Paper ML09

AI-Powered Multiple-Agent Pipeline for Automating ADaM Dataset Generation

Lizhong Liu, Yesod AI, Inc., Sacramento CA, USA
Bo Ci, Yesod AI, Inc., Sacramento CA, USA

ABSTRACT

Preparation of Analysis Data Model (ADaM) datasets remains one of the most time- and resource-intensive activities in clinical trial programming. Although Clinical Data Interchange Standards Consortium (CDISC) standards provide structural consistency, ADaM development still depends heavily on iterative manual coding, debugging, and repetitive quality control practices. Trial-specific derivations and ambiguous specifications further limit automation.

This paper presents an AI-powered, multi-agent pipeline that automates the end-to-end ADaM derivation workflow under human supervision. The framework begins with AI-driven specification review to identify ambiguous language, implicit logic, and missing edge-case handling. After user confirmation, a code generation agent produces executable programs aligned with the finalized specification. Generated code is executed in a sandbox environment with access to Study Data Tabulation Model (SDTM) datasets. Execution errors trigger a regeneration loop incorporating runtime feedback. Successfully executed outputs undergo quality control checks before integration into the target ADaM dataset.

By mirroring the established programming lifecycle—review, implementation, debugging, and validation—while automating repetitive correction cycles, the achieves greater than 80% reduction in programming time in proof-of-concept (PoC) cases.

INTRODUCTION

Preparation of Analysis Data Model (ADaM) datasets remains one of the most time-consuming and resource-intensive components of clinical trial data analysis. Clinical programmers devote substantial effort to drafting ADaM specifications and developing the corresponding implementation code. Despite advances in standards and tooling, this process continues to rely heavily on manual coding, iterative refinement, and repetitive quality control practices such as double programming. While these approaches are effective in ensuring dataset accuracy, they introduce operational inefficiencies, extend development timelines, and consume highly skilled programming resources.

With the establishment and widespread adoption of Clinical Data Interchange Standards Consortium (CDISC) standards, a significant proportion of ADaM datasets share common structural and derivation patterns across trials. This standardization provides a strong foundation for automation. However, complete automation has remained elusive due to trial-specific variables, customized derivation logic, and protocol-driven dataset variations. These elements require nuanced interpretation and flexible implementation, which traditionally depend on human expertise.

Recent advances in large language models (LLMs) have demonstrated strong capabilities in understanding complex natural language instructions and generating executable code. As model performance continues to improve, it becomes increasingly feasible to leverage LLMs for handling trial-specific variables and dataset derivations.

In this paper, we present an AI-powered, multi-agent pipeline that automates the end-to-end workflow from ADaM specification review and refinement to executable code generation and target ADaM dataset production. By enforcing explicit, implementation-ready derivation rules and integrating code execution and validation loops, the proposed framework reduces programming time by greater than 80% in proof-of-concept (PoC) cases.

