PharmaSUG 2026 - Paper AI-332

A Human-in-the-Loop AI-Assisted Framework for ADaM Standardization

Chunqiu Xia and Feiyang Du, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

Standardized ADaM (Analysis Data Model) implementations play a critical role in ensuring consistency, traceability, and regulatory compliance in clinical trial analysis. When organizations acquire clinical studies from external companies, statistical programming teams often receive ADaM SAS code and specifications developed under different programming standards and conventions. These externally developed deliverables typically require substantial manual rework before teams can integrate them into internal workflows and reuse standard Tables, Listings, and Figures (TLF) SAS programs.

This paper explores a human-controlled, template-driven framework that integrates large language model assistance, via the LLM API, into the ADaM standardization workflow for acquired studies. The framework treats internal ADaM specifications and SAS template programs as the authoritative reference and supports the transformation of both non-standard ADaM SAS code and specifications into internal standard formats. The approach focuses on structural alignment, derivation normalization, and specification reconciliation under programmer oversight rather than fully automated dataset generation, with collaboration between statistical programmers and statisticians to review and confirm derivation logic where data collection methods or internal standard interpretations differ. By producing standardized ADaM datasets aligned with internal templates, the proposed framework enables efficient reuse of standard TLF programs, improves consistency across acquired studies, reduces study-specific rework, and maintains regulatory expectations for compliant and traceable ADaM deliverables.

INTRODUCTION

THE ADAM TRANSFER CHALLENGE

In pharmaceutical development, clinical studies are frequently conducted by external Contract Research Organizations (CROs) or obtained through strategic collaborations, resulting in SAS programs and ADaM specifications that reflect the conventions, programming styles, and documentation standards of the originating organization. When these assets are transferred into an internal sponsor environment, for example, to support an interim analysis (IA), submission preparation, or integrated analysis—they must be converted to align with the receiving organization's standard operating procedures (SOPs), macro libraries, naming conventions, library references, coding patterns, and specification templates.

This transfer is not a simple formatting exercise. Rather, it requires substantial technical and domain expertise to preserve study-specific derivation logic while re-expressing it within the internal programming framework. In practice, a senior SAS programmer must typically: (1) learn the internal template structure from reference programs and specifications; (2) interpret the study-specific logic implemented in the external code; (3) map external conventions to internal standards, including variable names, labels, library references, and macro calls; (4) rewrite the SAS program according to internal coding style and structural expectations; (5) regenerate the ADaM specification workbook in the internal template format; and (6) verify consistency between the converted program and the regenerated specification.

For a typical oncology study comprising approximately 12 ADaM datasets, this process may require several weeks of dedicated programmer effort. Because the work is highly manual and involves repeated translation across multiple artifacts, it is also vulnerable to transcription mistakes, formatting inconsistencies, and logic misalignment between code and specification.

MOTIVATION FOR LLM ASSISTANCE

Large language models (LLMs) have shown strong capabilities in code understanding, structured transformation, and document generation tasks, making them promising candidates for supporting ADaM transfer activities. In principle, such models can help accelerate repetitive conversion work by learning relationships between external and internal programming conventions and by generating reformatted code and documentation based on template-driven inputs.

However, clinical programming presents a uniquely constrained environment in which automation must be applied with great caution. First, regulatory accountability requires that all generated outputs be traceable, reviewable, and suitable for Quality Control (QC) and potential regulatory inspection. Second, there is effectively zero tolerance for hallucination: invented derivations, fabricated variables, omitted dependencies, or altered analysis logic are unacceptable. Third, SAS programming contains syntax-specific risks, particularly in its comment structures, where minor errors may silently invalidate or absorb executable code. Fourth, template fidelity is critical: internal macro signatures, parameter names, section ordering, and block closures must conform exactly to sponsor standards.

For these reasons, fully autonomous LLM-based code generation is not appropriate for this setting. Instead, LLMs are better positioned as constrained transformation engines operating within a carefully designed framework. In such a framework, model behavior is bounded by structured prompts, anchored by trusted template materials, and supplemented by deterministic post-processing rules that address known failure modes.

