PharmaSUG 2026 - Paper AI-278

AI-Driven Intelligent Platforms for ADaM Specification and Code: Empowering Clinical Data Analysis

Tingting Zeng, Jia He, Yaohui Zhu, Shuang Gao, Jinling Li and Nana Yang, BeOne Medicines

ABSTRACT

Traditional ADaM (Analysis Data Model) implementations encounter critical challenges, including excessive complexity, time constraints, fragmented cross-layer traceability, and difficulty managing changes under evolving analysis requirements. To address these challenges, we have developed SpecMaster (ADaM Specification Management) and ACIRA (ADaM Code Intelligence for Rapid Analysis), two AI-driven platforms that transform clinical data analysis workflows through integrated specification management, code generation, and metadata intelligence. The framework combines centralized knowledge resources with multi-agent AI collaboration, end-to-end metadata lineage, and human-in-the-loop oversight. This design enhances efficiency, consistency, and cross-study governance while ensuring human analysts retain oversight of interpretation, validation, and final decision-making.

This paper describes the solution architecture, core capabilities, and practical considerations for applying AI in clinical data analysis while maintaining regulatory compliance.

INTRODUCTION

The ADaM serves as the cornerstone of clinical data analysis in pharmaceutical development, providing the standardized framework required for regulatory submissions to agencies such as the FDA and EMA. ADaM datasets transform raw clinical trial data into analysis-ready formats, enabling statistical analyses that support efficacy and safety evaluations critical to drug approval decisions.

Although common approaches can support operational tasks such as template management, baseline version control, and routine code reuse, several high-value challenges still require more intelligent, context-aware solutions. First, complex derivation logic often depends on integrated understanding of study protocols, SAP (statistical analysis plans), data standards, derivation logic, and statistical context beyond rule execution alone, while maintaining full traceability under regulatory scrutiny. Second, traceability across the CRF-SDTM-ADaM-TFL pipeline is frequently fragmented, limiting early identification of downstream impacts when upstream metadata or analysis requirements change, increasing the risk of rework and inconsistency across specifications, programming, and outputs. Third, evolving analysis requirements, including protocol amendments, updated regulatory expectations, and newly emerging safety or efficacy concerns, demand efficient propagation of specification and programming updates while preserving consistency and auditability.

To address these challenges, we developed SpecMaster and ACIRA, two AI-driven platforms forming an integrated framework where AI improves efficiency and consistency while human analysts retain responsibility for interpretation, validation, and final decisions. The following sections describe the solution architecture, core capabilities, implementation approach, and practical considerations.

SOLUTION ARCHITECTURE OVERVIEW

Our solution framework comprises four core pillars that work together to address the challenges outlined above. As illustrated in Figure 1, the integrated end-to-end pipeline synchronizes SpecMaster and ACIRA workflows and primarily includes:

- **A Unified Resource Hub for knowledge accumulation and dynamic optimization**, which centrally manages historical data, business logic, and rule libraries to provide high-quality inputs for AI and improve generation accuracy and reuse efficiency;
- **Multi-Agent AI Collaboration for task decoupling and strategic closed-loop execution**, in which AI agents coordinate to decompose requirements, generate rules, and execute tasks, thereby

reducing the risks of complex projects;

- **A Metadata Association Network for end-to-end lineage and change impact analysis**, which builds a metadata dependency network across SDTM, ADaM, and TFL to enable bidirectional data lineage tracing and systematic change impact analysis, supporting detection, assessment, notification, and AI-assisted decision-making with full versioned auditability;
- **A Human-AI Interactive Interface for transparent monitoring and real-time feedback**, which utilizes visualization to track AI generation steps, strengthen human-in-the-loop interaction, and ensure that quality remains controllable at critical stages.

The following sections describe these four components in detail.