SYSTEM OVERVIEW: AI PIPELINE ARCHITECTURE

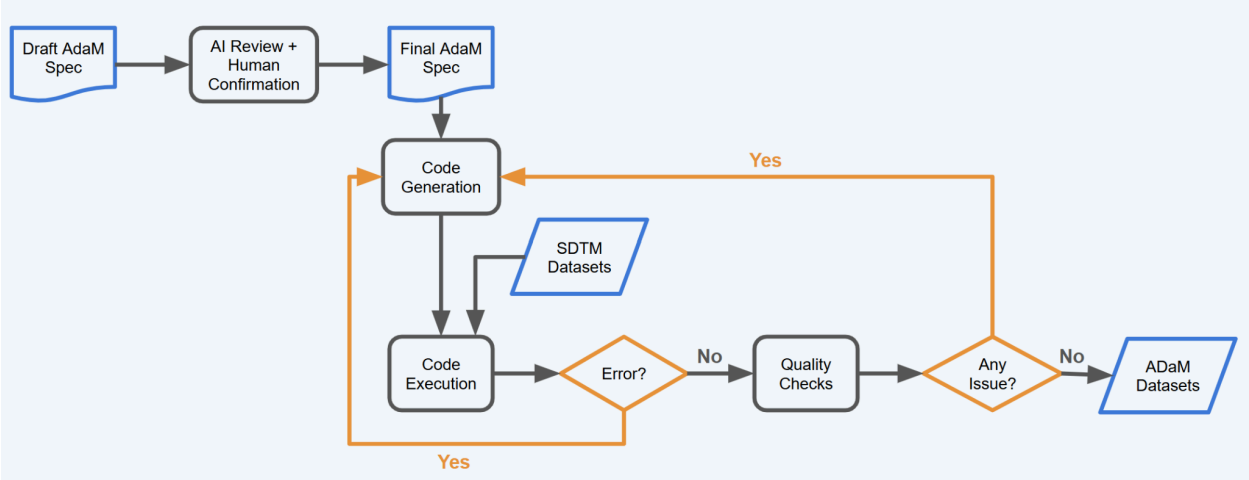
The proposed solution is implemented as a multi-agent pipeline operating under human supervision. The architecture is designed to mirror the disciplined lifecycle of clinical programming while minimizing manual intervention.

The workflow begins with the upload of an ADaM specification. The AI pipeline first performs a review of the specification to ensure that each variable derivation rule is explicit, logically completed, and implementation-ready. The objective of this step is to eliminate ambiguity before code generation begins.

Following specification refinement and user confirmation, the code generation agent produces executable programs aligned with the finalized ADaM specification. The generated code is executed within a controlled sandbox environment with access to relevant Study Data Tabulation Model (SDTM) datasets.

If execution errors occur, the system captures the error messages and routes them back to the code generation agent, triggering an automated iteration of code revision and re-execution. Once the program executes successfully, the output undergoes quality control checks. Any detected inconsistencies or rule violations initiate additional regeneration cycles or warning messages. Only variables that pass both execution and validation stages are merged into the target ADaM dataset.

This structured orchestration replicates best-practice programming workflows—including review, implementation, execution, and validation—while significantly reducing manual workload. In operational testing, the system achieved an estimated 80–90% reduction in turnaround time compared with traditional manual programming approaches.



ADAM SPECIFICATION REVIEW

Preparation of a clear and complete ADaM specification is one of the most critical bottlenecks in dataset development. Ambiguities in derivation rules frequently result in inconsistent interpretation among programmers, leading to discrepancies in implementation and outputs. In many cases, substantial time is spent reconciling differences before the root cause—imprecision in the specification—is identified.

AI-based automation introduces similar risks: LLMs can only implement logic that is explicitly defined. Implicit assumptions, incomplete derivation steps, and unaddressed edge cases may lead to incorrect or unstable outputs. Therefore, ensuring that the ADaM specification is fully explicit and logically comprehensive is a prerequisite for reliable automation.

The first stage of the proposed pipeline is a specification review process. The Review Agent systematically evaluates the uploaded ADaM specification for issues including, but not limited to:

1. Ambiguous or non-operational languages
2. Implicit derivation steps requiring inferred logic
3. Inconsistencies between variable name and identifier
4. Spelling mistakes
5. Missing logic for edge cases (e.g., handling of ties and missing values)

For each identified issue, the Review Agent proposes revised, implementation-ready languages. These suggested modifications are presented to the clinical programmer for review and confirmation. The human user remains the final decision-maker, ensuring that the specification accurately and completely reflects the trial's requirements.

EXAMPLE OF SPECIFICATION REVIEW

The example below illustrates how the Review Agent flags common issues, including typos, missing variable names, and implicit logic (e.g., sorting rules for selecting the first record and handling missing data).

Initial draft	AI suggested
Set to datetime of the first exposure observation start datetie of treatment and convert it to numeric datetime when sorted in datatime order.	Set to the numeric value of the earliest Start Date/Time of Exposure [EX.EXSTDTC] for the subject. If the subject has no exposure records, set it to null.

