From SAP to TFL: An AI-Assisted Workflow Framework for Automated Shells and TFLs

Xiaoyu Ding, Guoqing Zhao, Huafeng Gao, and Zhiyuan Guo

Keymed Corporation

ABSTRACT

The core deliverables in clinical trial statistics and programming typically include the Statistical Analysis Plan (SAP), statistical analysis shell templates (Shells), analysis programs, standardized macro libraries, controlled terminology, SDTM and ADaM datasets, and the final Tables, Figures, and Listings (TFLs) for clinical study reports and regulatory submissions. In the traditional workflow, programmers must repeatedly read SAPs, Shells, data specifications, and project communications to identify requirements, then manually translate these into table of contents (TOC) entries, program parameters, output formats, and validation checklists. While controllable, this process is costly, and frequently leads to missed requirements, version inconsistencies, duplicated development, and tracking difficulties—especially when managing multiple projects, frequent SAP amendments, or multi-team collaboration.

This paper presents an AI-assisted workflow framework designed for clinical statistical programming. The framework uses a structured metadata approach in which a large language model (LLM) extracts three categories of key metadata from the finalized SAP: table content metadata, table structure metadata, and table programming metadata. Through rule validation and human confirmation, these are consolidated into versioned project metadata. Confirmed metadata then drives the Tracker, standardized Shell Library, controlled terminology, macro library, and template mapping rules to enable semi-automated generation of Shells and TFL programming artifacts. This paper discusses the framework's process design, key metadata model, the interplay between Tracker and Library, quality control, validation strategy, and implementation risks. Practical experience shows that in regulated clinical statistical programming environments, the primary value of AI is not to fully replace programmers, but to embed into auditable, reusable, and governable engineering workflows—reducing repetitive labor and improving consistency and traceability across study deliveries.

Keywords: Statistical Analysis Plan; SAP; TFL; Shell; Metadata; Artificial Intelligence; Large Language Model; Tracker; SDTM; ADaM; Clinical Statistical Programming; Semi-Automated Delivery

1. INTRODUCTION

Tables, Figures, and Listings (TFLs) are essential outputs in clinical study reports, medical interpretation, and regulatory submissions. In a typical clinical trial, the statistical programming team must complete analysis dataset preparation, program development, output generation, result validation, and delivery archiving based on the SAP, Shells, SDTM/ADaM specifications, controlled terminology, and project conventions. Although many organizations have already established standard macro libraries, Shell templates, and project tracking workbooks, much work in real projects still relies on manual copying, manual interpretation, and manual checking.

Traditional TFL programming workflows suffer from several recurring problems. First, SAPs are typically semi-structured documents in which the same statistical requirement may be scattered across different sections—for example, analysis sets, population definitions, endpoints, visit windows, statistical methods, missing data handling, and output requirements. Programmers must read and translate these manually into executable programming specifications. Second, there is substantial redundancy among the TOC, Shells, Tracker, program parameters, and QC checklists; maintaining them separately creates version inconsistencies. Third, after a SAP amendment, newly added, deleted, or modified output requirements must be manually tracked—and the downstream impact on Shells, programs, QC, and final outputs is difficult to assess quickly. Fourth, when multiple project teams, regions, or vendors collaborate without a unified metadata model and standard assets, delivery style and tracking quality are easily affected by individual experience.

With advances in large language models, document parsing, rule engines, and code-assisted generation technology, statistical programming teams can reconsider how traditional workflows are organized. The framework proposed in this paper does not attempt to have AI autonomously make statistical decisions or replace programmers; rather, it confines AI to repetitive tasks such as structured document parsing, metadata extraction, template matching, parameter population, and consistency checking. Statisticians, medical experts, and programmers continue to be responsible for protocol judgment, result interpretation, program validation, and final sign-off.