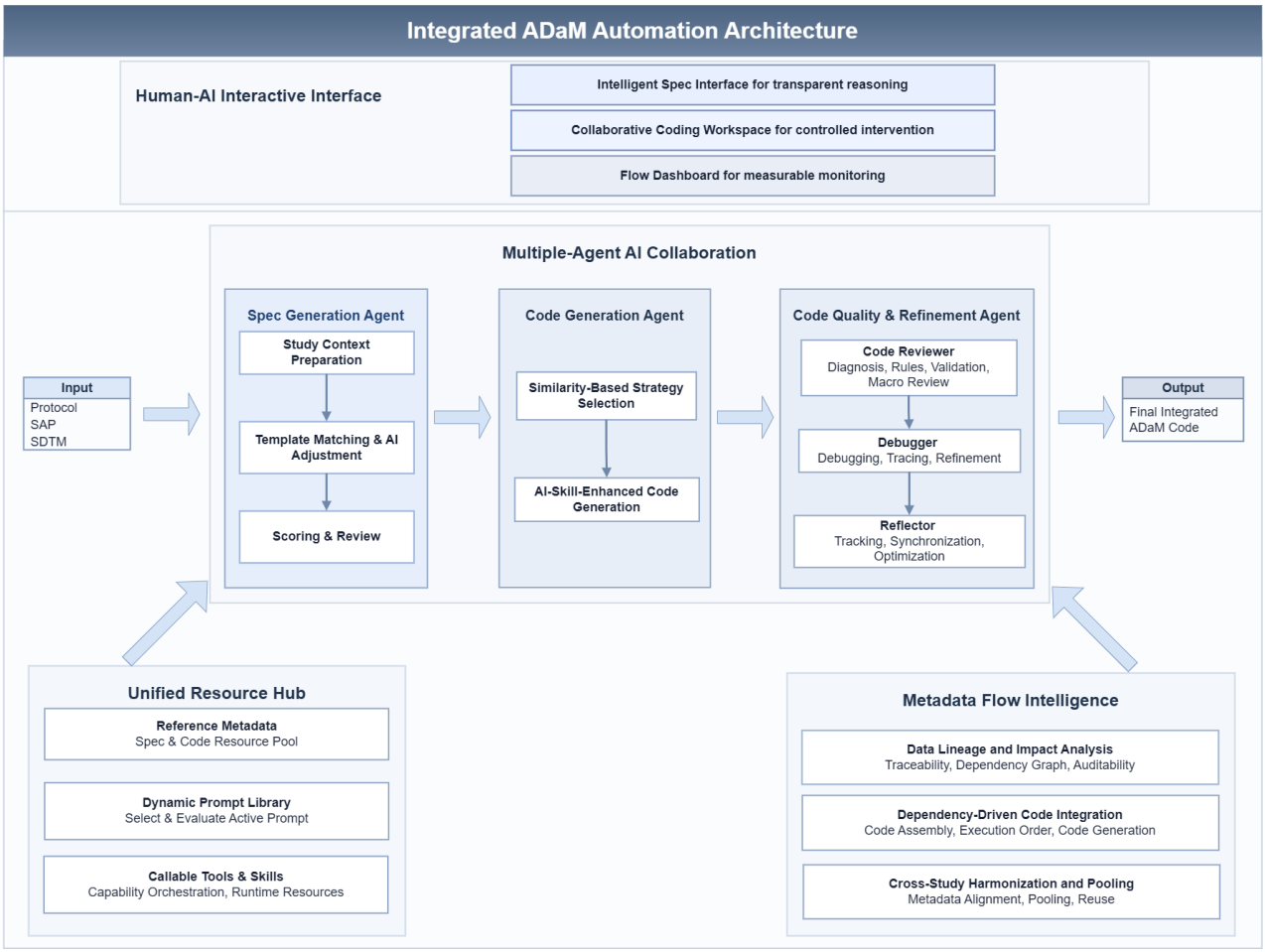


Figure 1. Integrated SpecMaster + ACIRA End-to-End Pipeline

UNIFIED RESOURCE HUB: KNOWLEDGE ACCUMULATION AND DYNAMIC OPTIMIZATION

Within this framework, a centralized resource layer connects SpecMaster and ACIRA. This layer is not merely a document repository or a simple list of reusable prompts. Instead, it serves as a governed operational foundation that accumulates reusable knowledge including template hierarchies, study documents, historical examples, prompt configurations, and review feedback from the specification authoring and code generation.

RESOURCE FOUNDATION AND RUNTIME ORCHESTRATION

Our centralized repository provides the reusable foundation for both SpecMaster and ACIRA. It includes standardized company-, therapeutic area-, and study-level templates for ADaM specification generation; scenario-specific rule resources that support context-sensitive derivation and review; curated historical specification-code examples that support example-based generation; and reusable macro or code assets for common derivation patterns. In the current architecture, however, these assets are not consumed as isolated static libraries. They are organized into a runtime resource layer that can be activated by AI at different stages of specification creation, code generation, and review, making the same resource base reusable across both platforms in a consistent and auditable manner.

DYNAMIC PROMPT LIBRARY

Another important part of this runtime layer is a dynamic prompt library. During execution, the system selects active prompt records according to task type and processing phase, then evaluates whether conditional prompt blocks should be activated. Some prompts serve as unconditional foundations, while others are injected only when rule-based or AI-powered trigger evaluation detects specific conditions, such as duration derivations, baseline selection logic, intermediate preprocessing, or other scenario-sensitive requirements. The final prompt is therefore assembled at runtime from ordered components instead of being authored as a single fixed template. This design makes the prompt layer extensible and maintainable: new rules can be added as independent records, scenario-specific safeguards can be activated only when relevant, and specialized review instructions can be introduced without rewriting the whole generation workflow.

CALLABLE TOOLS AND SKILLS

Beyond prompts, this runtime resource layer also contains callable tools and skills that allow AI to do more than generate text. These resources encapsulate executable capabilities for recurring tasks such as template-based variable construction, document retrieval, SDTM-to-ADaM evidence linking, example selection, derivation decomposition, dependency inspection, macro validation, and code review support. Some skills are narrow and scenario-specific, such as date imputation, baseline record identification, parameter-level derivation patterns for BDS datasets, or therapeutic-area-specific evaluation logic. Others serve as general orchestration tools, such as evidence routing, conflict checking, or review-time structural validation. By treating these skills and tools as managed runtime resources rather than hard-coded one-off logic, the platform allows AI agents to invoke the right capability for the right task, improving consistency, extensibility, and operational reliability across both SpecMaster and ACIRA.

Across all above resource libraries, changes to templates, prompts, tools, and skills are versioned, linked to generated outputs, and assessed against downstream review outcomes. This closes the loop between human feedback and AI behavior: resource configuration and usage strategies can be refined based on observed error patterns or reviewer intervention over time. As a result, the resource layer functions not only as a reusable knowledge repository, but also as a controlled optimization layer supporting accumulation, adaptation, and cross-study standardization across both specification generation and code generation workflows.