Importantly, the ADaM transfer process also depends on human expertise beyond coding mechanics alone. Study-specific derivation logic may require interpretation of statistical assumptions, endpoint definitions, censoring rules, analysis population conventions, and sponsor-specific implementation decisions that cannot always be inferred safely from source code or specifications in isolation. Accordingly, a human-in-the-loop design is essential. Statisticians and study leads provide input on ambiguous or clinically meaningful derivation logic, while senior programmers review transformed outputs to confirm that both programming intent and analysis behavior have been preserved. Under this model, the LLM serves to accelerate repetitive and labor-intensive transformation tasks, whereas human experts retain responsibility for logic confirmation, standards interpretation, and final approval of production-ready artifacts.

CONTRIBUTION

This paper presents the design, implementation, and validation of a human-in-the-loop ADaM transfer automation framework for converting externally developed SAS programs and ADaM specifications into an internal sponsor-standard format. The proposed framework combines LLM-assisted transformation, deterministic rule-based refinement, and expert review to improve efficiency while maintaining programming quality, traceability, and regulatory reliability.

The contributions of this work are fivefold. First, we introduce a structured five-step workflow—Extract → Context → Transform → Refine → Spec Generation—that formalizes the end-to-end transfer process and identifies where LLM support can be productively and safely incorporated. Second, we define a human-in-the-loop operating model in which statisticians and senior programmers provide study-specific logic input, resolve ambiguities in derivation interpretation, review transformed outputs, and approve final deliverables prior to production use. Third, we construct a rigorous system-prompt framework that encodes key safeguards ensuring reliable adherence to syntax and template expectations. Fourth, we apply a structured post-processing layer that automatically mitigates recurrent model-generated errors and enforces consistency with organizational formatting requirements. Fifth, we establish a template-driven architecture in which the internal standard (`output_expectation`) defines how the converted program and specification should be written, while the external source materials (`input_reference`) define what study-specific content should be transferred.

In addition, the framework includes a set of validation mechanisms designed to support reliable use in practice, including pre-conversion inventory auditing, post-conversion artifact verification, comment integrity validation, key-variable cross-referencing, and human review checkpoints. Together, these components demonstrate how LLMs can be integrated into clinical programming workflows not as

autonomous coding agents, but as controlled assistants within a rigorously supervised and quality-oriented transfer process.