This paper aims to present an AI-assisted workflow framework from SAP to Shell and TFL delivery, focusing on three key questions:

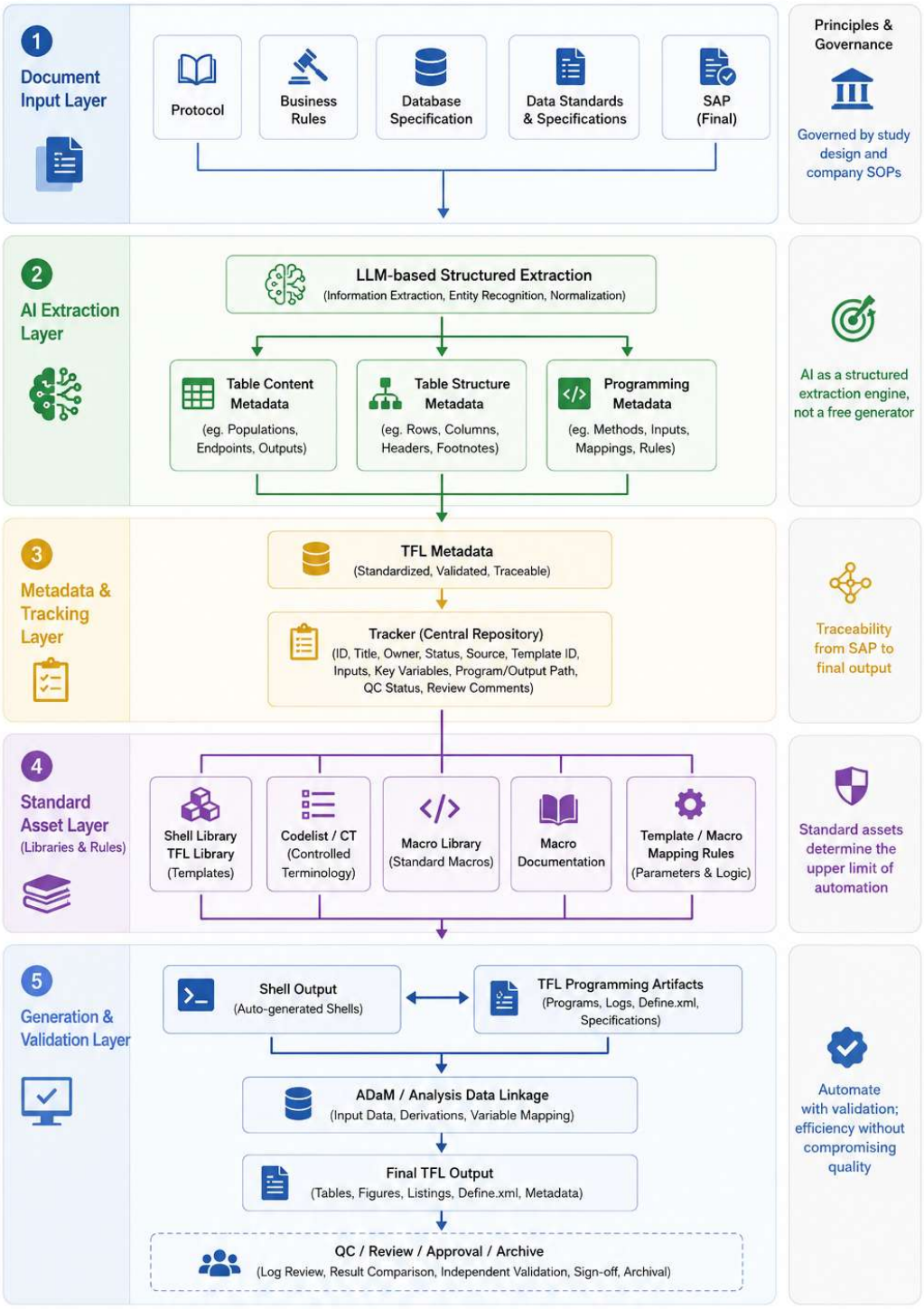
- How to transform natural-language analysis requirements in the SAP into structured metadata;
- How to connect Shells and TFL programming artifacts through the Tracker, Library, codelists, and macro library;
- How to preserve traceable, verifiable, and auditable quality control mechanisms while leveraging AI-assisted generation.