MULTI-AGENT AI COLLABORATION: TASK DECOUPLING AND STRATEGY CLOSED-LOOP

Our platform employs multiple specialized AI agents that collaborate to handle different aspects of ADaM development, each with clearly defined agents and feedback mechanisms.

ADAM SPECIFICATION GENERATION (SPEC CREATOR)

The Spec Creator agent reimagines specification development as an intelligent, multi-stage pipeline—transforming scattered clinical documents and reusable templates into evidence-backed, reviewer-ready ADaM specifications. As illustrated in Figure 2, this process operates through three sequential stages: Study Context Preparation, Template Matching and AI Adjustment, and Scoring and Review.

Study Context Preparation

The system builds the study context required for specification generation. It parses Protocol and SAP documents using semantic chunking and indexing, employing heading-level semantic matching combined with content-level dense retrieval to support evidence retrieval. The system identifies study structure including treatment type, study design, and treatment arm configuration, then extracts parameter values needed for ADaM derivations, such as population definitions, TEAE windows, and baseline rules. It also identifies which SDTM datasets and variables serve as sources through historical pattern matching and method-text parsing.

Template Matching and AI Adjustment

With study context prepared, the system generates candidate specifications. It selects appropriate templates from multiple template pools including company-level, compound-level, and historical studies with similar characteristics. A Context Builder assembles relevant context for each candidate, and a Scenario Detector classifies the adjustment task to provide focused instructions. The AI then replaces placeholders in template methods with study-specific values, with each adjustment linked to supporting evidence. The system outputs candidate specifications for each variable, annotated with source template and linked evidence.

Scoring and Review

AI generates confidence scores for each candidate based on evidence quality and adjustment complexity. Reviewers compare candidates' side-by-side, verify linked evidence, and accept, modify, or reject recommendations. Approved specifications proceed to code generation.

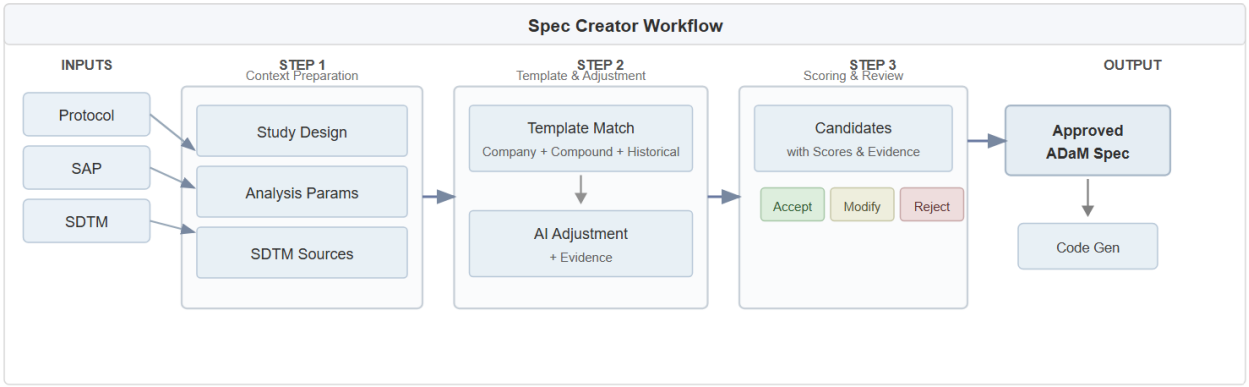


Figure 2. Spec Creator End-to-End Pipeline

As a complementary illustration, Figure 3 walks through a concrete example of the Spec Creator workflow: the system parses the SAP content and identifies the relevant SDTM datasets, selects the appropriate variable (EVLVSFL – Evaluable Vital Signs Flag) from the template pool, tailors the derivation rules based on the SAP-defined criteria, and presents supporting evidence for user review and validation.

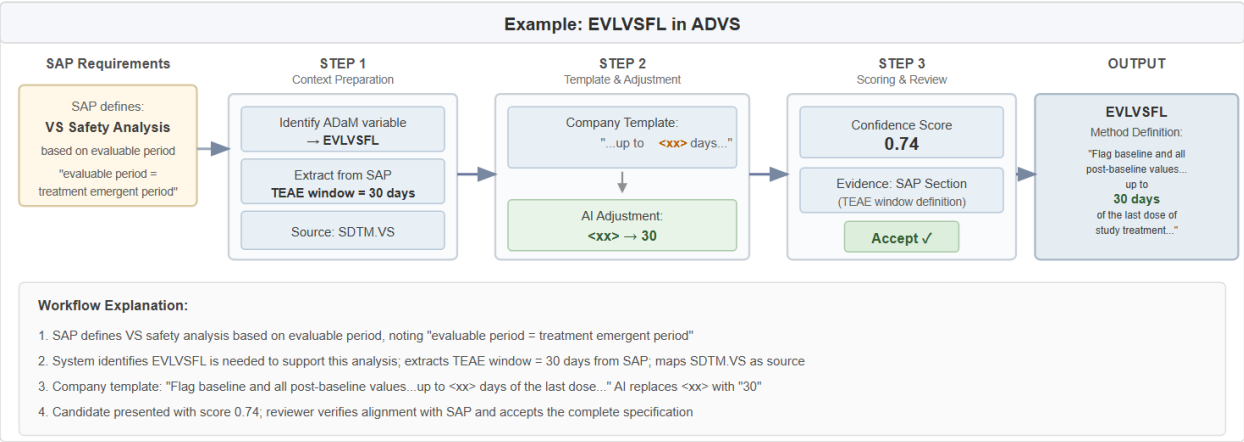


Figure 3. Spec Creator End-to-End Example: EVLVSFL in ADVS

AI-ASSISTED CODE GENERATION (CODE GENERATOR)

The Code Generator agent transforms approved specification methods into SAS code snippets through an adaptive workflow. Rather than rigid categorization, the system dynamically adapts its strategy based on how closely each derivation target matches historical examples and leverages pre-configured AI skills when complex derivations are encountered.