FRAMEWORK ARCHITECTURE

DESIGN PHILOSOPHY

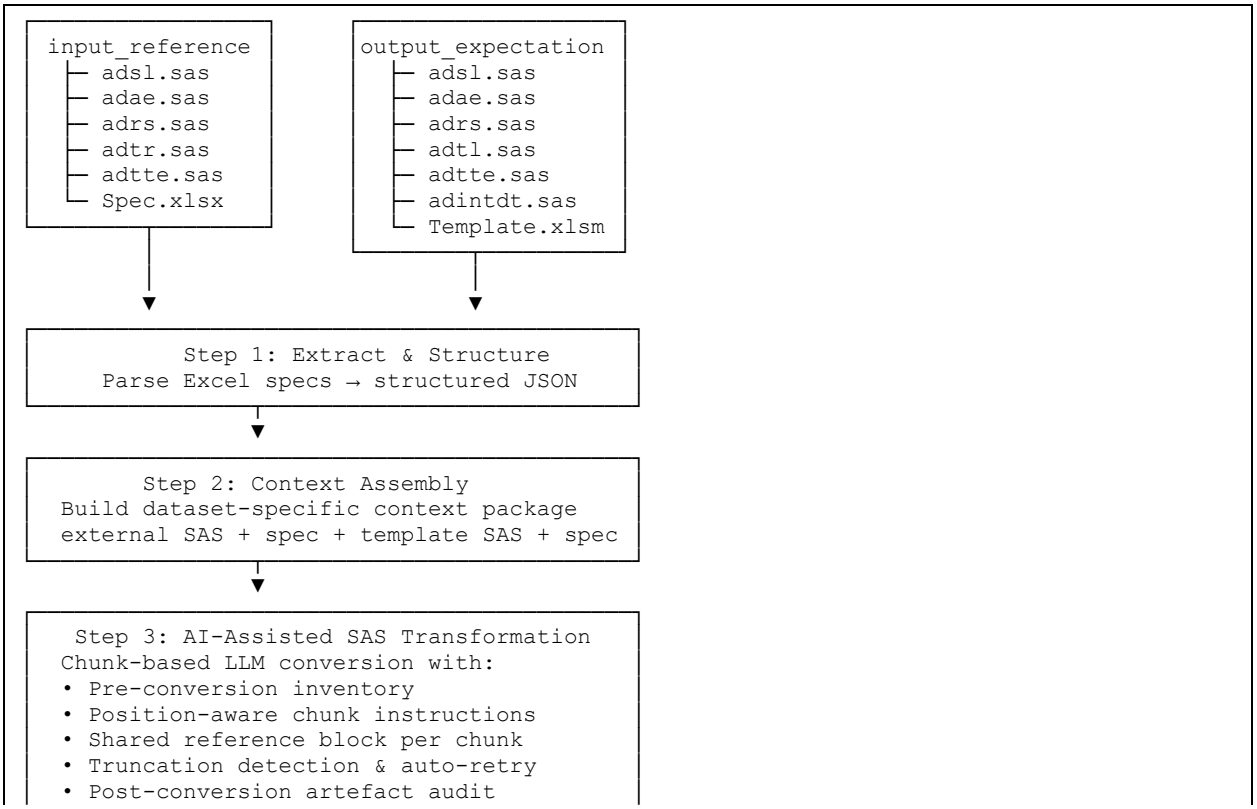
The framework enforces a strict separation of concerns:

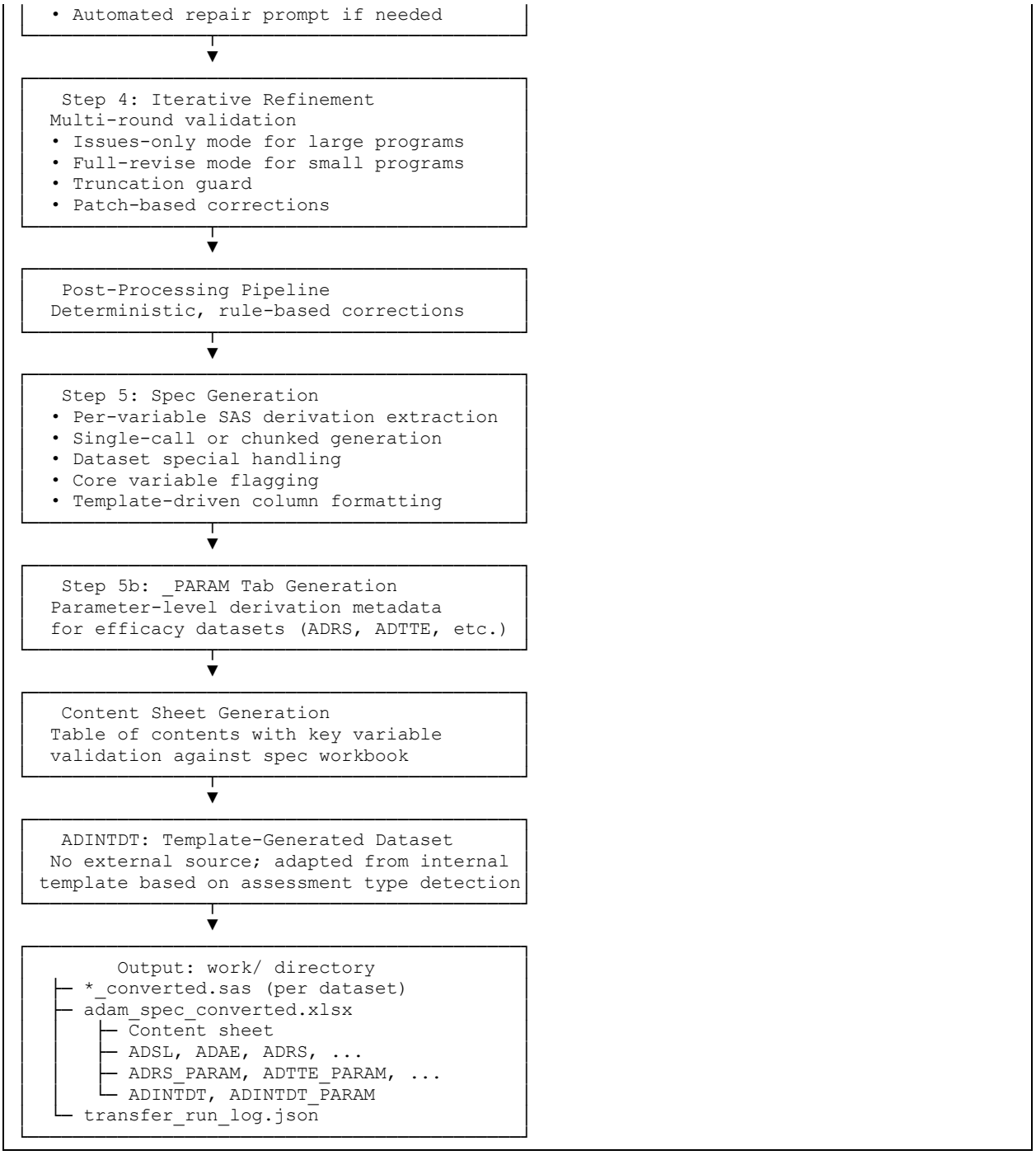
Concern	Source	Role
Content (WHAT)	`input_reference`	External SAS programs and ADaM spec Excel
Format (HOW)	`output_expectation`	Internal template SAS programs and spec Excel
Transformation	LLM (GPT)	Guided mapping under system prompt constraints
Correction	Post-processing pipeline	Deterministic rule-based fixers
Validation	Human programmer	Inventory check, comment integrity, key variable validation, stats input, final review, QC

Table 1. Design Philosophy and Responsibility Separation

This separation ensures the LLM never operates without context from both sources and never produces output that bypasses deterministic validation.

SYSTEM OVERVIEW





Program 1. System Overview Workflow

DATASET NAME MAPPING

The framework supports structural conversions where the external and internal dataset names differ. A `DATASET_TEMPLATE_OVERRIDE` dictionary maps external names to internal targets. For example, the external ADTR (Tumor Results) dataset maps to the internal ADTL (Tumor Lesion) dataset. This is not a simple rename but a structural conversion: the LLM is instructed to learn the ADTL structure from the internal template and map ADTR content into that structure.

PIPELINE STEPS IN DETAIL

EXTRACT AND STRUCTURE ADAM SPECIFICATIONS

The framework processes both the external and internal Excel specification workbooks with openpyxl in a deterministic preprocessing stage. During this step, it extracts workbook structure information, including sheet names, header row positions, standardized column headers, and row-level previews for each sheet. It also creates a cross-sheet variable index that links variable names to their source sheets. To improve computational efficiency, the framework caches the extracted workbook representations in JSON format for reuse in subsequent runs. This stage does not use the LLM.

DATASET-SPECIFIC CONTEXT ASSEMBLY

For each ADaM dataset, the framework constructs a dataset-specific context package that combines internal reference materials with external study materials. This package includes the internal template SAS program, a summary of the internal specification sheet, the external SAS program, and a summary of the external specification sheet. To remain within model input limits, long SAS programs are truncated to predefined lengths and specification sheets are represented through structured previews.

A consistent naming convention is used throughout the framework: `output_expectation` denotes the internal standard, whereas `input_reference` denotes the external source material. This distinction is preserved in all prompts to reinforce the separation between format and content, thereby helping the LLM distinguish what should be transformed from what should be retained.