2. END-TO-END WORKFLOW FRAMEWORK

Figure 1 illustrates the AI-assisted TFL delivery process proposed in this paper. The overall flow can be summarized as:

Upstream Documents / Finalized SAP → LLM Structured Extraction → TFL Metadata → Tracker → Shell Library → Shell Output → Programming Artifact Mapping → TFL Programming Artifacts → ADaM Linkage → Final TFL Output

Figure 1. AI-Assisted End-to-End Automation Framework from SAP to TFL Output



Abbreviations: SAP = Statistical Analysis Plan; TFL = Tables, Figures and Listings; LLM = Large Language Model; ADaM = Analysis Data Model; QC = Quality Control; CT = Controlled Terminology.

Figure 1. AI-Based SAP-to-TFL Automated Programming Flow Chart

The framework consists of five layers.

Layer 1 is the document input layer, comprising the study protocol, business rules, database construction specifications, data specifications, and finalized SAP. These documents provide information on study design, treatment groups, endpoints, analysis sets, visit windows, data sources, and output scope. The finalized SAP and Spec are the primary basis for metadata extraction.

Layer 2 is the AI extraction layer. The LLM is responsible for converting the unstructured or semi-structured text in the SAP into candidate metadata—for example, analysis populations, endpoints, statistical methods, output types, titles, footnotes, input datasets, and variable mappings. The role of AI at this layer is closer to a "structured extraction engine" than a free generator.

Layer 3 is the metadata and tracking layer. Confirmed metadata enters the Tracker. The Tracker not only records TFL numbers, titles, owners, and status, but also stores source sections, template IDs, input data, key variables, program paths, output paths, QC status, and review comments. The Tracker is therefore the central object connecting SAP, Shells, programs, data, and final outputs.

Layer 4 is the standard asset layer, comprising the Shell Library, TFL Library, controlled terminology, standard macro library, macro documentation, and template mapping rules. This layer determines the upper bound of automation. If an organization has already accumulated stable templates and macro libraries, AI can take on more work in selecting templates, populating parameters, and generating frameworks; if standard assets are lacking, AI-generated content tends to become one-off scripts that are difficult to reuse and verify.

Layer 5 is the generation and validation layer. The system automatically generates Shells based on the Tracker and Library, and generates TFL programming artifacts based on program assets and mapping rules. Generated output enters human review, log checks, result comparison, independent validation, and sign-off—ultimately producing deliverable TFL output.

The foundational principles of this framework are: AI-assisted but without abandoning governance; automated generation but without bypassing validation; improving efficiency but without undermining statistical and programming accountability.

3. SAP STRUCTURED METADATA EXTRACTION

The SAP is the primary basis for statistical analysis and TFL development, but it is not a natively machine-readable structured data source. Across different projects, statisticians, and templates, the way output requirements are described in an SAP can vary considerably. For example, information about a single table may be scattered across the analysis population, endpoint definition, statistical methods, handling of missing data, multiplicity adjustment, and TFL sections.

Accordingly, this framework first uses an LLM to extract structured metadata from the SAP. To reduce confusion, it is recommended to categorize extraction results into three types: table content metadata, table structure metadata, and table programming metadata.

3.1 Table Content Metadata

Table content metadata describes the statistical content displayed within a TFL, typically including:

- Analysis population
- Endpoint or analysis variable
- Treatment groups
- Visit or analysis window
- Statistics
- Denominator rules
- Sorting rules
- Missing data handling
- Display labels
- Stratification or grouping variables

For example, for an adverse event summary table, content metadata might include the Safety Analysis Set, system organ class, preferred term, treatment groups, subject counts, percentages, severity, relatedness, and sorting rules. For a laboratory test listing, content metadata might include subject ID, visit, test parameter, result, unit, normal range, abnormality flag, and analysis date.

3.2 Table Structure Metadata

Table structure metadata describes the external structure of an individual output, typically including:

- Output type
- TFL number
- Title
- Population description
- Footnotes
- Programming note
- Page layout
- Column structure
- Shell template ID
- Source SAP section
- Whether special formatting is required

This type of metadata is primarily used to generate the TOC, Tracker, and Shells. Unlike manually maintaining a static TOC, this framework treats the TOC and Tracker as structured objects, enabling them to further drive template matching, program generation, and change tracking.