Similarity-Based Strategy Selection

The code generation process combines similarity evaluation (multi-layer intelligent matching) with adaptive strategy selectin to produce target code, as illustrated in Figure 4.

In the evaluation stage, the system assesses each input specification method against the method-code pair database through a multi-layer scoring mechanism. Raw similarity scores are refined by considering dataset type alignment, variable name matching, and study hierarchy relationships, applying incremental relevance adjustments for matches from the same study, best reference studies, or studies under the same compound. The resulting normalized scores provide a consistent basis for downstream strategy selection.

In ADaM programming, variable derivations range from standardized implementations to complex, study-specific logic, and a single generation approach would either over-engineer simple tasks or under-serve complex ones. The system therefore dynamically selects strategies based on the evaluated similarity. For high-similarity matches, the system performs differential analysis between the target and best-matching examples, identifying where derivation logic and attributes diverge, and recommends targeted adjustments to produce the final code. For lower-similarity cases, the system extracts derivation requirements and, depending on complexity, enriches context with pre-configured workflows or skills from system resources.

Across both strategies, pseudocode serves as a critical intermediate artifact that captures source data, variable dependencies, and step-by-step derivation logic in a language-agnostic form—ensuring transparency before final code generation and enabling translation into different target languages such as SAS or R. Generated code then undergoes systematic review through the Review Agent, producing a review-passed, integration-ready code snippet.

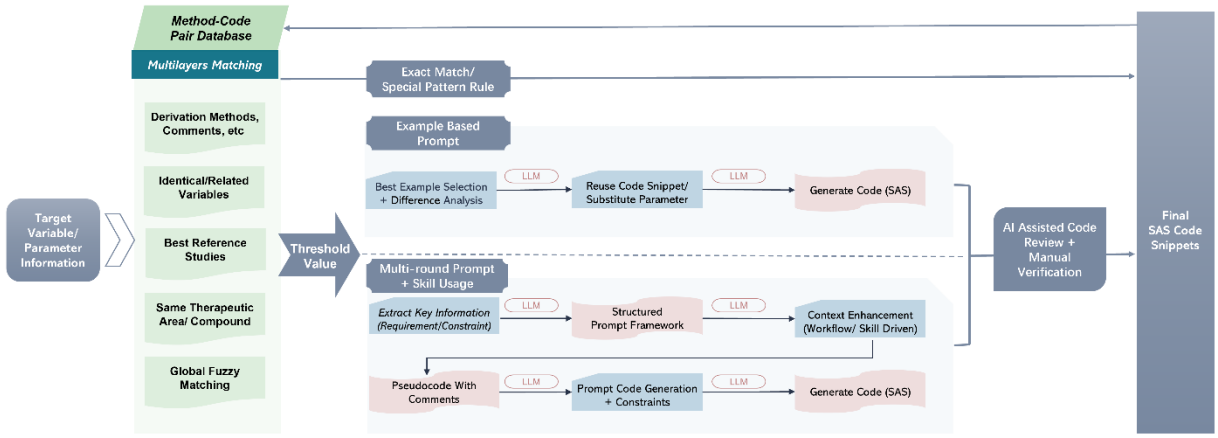


Figure 4. AI Workflow for Code Generation—From Specification Method to Final SAS Code Snippet

AI Skills for Complex Derivations

While example-based or prompting strategies are sufficient for many ADaM variable derivations, certain complex scenarios require more dynamic handling. AI Skills are modular, task-specific capabilities designed for situations where derivation rules require additional reasoning or resource retrieval.

AI Skills are typically invoked in the following situations: when derivation logic is influenced by therapeutic area conventions or study design characteristics; when derivations depend on project-level resources such as study-specific SDTM variables or codelists; or when complex but well-established processing rules—such as missing date imputation—require referencing pre-defined macros or rule modules.

Figure 5 illustrates the structure of a derivation skill using Dose Intensity parameters as an example. This condensed version just highlights the design concept—parameter dependencies, decision points, and generic skill calls—while the production version would contain more information such as derivation logics, edge cases, references and examples.

Dose Intensity Parameters Derivation Skill (Concept Version)

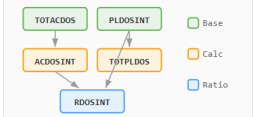
// Condensed illustration. [...] indicates detailed logic in full version.

1. Purpose

Guide AI to derive dose intensity parameters for ADEXSUM, determining appropriate source variables, decision rules, and derivation steps based on study-specific dosing characteristics.