AI-ASSISTED SAS CODE TRANSFORMATION

The SAS code transformation stage is the core LLM-assisted component of the framework. In this step, the external SAS program is converted into the internal programming style using the internal template as guidance. Because program length may exceed practical model limits, the external code is divided into smaller segments at logical boundaries such as the ends of data steps, procedure blocks, or `run;` and `quit;` statements.

Before transformation begins, the framework performs a pre-conversion inventory of the external program. This inventory captures key structural elements, including created datasets, macro calls, SQL output tables, merge inputs, and important variables. These elements are then incorporated into the prompt as a preservation checklist to reduce the risk of code loss during conversion.

The framework also provides position-aware instructions for each code segment. The first segment generates the opening program structure, the middle segments continue derivation logic, and the final segment completes the program and required closing blocks. If the program is short enough, the full conversion is generated in a single interaction.

To address incomplete responses, each returned segment is checked for possible truncation based on output length and missing expected elements. When necessary, the segment is further divided and resubmitted. After all segments are assembled, the combined program is audited against the original inventory, and any missing components are repaired before finalization.

ITERATIVE PROMPT REFINEMENT

After the initial transformation, the converted SAS program enters a refinement stage in which the LLM is used to identify and correct residual issues. This stage is limited to a small number of validation rounds to improve quality without introducing unnecessary instability.

Two refinement strategies are used depending on program size. For larger programs, the LLM returns only identified issues and targeted correction suggestions, rather than regenerating the full program. For smaller programs, full-program revision is allowed. This design helps reduce the risk of truncation while supporting further improvement in structural consistency and syntax quality.

POST-PROCESSING PIPELINE

Following LLM-based transformation and refinement, the converted SAS code is processed through a deterministic post-processing pipeline composed of 13 rule-based fixers. These fixers address recurrent and predictable error patterns observed during framework development, particularly those that may lead to invalid SAS syntax or deviation from internal standards.

The post-processing rules correct issues such as malformed comments, incorrect macro parameter names, unintended macro resolution, duplicated boilerplate, missing required options, and incomplete closing blocks. Final validation also checks overall comment integrity and structural completeness.

This stage reflects a central design principle of the framework: the LLM performs the semantic transformation, whereas deterministic rules are used to resolve known syntactic and structural failure modes. The combination improves both flexibility and reliability.

SPECIFICATION GENERATION

After SAS code conversion, the framework generates the ADaM specification in the internal template format. To support this step efficiently, Python-based parsing is used to extract variable-level metadata directly from the converted SAS program, including variable names, labels, formats, types, and lengths. Variable-specific derivation-related code segments are also identified to support specification authoring.

A key focus of this stage is the generation of the “Define Derivation” field. To ensure consistency, the framework applies a fixed priority order. When available, derivation logic is translated directly from SAS code into plain-language descriptions. If this is not possible, derivation text from the external specification is used. If neither source provides sufficient detail, a standardized predecessor-style reference is inserted. A review flag is used only when no adequate derivation can be identified.

For efficacy datasets with parameter-level metadata, such as ADRS and ADTTE, dataset-level and parameter-level information are generated separately. For datasets with many variables, specification generation is performed in smaller sections to preserve consistency while remaining within model limits.

CONTENT SHEET AND KEY VARIABLE VALIDATION

In addition to dataset-level specification sheets, the framework generates the workbook Content sheet, which serves as a table of contents for the ADaM specification workbook. This sheet is assembled using protocol-level information and domain metadata from the external specification, together with default descriptive text from the internal template.

Before finalizing the Content sheet, the framework validates key variables used in SAS `BY` statements against the variables retained in the converted specification. Variables that appear in external metadata but are absent from the converted structure are excluded from the final key-variable listing and recorded through warning messages. This step helps prevent downstream inconsistencies between the specification and executable SAS code.