3.3 Table Programming Metadata

Table programming metadata is oriented toward downstream program implementation, typically including:

- Input datasets
- Filter conditions
- Treatment group variable
- Analysis population variable
- Analysis variables
- Variable labels
- Variable display order
- Codelists corresponding to variables
- Macro candidates
- Other template parameters
- Special handling rules
- QC checkpoints

For example, a demographics summary table may need to specify the ADSL dataset, the FASFL flag, treatment group variable, age, sex, race, statistics, and display format. By structuring this information, the system can further generate program frameworks, parameter lists, and validation checklists.

3.4 The "AI Extraction + Rule Validation + Human Confirmation" Approach

Relying solely on an LLM to directly produce final results is not suitable for clinical statistical programming. This paper recommends a combined approach of "AI extraction + rule validation + human confirmation."

AI is responsible for extracting candidate metadata from the SAP; the rule engine checks required fields, field types, terminology consistency, template compatibility, and variable availability; programmers or statisticians confirm the candidate results. Confirmed metadata should be saved in a versionable medium such as a database, Excel, or JSON, and should retain source sections and manual modification records, rather than remaining only in chat logs or temporary documents.

The core of this mechanism is to constrain the LLM within a well-defined context and data structure. The LLM can help understand and transcribe the SAP, but cannot replace statistical judgment and project sign-off.

4. TRACKER AND SHELL GENERATION

The Tracker is the tracking hub of the entire workflow. In traditional projects, the Tracker is often merely a project management spreadsheet recording output numbers, titles, owners, and status. In this framework, the Tracker is elevated to a structured workflow object that connects SAP interpretation, Shell generation, program development, QC, and final delivery.

4.1 Core Tracker Fields

A Tracker suitable for automation should include at minimum the following fields:

Field Category	Example Fields
Basic Information	TFL number, title, output type, version
Source Information	SAP section, source sentence, Shell source, Library source

Analysis Information	Analysis population, endpoint, visit, treatment group, statistical method
Data Information	Input dataset, key variables, filter conditions
Template Information	Shell template ID, program template ID, macro candidates
Delivery Information	Program path, output path, output filename, delivery batch
Management Information	Owner, development status, QC status, review comments, sign-off status

With this design, the Tracker is not merely a status table but the metadata source for subsequent Shells and TFL programming artifacts.

4.2 Shell Library and Template Matching

The Shell Library manages the organization's standard Shell templates, including title patterns, footnotes, column structures, statistic displays, treatment group layouts, page layouts, and output formats. Managing Shells through a Library reduces the effort of re-creating shell templates for each project and improves consistency across studies.

Template matching can be based on:

- Output type
- Statistical method
- Visit structure
- Treatment group structure
- Analysis variables
- Output format
- Company or project standards

For example, the system can match a TEAE summary template based on "Safety Analysis Set + adverse events + treatment group summary + counts and percentages"; or match a continuous variable visit summary template based on "FAS + continuous efficacy endpoint + visit + descriptive statistics."

4.3 Shell Output Generation Process

Shell generation typically includes the following steps:

- Read confirmed TFL metadata;
- Match a Shell template based on output type and analysis characteristics;
- Retrieve variables, labels, and values from study database metadata or ADaM specification;
- Populate titles, footnotes, column structure, population, variables, and statistics;
- Apply company standard formatting and project rules;
- Generate a near-final Shell;
- Enter human review and necessary adjustments.

In this process, the role of AI is not to directly "draw a table" but to help translate natural-language requirements in the SAP into parameters that templates can understand. What truly ensures consistency is the collaboration among the Tracker, Library, metadata, and rule engine.

5. CODELISTS, MACRO LIBRARY, AND PROGRAMMING ARTIFACT MANAGEMENT

After Shell generation, the next step is to produce TFL programming artifacts. This paper uses the term "programming artifacts" because actual deliverables include not only programs themselves but also program configurations, parameter lists, input/output descriptions, validation checklists, and review records.

5.1 Programming Asset Layer

The programming asset layer primarily consists of codelists, standard macros, macro documentation, TFL templates, and mapping rules.