2. Parameters and Dependencies

PARAMCD	PARAM	Dependency
TOTACDOS	Total Actual Dose	Base
PLDOSINT	Planned Dose Intensity	Base
ACDOSINT	Actual Dose Intensity	TOTACDOS
TOTPLDOS	Total Planned Dose	PLDOSINT
RDOSINT	Relative Dose Intensity (%)	ACDOSINT, PLDOSINT



3. Information Extraction

Information	Source	If Missing
Derivation methods	ADaM Spec	-
Dosing frequency (ECDOSFRQ)	SDTM EC	Check TS or Protocol
Dose unit (ECDOSU)	SDTM EC	-
Cycle length, gap days	ADaM Spec	Clarify from Protocol

4. Key Decisions

Decision 1: Dosing Schedule → trt_dur

```
IF ECDOSFRQ in (QD, BID, ...)
   → Daily: last_dt - first_dt + 1
IF ECDOSFRQ in (Q3W, Q4W, ...)
   → Cycle: last_dt + gap - first_dt + 1
```

Decision 2: Unit Conversion

```
IF ECDOSU = "mg/kg"
   → need baseline Weight
IF ECDOSU = "mg/m²"
   → need baseline BSA
   CALL Baseline_Value_Skill
```

5. Generic Skill Calls

Skill	Trigger	Purpose
Date_Imputation_Skill	ECENDTC missing	Derive last dose date
Cutoff_Handling_Skill	date > cutoff	Truncate to cutoff
Baseline_Value_Skill	unit = mg/kg or mg/m²	Get Weight/BSA for conversion

6. Derivation Flow (ordered by dependency)

```
/* ── Step 0: Preparation ── */
0. Extract dosing info → classify schedule, check unit
   trt_dur = treatment duration [via Decision 1]

/* ── Level 1: Base Parameters ── */
1. TOTACDOS = sum(ADEX.ADOSTOT) [...conversion if needed]
2. PLDOSINT = planned dose per time unit [...from Spec/Protocol]

/* ── Level 2: Calculated Parameters ── */
3. ACDOSINT = TOTACDOS / trt_dur
4. TOTPLDOS = PLDOSINT × trt_dur

/* ── Level 3: Ratio Parameter ── */
5. RDOSINT  = ACDOSINT / PLDOSINT × 100
```

Edge Cases: ECDOSE=0 exclusion, missing dates handling [...apply at each step as needed]

7. Output Format (per parameter)

```
PARAMETER: [PARAMCD] - [PARAM]
Sources: [...] | Depends: [...] | Calls: [...]
Derivation: [pseudo-code steps...]
Edge Cases: [...]
```

[Full version contains complete derivation logic for all parameters with drug-specific rules]

Figure 5. Example of a Derivation Skill (Dose Intensity Parameters)

CODE QUALITY AND REFINEMENT (REVIEWER, DEBUGGER AND REFLECTOR)

The **Code Reviewer** agent performs automated quality assurance after each code snippet is generated. By evaluating derivation rules and code context, the agent dynamically invokes pre-configured checks to either approve the code or suggest revisions. Its core capabilities include **defect diagnosis** that automatically identifies structural/syntax errors, specification violations, and other quality risks; **configurable rules** that dynamically load customized inspection strategies from the system design to improve adaptability; and **intelligent remediation** that provides correction suggestions with closed-loop resolution.

To enhance review effectiveness, a suite of specialized tools is under development, including **dependency analysis** to inspect dataset and variable dependency graphs and identify conflicts or risks; **logic validation** to reverse-check code against ADaM rules and supplementary documentation; **design compliance** to validate code against the system's pattern library; and **custom macro review** to verify macro applicability and parameter compliance.

Other advanced agents are currently under development to further bolster the framework. The **Code Debugger** agent executes code for multi-dimensional diagnosis and error tracing, learns from historical debugging cases to refine rules, and supports human-in-the-loop intervention triggered by complexity assessment. Meanwhile, the **Code Reflector** agent monitors manual correction records to track changes, synchronizes updated code with rules to ensure consistency, and optimizes code generation/review prompts and tools based on historical modification patterns.