ADINTDT: TEMPLATE-GENERATED DATASET

ADINTDT is handled differently from the other datasets because no corresponding external SAS program is available for direct conversion. Instead, the dataset is generated from the internal template and adapted to the study context using related external materials.

To support this process, the framework examines the external ADRS program and specification for indicators of assessment types, such as BICR, investigator assessment, or iRECIST evaluation. Based on the detected assessment structure, the appropriate parameter configuration is selected and applied to the internal ADINTDT template in both code and specification form.

This example demonstrates that the framework supports not only direct transfer of existing programs, but also template-based generation of study-specific datasets when no external implementation is available.

SYSTEM PROMPT ENGINEERING

PROMPT STRUCTURE

The system prompt uses structured sections to define the conversion task and constrain model behavior. These sections address the task objective, explicit definitions of input_reference and output_expectation, the principle of content preservation with format standardization, an execution workflow, special handling for ADTR-to-ADTL conversion, cross-dataset dependency rules, strict SAS transformation rules, specification-generation standards, and a final instruction that emphasizes a template-first, content-second, validate-always strategy.

SAS-SPECIFIC RULES (RULES 12–24)

These rules encode SAS syntax constraints that LLMs frequently violate:

Rule	Constraint
12	No nested block comments (`/* ... /* ... */ ... */`)
13	No semicolons inside `%*` macro comments
14	Standard macro call signatures with exact parameter names
15	Chunk header rule — only first chunk emits boilerplate
16	No orphaned comment closers (`*/` without matching `/*`)
17	No inline comments that break SAS statements
18	`%nrstr()` wrapping for unresolved macro triggers in labels
19	`options minoperator;` required for internal macros
20	`&debug` not `&_debug` for macro parameter references

Rule	Constraint
21	`%*` comments must not contain semicolons
22	Header comment must use `/* */` not `%*`
23	Macro parameter must be `adam_spec_file` (with underscores)
24	Do not generate calls to external-only macros

Table 3. SAS-Specific Prompt Rules

Each rule was added in response to a specific SAS log error encountered during validation. The rules serve as both LLM guidance and documentation of known failure modes.

API INTEGRATION AND RELIABILITY

MODEL CONFIGURATION

The framework uses an enterprise GPT endpoint configured to support stable and consistent output generation for code transformation and specification generation tasks. Model settings are chosen to favor low-variability responses and to provide sufficient output capacity for long-form structured generation. Separate completion limits are applied for transformation, repair, and specification-generation tasks according to their expected output size. In addition, the framework includes multiple retry attempts and controlled delays between calls to improve robustness during repeated API interaction.

RATE LIMITING AND ERROR HANDLING

To support reliable execution, the framework incorporates several error-handling and request-management mechanisms. Rate-limited responses are detected and retried using extended backoff intervals to reduce the likelihood of repeated request failure. Empty responses are also detected and retrieved, as they may indicate request instability or context overload. The framework further handles malformed or non-JSON responses, including unexpected HTML error pages, to ensure that downstream processing is not disrupted by invalid API outputs.

In addition, completion metadata is monitored to identify events such as truncated responses or content filtering. The framework also tracks prompt size and issues warnings when unusually large prompts may affect response quality or completion stability. Together, these mechanisms improve the operational reliability of the framework during large-scale or repeated conversion workflows.

CACHING STRATEGY

All intermediate outputs are cached to the `work/` directory:

- Step 1 spec structures (JSON)
- Step 2 context packages (JSON)
- Step 3 transformed SAS (`.sas`)
- Step 4 issues (JSON)
- Step 5 prompts and raw responses (`.txt`)
- Final converted SAS and spec outputs

A `FORCE_RERUN_SAS_TRANSFORM` flag controls whether cached Step 3+4 outputs are reused, allowing rapid iteration on post-processing without re-invoking the LLM. A `DATASETS_TO_PROCESS` set enables selective re-processing of individual datasets.