Codelists manage categorical variables, coded values, sort values, and display labels. For example, sex, race, visit, adverse event severity, and laboratory abnormality grades all require uniform display and sorting rules.

The standard macro library encapsulates common statistics, transposition, formatting, output, and QC logic. The stability of the macro library directly determines the degree of reusability in automation. Without a stable macro library, AI-generated programs easily become one-off scripts that are difficult to validate and maintain; with a well-designed macro library, AI can take on more of the work of selecting macros, populating parameters, generating call frameworks, and supplementing validation checklists.

Macro documentation must clearly describe macro parameters, input datasets, output datasets, constraints, applicable output types, and example usage. TFL template and macro matching rules automatically map a given type of output to the appropriate macro and program framework.

5.2 AI-Assisted Generation of Programming Artifacts

Based on the Tracker and table programming metadata, AI can assist with the following:

- Selecting appropriate program templates or standard macros;
- Populating macro parameters;
- Generating input dataset and variable mappings;
- Generating grouping, sorting, filtering, and output configurations;
- Generating the program framework;
- Generating QC checklists;
- Generating implementation notes readable by reviewers.

For high-frequency, rule-stable outputs—such as tables in the 14.1 and 14.3 sections of a Shell—the system can prioritize calling validated standard macro libraries. For complex efficacy analyses, special statistical methods, or highly customized graphical outputs, programmers should perform further development and validation based on the auto-generated framework.

5.3 Linkage with the ADaM Data Processing Pipeline

TFL automation cannot be separated from standardized datasets. ADaM datasets provide a structured foundation for statistical analysis, and TFL metadata must also align with ADaM specifications. For each TFL, the system should verify that required datasets, variables, analysis flags, parameter codes, visit variables, and grouping variables are all supported by ADaM.

When the Tracker, Shells, and program parameters all reference the same ADaM metadata, inconsistencies between SAP interpretation, Shell design, and program implementation can be reduced. For example, if an endpoint variable changes in the ADaM specification, the system can flag affected Shells, programs, and outputs to assist the project team in change impact assessment.

6. EXAMPLE: FROM SAP TO AE SUMMARY AUTOMATION CHAIN

To illustrate how the framework works in practice, this section uses a common adverse event summary table as an example to demonstrate the transformation from SAP text to metadata, Tracker, Shell, and programming artifacts. This example is intended solely to illustrate the process; in real projects, fields and rules should be adjusted according to company standards, project SAP, and ADaM specifications.

6.1 Original SAP Description

Suppose the SAP contains the following description:

```
Treatment-emergent adverse events (TEAEs) will be summarized by system organ
class and preferred term for the Safety Analysis Set. The number and
percentage of subjects with at least one TEAE will be presented by treatment
group. Percentages will be calculated using the number of subjects in the
Safety Analysis Set as the denominator.
```

This type of description typically contains the key information needed for table generation, but in the traditional workflow programmers must manually translate it into TOC entries, Shells, program parameters, and QC rules.

6.2 LLM-Extracted Structured Metadata

The LLM first converts the SAP description into candidate structured metadata. Example:

```
{
  "output_type": "Table",
  "analysis_topic": "Treatment-emergent adverse events",
  "population": "Safety Analysis Set",
  "population_flag": "SAFFL",
  "input_dataset": "ADAE",
  "grouping_variables": ["TRT01A"],
  "row_variables": ["AEBODSYS", "AEDECOD"],
  "statistics": ["n(%)", "Events"],
  "denominator": "Number of subjects in Safety Analysis Set",
  "sort_rule": "By system organ class and preferred term",
  "template_candidate": "AE_SUMMARY_BY_SOC_PT",
  "source_section": "SAP Safety Analysis / Adverse Events",
  "source_text": "Treatment-emergent adverse events ... denominator."
}
```

This result does not directly serve as the final deliverable; instead it enters the rule validation and human confirmation step.