METADATA FLOW INTELLIGENCE: DATA ASSOCIATION NETWORK CONSTRUCTION AND APPLICATION

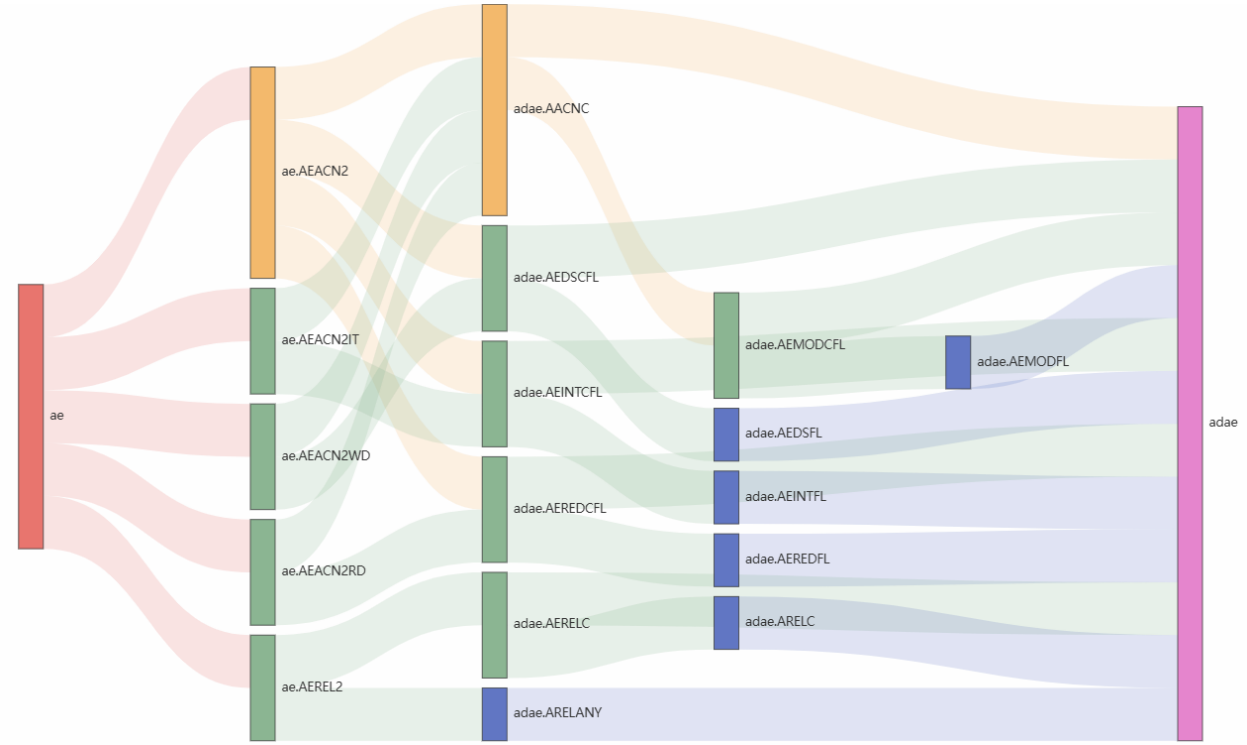
Our platform establishes a metadata relationship network that systematically captures the dependencies among variables across the SDTM, ADaM, and TFL layers. This interconnected graph serves as the foundation for a range of intelligent data lineage capabilities, including end-to-end change impact tracing, automated code assembly, and cross-study metadata harmonization.

DATA LINEAGE TRACEABILITY AND CHANGE IMPACT ANALYSIS

A direct application of the metadata association network is end-to-end data lineage visualization. The platform renders interactive Sankey diagrams that map variable and value level dependencies from SDTM through ADaM to TFL outputs, enabling bidirectional traceability—any ADaM variable can be traced upstream to its SDTM sources or downstream to its TFL usages.

Leveraging the dependency graph, the platform provides systematic change impact analysis through four integrated functions: (1) **Detection**—real-time monitoring of changes in upstream specifications or source data; (2) **Assessment**—automated evaluation of downstream impact using the association network; (3) **Notification**—targeted alerts to relevant team members with periodic summary reports; and (4) **Decision support**—AI-assisted recommendations for change propagation informed by historical patterns. These capabilities are underpinned by a versioning infrastructure that maintains timestamped transaction logs and automatic history snapshots for all version-controlled metadata objects, ensuring full auditability of every change event.

Figure 6 illustrates this with a concrete example in which variables are removed from an upstream SDTM domain. The system traverses the dependency graph to identify all affected variables in the corresponding ADaM dataset (e.g., ADAE)—both direct dependencies and indirect impacts through derivation chains. Beyond the SDTM–ADaM linkage, the full traceability chain—extending upstream to CRF collection and downstream to TFL outputs—is also maintained and queryable within the platform.



Note: above figure contains the SDTM variables for action taken (e.g. AEACN2IT/AEACN2WD/AEACN2RD for Drug Interrupted, Withdrawn or Dose Reduced) or causality (e.g. AEREL2) to specific drug, the directly impacted ADaM variables (e.g. AEINTCFL/ AEDSCFL/AEREDCFL/AERELC) and indirect impacted ones (e.g. adae.AEMODCFL/ARELC/AEMODFL) through derivation chains.

Figure 6. Example of SDTM Change Impact Propagation to ADaM Variables

CODE INTEGRATION AS PART OF DATA NETWORK

Beyond visualization and change analysis, the metadata association network also underpins automated code integration. The variable and value-level snippet design supports flexible and maintainable metadata management but simply concatenating these fragments cannot resolve cross-variable dependencies, shared intermediate logic, or proper execution ordering. A dependency-driven synthesis workflow is therefore required.

As illustrated in Figure 7, the integration process operates in two stages. The pre-integration assessment stage leverages the dependency network to analyze input dataset requirements, format and macro dependencies, intermediate dataset needs, and derivation ordering constraints. The dependency-guided assembly stage then resolves both explicit and implicit relationships—particularly those introduced by shared derivations and macro-based transformations—and assembles derivations in dependency order into a complete SAS program with header initialization, format generation, ordered variable derivation blocks, and post-processing steps. Elements derived directly from the dependency network are highlighted in red in Figure 7.

This ensures that integrated programs remain structurally traceable to the metadata network, closing the loop from metadata management to submission-ready program delivery.