CODE GENERATION & EXECUTION

The core functionality of the AI pipeline is to ingest the finalized ADaM specification and generate all the ADaM variables in parallel with minimal human intervention. Each variable is processed independently through a structured lifecycle consisting of code generation, execution, and a conditional error-feedback loop.

For each variable, the Code Generation Agent produces executable code aligned strictly with the confirmed specification. The agent can reference built-in programming templates as well as user-provided templates supplied in a predefined format. This approach ensures consistency with sponsor- or organization-specific programming standards and enables systematic reuse of internal macros and code libraries.

The generated code is then submitted to the Code Execution Agent, which runs the program within a controlled sandbox environment. The sandbox has access to the relevant SDTM datasets and mimics a standard production programming setup.

If execution completes without errors, the derived variable proceeds to the quality control stage. If an execution error occurs—such as syntax errors, inappropriate function arguments, or dataset reference issues—the system captures both the code and the associated error message. These are automatically fed back to the Code Generation Agent for regeneration and correction.

This execution–error–generation loop continues iteratively until one of the following conditions is met:

1. The code executes successfully; or
2. A predefined maximum number of attempts is reached.

If the maximum attempt threshold is exceeded, the system escalates the issue to the clinical programmer. The programmer may clarify logic, adjust the specification, or directly modify the generated code. Once updated input is provided, the automated loop resumes.

This design minimizes manual copy-and-paste debugging while maintaining clear human oversight when automated correction reaches its limits.

QUALITY CONTROL CHECKS

Successful code execution does not, by itself, guarantee correctness of the derived variable. Logical errors, inappropriate actions, or edge-case misinterpretations may still exist. Therefore, an additional layer of automated quality control (QC) checks is required before integration into the final ADaM dataset.

QC procedures may include, but are not limited to:

1. Dataset dimension verification (e.g., expected number of records per subject)
2. Variable type and length confirmation
3. Missing value and NA patterns

Only variables that pass all designated QC checks are eligible for merging into the target ADaM dataset. If discrepancies are detected, the issue is routed back into the regeneration loop, triggering refinement of the code.

This structured QC stage mirrors traditional independent programming validation, but shifts the repetitive detection and correction cycles into an automated framework.

DISCUSSION

Explicit, implementation-ready ADaM specifications are essential for both code quality and overall system efficiency. Ambiguous or partially defined derivations could significantly increase regeneration cycles and reduce automation performance. For this reason, the specification review module plays a critical enabling role in the overall architecture.

The execution–error–generation feedback loop is designed to allow the system to resolve routine programming errors autonomously. Unlike static code suggestion tools, which often require multiple rounds of human–AI interaction to correct syntax or runtime issues, this closed-loop architecture incorporates real execution feedback directly into the regeneration process. By automatically capturing and responding to error messages, the system minimizes repetitive manual debugging, shortens turnaround time, and enhances the robustness of the final code. In practice, it operates as an automated junior programmer—iteratively testing, diagnosing, and refining its own output until successful execution is achieved.

This architecture also redefines the role of the clinical programmer. Rather than focusing on repetitive coding and debugging tasks, the programmer assumes a supervisory role: reviewing specifications, monitoring process status, validating outputs, and providing domain clarification when required. Direct intervention is required only when

automated regeneration fails after multiple attempts, at which point the programmer may either interact with the AI to provide additional guidance or modify the generated code directly. This shift enables programmers to redirect their expertise toward more creative, higher-value tasks.

CONCLUSION

We developed a robust multi-agent AI pipeline to support end-to-end ADaM derivation, from specification refinement to validated dataset production. By combining structured specification review, controlled code generation, execution feedback loops, and automated QC validation, the system preserves programming rigor while substantially reducing manual effort.

Operational evaluation demonstrates greater than 80% reduction in programming time for eligible derivations in PoC cases.

This framework represents a practical step toward scalable automation in clinical programming, enabling teams to increase efficiency without compromising data quality.

CONTACT INFORMATION

Name: Bo Ci
Company: Yesod AI, Inc.
Work Phone: (469)-601-7522
Email: ci.bo@yesodai.com