6.3 Rule Validation

The rule engine can check the following:

Check Item	Example Rule
Required fields	output_type, population, input_dataset, row_variables, statistics must not be empty
Dataset check	input_dataset = ADAE must exist in ADaM specification
Population check	SAFFL must exist in ADSL or the relevant analysis dataset
Variable check	AEBODSYS, AEDECOD, TRT01A must exist in ADAE or be obtainable via mapping
Denominator check	Percentages must use number of subjects in Safety Analysis Set as denominator

Template check	AE_SUMMARY_BY_SOC_PT must be applicable to SOC/PT-level summary
Source check	Each key field must retain SAP source section or source sentence

If the rule validation reveals missing or inconsistent fields, the system should return the issues to the programmer or statistician for confirmation rather than guessing automatically.

6.4 Tracker Record

Confirmed metadata enters the Tracker. Example:

Field	Example Value
TFL No.	Table 14.3.1.2
Title	Summary of Treatment-Emergent Adverse Events by System Organ Class and Preferred Term
Output Type	Table
Population	Safety Analysis Set
Source Section	SAP Safety Analysis / Adverse Events
Input Dataset	ADAE
Key Variables	TRT01A, AEBODSYS, AEDECOD, SAFFL
Template ID	AE_SUMMARY_BY_SOC_PT
Macro Candidate	%ae_socpt_summary
Program Path	/programs/tfl/t_14_3_1_2.sas
Output Path	/outputs/tfl/t_14_3_1_2.rtf
Development Status	Draft
QC Status	Pending
Reviewer Comment	To be confirmed by study statistician

Under this design, the Tracker simultaneously serves as project management, program generation input, QC tracking, and audit index.

6.5 Shell Output Example

Based on the Tracker and Shell Library, the system can generate the following Shell structure. The title, population, column structure, statistics, footnotes, and coding notes all originate from the combination of Tracker, SAP metadata, Library, and codelist—not created manually from scratch.

Shell Structure Example (Table 14.3.1.2):

System Organ Class / Preferred Term	Placebo n=XXX	Treatment A n=XXX	Treatment B n=XXX
Any TEAE	n (%)	n (%)	n (%)
Cardiac Disorders			
Palpitations	n (%)	n (%)	n (%)
Gastrointestinal Disorders			
Nausea	n (%)	n (%)	n (%)

6.6 Programming Artifact Example

Based on the same Tracker record, the system can further generate programming artifacts. This paper does not emphasize any particular programming language but rather the structured content of programming artifacts. Example:

```
Programming Artifact: AE_SUMMARY_BY_SOC_PT

Input:
  - Dataset: ADAE
  - Population flag: SAFFL = "Y"
  - Treatment variable: TRT01A
  - Row variables: AEBODSYS, AEDECOD
  - Statistics: subject count and percentage
  - Denominator: number of subjects in Safety Analysis Set by treatment
group

Template / Macro:
  - Template ID: AE_SUMMARY_BY_SOC_PT
  - Macro candidate: %ae_socpt_summary

Output:
  - TFL No.: Table 14.3.1.2
  - Output file: t_14_3_1_2.rtf
  - Shell: shell_t_14_3_1_2.docx

Validation Checklist:
  - Confirm ADAE contains one record per adverse event.
  - Confirm TEAE derivation flag is correctly applied.
  - Confirm SAFFL denominator matches ADSL.
  - Confirm SOC/PT coding and sorting rules.
  - Confirm treatment group order and display labels.
  - Compare total subjects with safety population summary.
  - Review log and output formatting.
```

This example embodies the core idea of the framework: the same set of structured metadata can be reused multiple times, respectively supporting the Tracker, Shell, program parameters, QC checklist, and final delivery—reducing repeated data entry and improving consistency among the SAP, Shell, program, and output.

7. QUALITY CONTROL, VALIDATION, AND GOVERNANCE

The key in an AI-assisted process is not to have AI directly replace programmers, but to incorporate AI-generated artifacts into the existing quality system. All SAP drafts, metadata, Shells, programming artifacts, and QC checklists generated or modified by AI should have version records, human review records, and traceable sources.