FRAMEWORK VALIDATION AND OUTPUT ASSESSMENT

DATASETS PROCESSED

The team validated the framework on an oncology interim-analysis transfer covering six ADaM datasets:

External	Internal	Lines (converted)	Type
ADSL	ADSL	~800	Subject-level
ADAE	ADAE	~700	Occurrence
ADRS	ADRS	~1,100	BDS (efficacy)
ADTR	ADTL	~320	BDS (structural conversion)
ADTTE	ADTTE	~610	BDS (time-to-event)
—	ADINTDT	~400	BDS (template-generated)

Table 5. Validation Datasets Processed

SAS CODE CONVERSION EXAMPLES

The converted outputs show that the framework performs not only syntactic rewriting but also structural adaptation to align with internal programming standards.

EXAMPLE 1: EXTERNAL CODE CONVERTED TO INTERNAL STANDARD MACRO

In one representative case, an external SAS program used a flat script without a macro wrapper or internal attribute-handling macros. After conversion, the framework reorganized the program into the internal standard structure, including a macro definition, an initialization section, standardized logging, attribute assignment, and closing logic.

```
/* EXTERNAL (adtte.sas:537) */
data lptda.adtte (label='...'
  compress=yes);
  set adtte; run;
```

```
/* CONVERTED (adtte_converted.sas:588) */
%add0attribute(...
  ,output_dataset = lptda.adtte(
    label="..."));
```

EXAMPLE 2: STRUCTURAL DATASET CONVERSION

A second example involved a dataset whose internal representation differed structurally from the external source. In this case, the framework not only reformatted the program but also adapted dataset naming, aligned the output to the internal target structure, and flagged areas where template expectations and study-specific source conventions did not fully align.

```
EXTERNAL:
─────────
output dataset: ADTR
(no macro wrapper)
structure
```

```
CONVERTED:
──────────
%macro ADTL(...);
output dataset aligned to internal ADTL

REVIEW REQUIRED markers inserted for
study-specific convention differences
```

This example highlights an important boundary of automation: although the framework can perform structural conversion and identify mismatches, certain study-specific convention changes still require expert review to confirm that the converted representation is analytically appropriate.

SPECIFICATION DERIVATION EXAMPLES

The generated specifications reflect a similar balance between automation and controlled review. When derivation logic could be identified directly from SAS code, the framework translated that logic into plain-language specification text. When the internal template required variables that were not present in the external study materials, the framework did not invent unsupported derivations; instead, it inserted explicit REVIEW REQUIRED markers so that these cases could be evaluated by programmers and statisticians.

Context. When the external specification does not contain derivation text but the SAS program includes explicit logic, the framework generates a natural-language derivation and flags the discrepancy for review.

Layer	Content
External Spec	(blank)
External SAS	enrldt=input(dsstdtc,yymmdd10.);
Converted Spec "Define Derivation"	Numeric date from DS.DSSTDTC (input(DSSTDTC,yymmdd10.)).
Developer's Notes	REVIEW REQUIRED: input spec derivation is blank; code derives from DSSTDTC.

Interpretation. This example illustrates how the framework prioritizes executable SAS logic when the external specification is incomplete, while preserving transparency through an explicit review marker.

EXECUTION AND OUTPUT VALIDATION

After post-processing, the team executed the converted SAS programs in SAS 9.4 on UNIX with generally successful results. Deterministic fixers resolved most syntax and structural issues, while reviewers retained a limited number of study-specific findings for human assessment. The post-processing pipeline consistently corrected recurrent issues that prompt constraints alone could not fully prevent, which supports the value of a hybrid LLM-plus-rule-based approach. The framework also generated a complete ADaM specification workbook with validated metadata structure and standardized formatting for downstream use.

HUMAN-IN-THE-LOOP VERIFICATION

Although the proposed framework automates substantial portions of SAS program conversion and specification generation, human verification remains an essential component of the overall workflow. In regulated clinical programming, certain derivation decisions cannot be determined reliably from source code and metadata alone, particularly when the external implementation is incomplete, ambiguous, or based on sponsor-specific analysis conventions. Accordingly, the framework is designed not as a fully autonomous conversion system, but as a human-in-the-loop process in which LLM-generated outputs are reviewed and validated by subject-matter experts before acceptance.

