 1: Debug Agent Workflow

3. HUMAN REVIEW AND APPROVAL OF ADJUSTMENTS

After automated diagnosis and preliminary verification are completed, the ACIRA workflow introduces human review before any suggested modification is adopted. Reviewers examine the summarized log issue, risk classification, affected code block, AI rationale, and proposed correction to determine whether the suggestion should be accepted, rejected, or revised manually.

This step provides an important safety mechanism. It allows reviewers to compare the original and suggested code, confirm that the change addresses the log issue without altering intended derivation logic, and decide whether a re-run is needed to verify the correction. By keeping the final approval with programmers, the Debug Agent improves debugging efficiency while preserving accountability and control.

4. BUSINESS EVALUATION

The Debug Agent provides several practical benefits:

- Faster root-cause analysis: log events are parsed, categorized, and linked to likely code blocks and variables.
- Reduced manual effort: programmers spend less time searching through logs and code to find the relevant derivation.
- Structured fix suggestions: proposed changes include rationale and code differences, making them easier to review.
- Verification loop: accepted fixes can be re-run, connecting AI suggestions to execution evidence.

FUTURE EXPECTATIONS

In the current ACIRA ADaM programming workflow, AI supports several key activities, including code metadata generation, code review, and debugging runs, while human reviewers remain involved at each step to confirm results and make final decisions. This human-in-the-loop model provides important safeguards during the early stage of AI adoption, ensuring that generated metadata, review conclusions, and debugging suggestions are interpreted in the appropriate study and programming context.

The next stage is to connect these individual capabilities into a more integrated and automated workflow. In this target state, AI would be able to execute the full process sequentially: interpreting AI-ready metadata, reviewing, validating and generating SAS codes, submitting batch jobs, reviewing execution results, diagnosing log issues, proposing corrections, and re-running the program after approved adjustments. When execution failures or quality issues are detected, the system could attempt several rounds of self-correction by using structured log evidence, affected code context, and related metadata to refine its suggested changes.

This future direction represents a shift from step-by-step human operation toward supervised automation. AI can handle repetitive execution, first-pass diagnosis, evidence organization, and preliminary correction generation, while human intervention is focused on critical decision points, such as unresolved failures after multiple repair attempts, high-risk findings, completed jobs requiring final confirmation, or changes that may affect derivation logic. Such a supervised automation model can improve efficiency and scalability while preserving the transparency, traceability, and accountability required in clinical programming.

CONCLUSION

This paper presents a metadata-centered and agent-assisted framework for advancing AI-enabled ADaM programming. By transforming implicit derivation logic into AI-ready metadata, the framework improves the interpretability, consistency, and reusability of ADaM specifications and provides a stronger foundation for automated SAS code generation. Building on this foundation, the Review Agent and Debug Agent extend AI participation beyond code creation into quality assessment, issue diagnosis, correction suggestion, and execution-based verification.

The Review Agent supports structured code evaluation and helps reviewers identify high-risk outputs and make informed accept, reject or revision decisions. The Debug Agent complements this capability by analyzing SAS execution logs, generating targeted fix suggestions, and supporting human-approved re-run verification. Together, these agents address two major bottlenecks in AI-assisted programming: reliable review and efficient debugging.

Overall, this framework demonstrates that sustainable AI integration in clinical programming requires more than automated code generation. It requires machine-readable metadata, intelligent agents, and human-in-the-loop decision mechanisms working together across the programming lifecycle. By linking metadata, generated code, review evidence, debug findings, and human decisions, the proposed approach improves transparency, traceability, scalability, and accountability while providing a practical path toward supervised automation.

CONTACT INFORMATION

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

Yanan Sui
BeOne Medicines
yanan.sui@beonemed.com