7.1 Source Traceability

Each key metadata field should be traceable to a clear source, including:

- SAP section
- SAP source sentence
- Shell Library
- ADaM specification
- Codelist
- Macro documentation
- Manual modification records

For example, the analysis population field of a TFL should be traceable to the analysis set definition in the SAP; a title or footnote should be traceable to the SAP or Shell Library; a variable mapping should be traceable to the ADaM specification. Only in this way can review, QC, and audit be properly supported.

7.2 Change Comparison

SAP amendments are a common occurrence in clinical projects. Without structured metadata, assessing change impacts often requires manually comparing page by page. With structured metadata, the system can automatically identify:

- New TFLs
- Deleted TFLs
- Title changes
- Population changes
- Endpoint changes
- Statistical method changes
- Variable changes
- Shell impact scope
- Program impact scope
- QC impact scope

Change comparison results can be synchronously updated in the Tracker and can alert programmers and statisticians to outputs requiring attention.

7.3 Validation Checks

It is recommended to set up multiple layers of validation checks in the process, including:

- Required field check
- Duplicate TFL check
- Template compatibility check
- Variable existence check
- Population flag check
- Codelist consistency check

- Macro parameter check
- Output naming check
- Log check
- Result comparison
- Independent programming review

These checks cannot fully replace manual QC, but they can significantly reduce low-level errors and repetitive verification work.

7.4 Human Accountability Boundaries

AI-generated content cannot bypass human review and sign-off. Statistical judgment, medical interpretation, exception handling, complex rule confirmation, and final delivery approval should still be performed by qualified project members. AI can provide candidate results, difference prompts, and checklists, but final accountability must remain clear.

8. COMPARISON WITH TRADITIONAL WORKFLOWS: BENEFITS AND LIMITATIONS

Compared with traditional manual workflows, the primary benefits of this framework are reflected in three areas.

First, requirement parsing is faster. The LLM can quickly extract TFL lists and initial programming information from the SAP, reducing the time programmers spend manually compiling TOCs, Shells, and parameters. For SAP templates with relatively stable structures, extraction accuracy is easier to maintain.

Second, delivery consistency is higher. The Shell Library, codelists, and macro library distill project experience into standard assets, reducing style differences between programmers. Through unified templates and mapping rules, similar tables across different studies can maintain more consistent titles, footnotes, column structures, sorting, and formatting.

Third, tracking is clearer. The Tracker connects SAP, Shells, programs, data, and outputs, making project management, review, change assessment, and audit more convenient. Compared with dispersed maintenance across multiple Excel files, Word documents, and folders, a structured Tracker is better suited as the workflow hub.

At the same time, the framework has practical limitations. AI may still misinterpret complex statistical semantics, vague descriptions, and project-specific exception rules. If SAP document quality is poor, extraction results will be affected. If an organization lacks a unified Library, macro library, and metadata standard, automation can only achieve local efficiency gains and cannot scale into enterprise-level delivery capability. Furthermore, complex efficacy analyses, complex graphics, special models, and study-specific rules still require deep involvement from programmers and statisticians.

Therefore, it is recommended to start piloting with high-frequency, low-complexity, rule-stable TFL types—such as demographics tables, disposition tables, exposure summary tables, adverse event summary tables, and laboratory-related tables and listings. After the Library, metadata model, macro library, and validation processes have progressively matured, the framework can be extended to complex efficacy analyses and graphical outputs.