Human verification is especially important for variables whose derivation requires study-level interpretation rather than straightforward structural translation. A representative example arises in ADAE, where the external data structure may record adverse events at the level of individual periods or fragments rather than as complete analysis episodes. Under the internal ADAE standard, these period-level records may need to be collapsed into a single analysis record representing the full event episode. In this setting, variables such as AEACN, AEREL, and AEOUT cannot always be transferred directly, because their final analysis values may depend on how information from multiple period-level records should be combined and prioritized within the collapsed record. The appropriate derivation may depend

on statistical conventions, medical review expectations, or sponsor-defined rules that are not fully recoverable from the external program or specification alone. In such cases, the LLM may identify candidate transformation patterns, but it cannot be assumed to determine the intended collapse logic with sufficient reliability. Statistical input is therefore needed to define the expected analysis-level derivation, and programmer review is required to confirm that the converted implementation reflects that intent accurately.

More broadly, human review is applied at multiple checkpoints throughout the framework. Statisticians or study leads provide guidance on ambiguous analysis logic, senior programmers review transformed SAS code for fidelity to internal standards, and generated specifications are checked for consistency with both the converted code and the intended analysis design. This verification process ensures that automation improves efficiency without displacing expert judgment in areas where regulatory accuracy and study-specific interpretation are critical.

CONCLUSION

The proposed framework is designed to use the LLM as a guided transformation engine rather than an autonomous code generator. Inputs are anchored in external study programs, specifications, and internal templates; outputs are constrained by preservation requirements and template-defined structure; deterministic post-processing resolves predictable implementation errors; and final acceptance remains subject to human review. This design reflects a central finding of the work: LLMs and deterministic rules are complementary. The LLM is well suited to interpreting derivation intent, translating between external and internal conventions, and generating draft code and specifications, whereas deterministic fixers are more reliable for enforcing exact syntactic and structural requirements in SAS. For this reason, the framework adopts a layered approach in which prompt design reduces error frequency, rule-based fixers correct recurrent failure modes, and validation checks confirm output completeness. Equally important, the framework is explicitly human-in-the-loop. In regulated clinical programming, some derivations require study-specific interpretation that cannot be recovered reliably from code and metadata alone. This is particularly evident in ADAE, where external adverse event data may be recorded as individual periods, while the internal standard requires those periods to be collapsed into a single analysis episode. Under such circumstances, variables such as AEACN, AEREL, and AEOUT may depend on sponsor-defined prioritization rules or statistical conventions that require direct input from statisticians and confirmation by programmers. Human oversight is therefore not an exception but a design requirement of the framework. Within these boundaries, the approach can still provide meaningful efficiency gains, with an estimated 60–70% reduction in time required to prepare an initial draft. Nevertheless, several limitations remain, including the need for manual handling of highly complex derivations, context-window-related chunking constraints for large programs, dependence on the completeness of internal templates, and the lack of fully automated SAS execution and log interpretation. Future work may improve the framework through automated log parsing, cross-dataset consistency validation, incremental dataset-level reprocessing, and extension of the template-driven methodology to additional therapeutic areas.

REFERENCES

1. CDISC Analysis Data Model (ADaM) Implementation Guide, Version 1.3.
2. FDA Guidance for Industry: Providing Regulatory Submissions in Electronic Format, 2024.
3. ICH E9(R1): Addendum on Estimands and Sensitivity Analysis in Clinical Trials.
4. SAS 9.4 Language Reference: Concepts — Macro Language Elements.

CONTACT INFORMATION

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

Name: Chunqiu Xia
Enterprise: Merck & Co., Inc., Rahway, NJ, USA
E-mail: chunqiu.xia@merck.com

Name: Feiyang Du
Enterprise: Merck & Co., Inc., Rahway, NJ, USA
E-mail: feiyang.du@merck.com