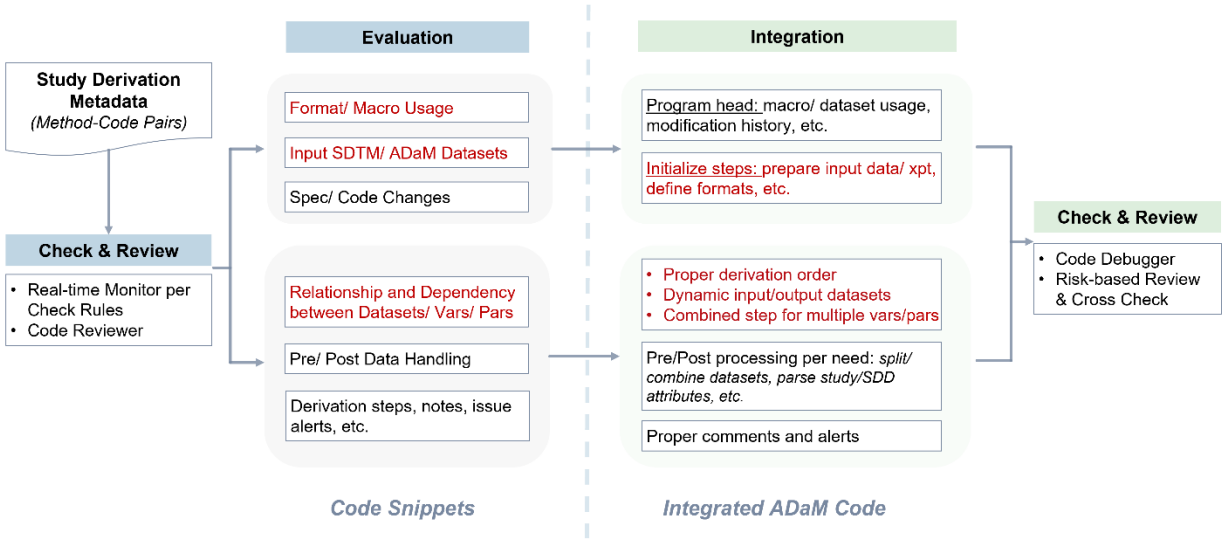


Figure 7. Code Evaluation and Integration Process

CROSS-STUDY METADATA HARMONIZATION AND POOLING

The metadata association network extends naturally from single-study operations to multi-study pooling scenarios such as DSUR, IB and ISS analyses. The system leverages dependency structures across studies to compare variable or value-level structures and derivation logic, identifying consistencies for direct merging and discrepancies requiring reconciliation. These results guide pooling specification design and inform structured code organization—determining where shared derivation blocks can be reused and where study-specific processing must be isolated. The cross-study traceability can also support ongoing change impact analysis, tracing how upstream study-level modifications propagate to pooled datasets. This transforms cross-study pooling from labor-intensive manual comparison into a network-driven, traceable workflow.

HUMAN-AI INTERACTIVE INTERFACE: TRANSPARENT MONITORING AND REAL-TIME FEEDBACK

Clinical data analysis demands high accuracy, traceability, and auditability. While AI generation and integration operate largely in the backend, a well-designed interactive interface is essential to make AI outputs **transparent**—so users can see the reasoning and evidence behind recommendations, **controllable**—so users can govern key decisions at critical points, and **measurable**—so project progress, quality metrics and AI performance patterns can be monitored to drive continuous optimization.

The interface comprises three components. The **Intelligent Spec Generation Interface** presents AI reasoning and confidence indicators during rule creation, enabling users to evaluate recommendation rationale and approve or revise suggested specifications before downstream processing begins. The **Collaborative Coding Workspace** provides an integrated environment where developers can review AI-generated code against approved specifications, intervene in the generation and integration process, track metadata changes, and submit feedback that feeds into continuous AI improvement. **The Metadata Flow Dashboard** visualizes end-to-end data lineage and provides project-level monitoring of progress, quality metrics, and change impacts across teams. Across all components, review checkpoints are embedded at critical stages with risk-based escalation, ensuring that high-risk items receive focused human attention while lower-risk items follow a streamlined process.

TECHNICAL FRAMEWORK

SpecMaster and ACIRA are built as full-stack web applications as illustrated in Figure 8. The backend is developed in Python 3 using the Flask web framework, with SQLAlchemy as the ORM layer and Microsoft

SQL Server as the production database. Celery with Redis handles asynchronous task processing for long-running operations such as batch code generation and specification synchronization. The frontend is a web application built with Vue.js and Element UI, providing a rich interactive interface for specification editing, code review, and batch monitoring.

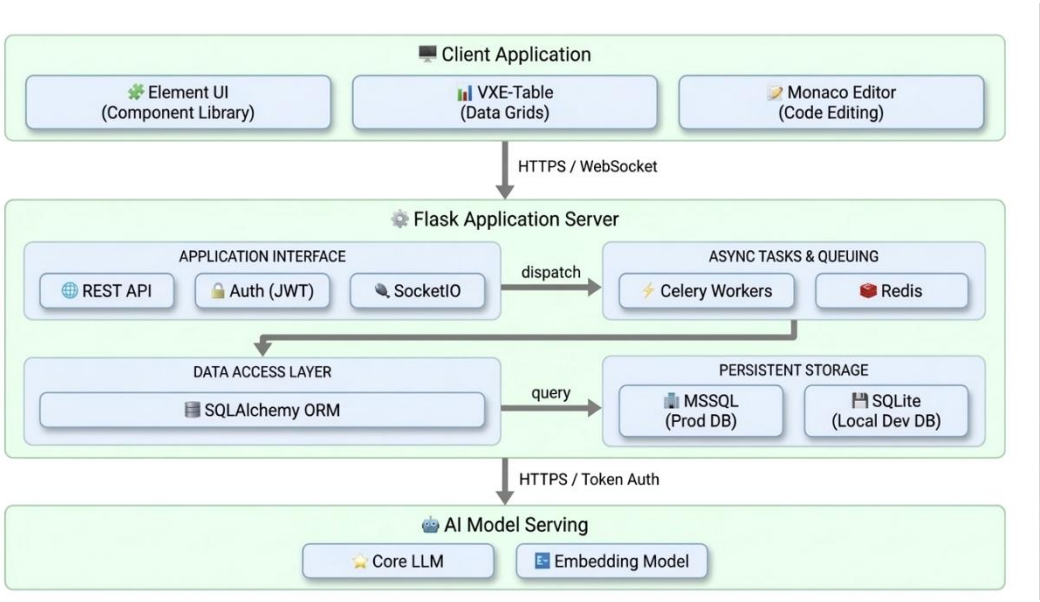


Figure 8. System Architecture

CHALLENGES AND CONSIDERATIONS

Deploying AI in regulated clinical data workflows poses inherent challenges: AI outputs lack guarantees of accuracy or consistency, task complexity varies across ADaM dataset types and individual variables, and existing processes and resources should be accommodated and leveraged. These challenges shape several design considerations. **Human-AI collaboration should be architected from the outset**, with task decomposition, checkpoint placement, and designed automation boundaries—current practice requires human confirmation at critical points while evolving toward risk-stratified autonomy. **The depth of domain knowledge embedding largely determines how effectively AI capabilities are realized**, distributed across reference libraries, prompt design, tool definitions, and evaluation criteria. **A reliable evaluation loop is the foundation for continuous improvement**, covering accuracy, confidence calibration, and identification of cases needing human intervention. Additional considerations include selecting between rule-based and AI-driven approaches per sub-task, surfacing retrieved evidence rather than AI-generated rationale for human verification, and maintaining resource library quality as projects scale.

FUTURE DIRECTIONS

Several directions guide ongoing and planned development. In the near term, continued iteration on core modules, dynamic resource library maintenance, and evaluation framework refinement will strengthen **system reliability and effective coverage**. For the multi-agent architecture, the focus increasingly shifts toward **deepening each agent's skills for complex scenarios**—accumulated through project experience across spec creation, code generation, review, and debugging—while enabling **end-to-end automation and expanding integration** across the clinical data lifecycle. Looking further ahead, we envision a **paradigm shift toward natural-language-driven analysis**, fundamentally transforming how statisticians and programmers interact with clinical data—from manually orchestrating derivation workflows to articulating analytical intent and receiving traceable, verifiable results. Underlying this roadmap, the system architecture is designed to **remain adaptive to rapid advances in model**

capabilities, allowing task decomposition granularity and validation layers to be dynamically recalibrated as underlying models evolve.

CONCLUSION

Traditional ADaM implementation faces significant challenges, including complex derivation logic requiring cross-domain expertise, fragmented traceability across the data pipeline, and difficulty propagating changes as analysis requirements evolve. Our AI-driven platforms—SpecMaster and ACIRA—address these challenges through a comprehensive approach combining:

- **Unified Resource Hub**: Enables standardization and reusability through well-structured metadata, code libraries, AI prompts, and specialized skills.
- **Multi-Agent AI Collaboration**: Leverages specialized AI agents for specification generation, code creation, review, and continuous improvement through feedback loops.
- **Metadata Flow Intelligence**: Provides end-to-end visibility, impact analysis, seamless code integration, and dynamic dataset generation across the clinical data pipeline.
- **Human-AI Interactive Interface**: Ensures transparency, quality control, and effective collaboration between AI and human experts.

Our experience demonstrates that AI-driven platforms can significantly enhance ADaM development efficiency while maintaining the quality and compliance standards required for regulatory submissions. The key to success lies in thoughtful system design, appropriate human oversight, and continuous improvement based on user feedback. As AI technologies continue to advance, we anticipate further opportunities to streamline clinical data analysis workflows while ensuring the accuracy and reliability expected by regulatory agencies and patients.

RECOMMENDED READING

Lewis, P., et al. (2020). "Retrieval-augmented generation for knowledge-intensive nlp tasks." Advances in Neural Information Processing Systems (NeurIPS), 33, 9459-9474. Available at https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

Wu, Qingyun, et al. (2023). "Autogen: Enabling next-gen llm applications via multi-agent conversation." arXiv preprint arXiv:2308.08155. Available at https://arxiv.org/abs/2308.08155

Dong, Qingxiu, et al. (2022). "A survey on in-context learning." arXiv preprint arXiv:2301.00234. Available at https://arxiv.org/abs/2301.00234

Amershi, S., et al. (2019). "Guidelines for human-AI interaction." Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. Available at https://dl.acm.org/doi/10.1145/3290605.3300233

Wei, J., et al. (2022). "Chain-of-thought prompting elicits reasoning in large language models." Advances in Neural Information Processing Systems (NeurIPS). Available at https://arxiv.org/abs/2201.11903

CONTACT INFORMATION

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

Tingting Zeng
tingting.zeng@beonemed.com

Jia He
jia.he@beonemed.com

Yaohui Zhu
yaohui.zhu@beonemed.com

Shuang Gao
shuang.gao@beonemed.com

Jinling Li
jinling.li@beonemed.com

Nana Yang
nana.yang@beonemed.com