9. AGENTIC WORKFLOWS, HERMES ENGINEERING, AND WORKFLOW-ORIENTED AUTOMATION

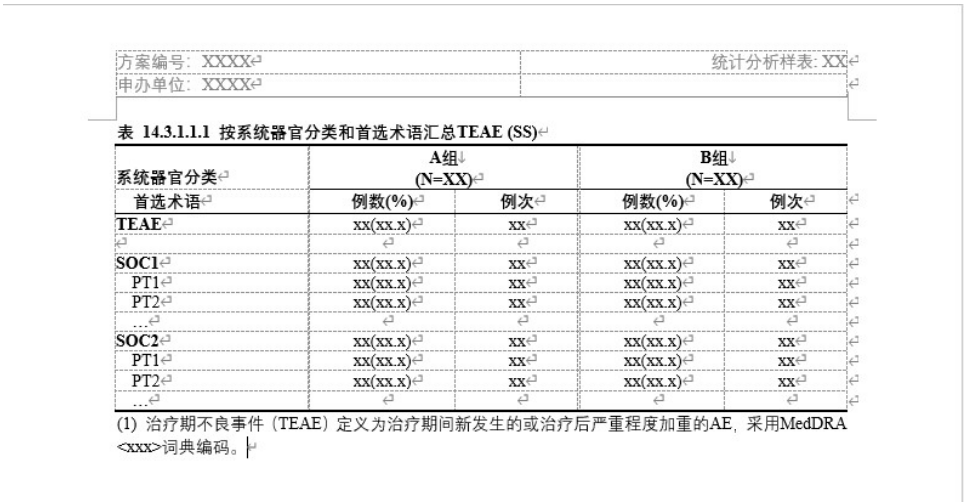

方案编号：XXXX　　统计分析样表: XX
申办单位：XXXX

表 14.3.1.1.1 按系统器官分类和首选术语汇总TEAE (SS)

系统器官分类 / 首选术语	A组 (N=XX) 例数(%)	A组 (N=XX) 例次	B组 (N=XX) 例数(%)	B组 (N=XX) 例次
TEAE	xx(xx.x)	xx	xx(xx.x)	xx
SOC1	xx(xx.x)	xx	xx(xx.x)	xx
PT1	xx(xx.x)	xx	xx(xx.x)	xx
PT2	xx(xx.x)	xx	xx(xx.x)	xx
…				
SOC2	xx(xx.x)	xx	xx(xx.x)	xx
PT1	xx(xx.x)	xx	xx(xx.x)	xx
PT2	xx(xx.x)	xx	xx(xx.x)	xx
…				

(1) 治疗期不良事件（TEAE）定义为治疗期间新发生的或治疗后严重程度加重的AE，采用MedDRA <xxx>词典编码。

Figure 2. Agentic Workflow and Hermes Engineering Concept Diagram

In recent years, agentic workflows and Hermes Engineering have attracted increasing attention in the AI field. These approaches typically emphasize enabling AI to autonomously complete code generation, task decomposition, debugging, and delivery with the support of skills, tools, and context memory. For software development scenarios, these paradigms hold considerable promise and provide a new imaginative space for statistical programming automation.

However, in regulated clinical statistical programming environments, a fully autonomous approach still faces practical challenges. TFL delivery is not just a code generation problem; it also involves statistical interpretation, medical judgment, data standards, version control, validation records, audit trails, and accountability. Even if AI can generate runnable code, this does not mean the code is suitable for direct entry into a production programming workflow.

The practice in this paper supports a more robust path: in the near term, AI is better suited as a component of workflow-oriented engineering automation rather than as a fully autonomous programming agent. That is, AI should first be embedded into existing standardized processes to help complete document parsing, metadata extraction, template matching, parameter population, consistency checking, and change tracking for repetitive tasks.

In the future, stronger agentic capabilities can be gradually introduced—but the prerequisite should be that the organization has already established a stable metadata model, verifiable Shell Library, controlled macro library, traceable records, and human approval mechanisms. Otherwise, agentic coding can easily turn automation into one-off generated results that are difficult to track, rather than auditable and reusable engineering assets.

10. CONCLUSION

This paper presents an AI-assisted workflow framework for delivering Shells and TFLs from the SAP. The framework uses an LLM to convert natural-language analysis requirements in the SAP into structured metadata, with the Tracker as the hub connecting the SAP, Shells, programming assets, ADaM data, and final outputs. Through the Shell Library, codelists, standard macro library, and rule engine, the system can support semi-automated generation of Shells and TFL programming artifacts.

The core argument of this paper is that the value of AI in clinical statistical programming should not be simplistically understood as "automatically writing programs" or "replacing programmers," but rather as an engineering-level enhancement of existing workflows. AI can reduce repetitive interpretation, repeated data entry, and repetitive checking, while improving consistency and traceability across study deliveries; however, statistical judgment, program validation, medical interpretation, and final sign-off must remain within a controlled human-machine collaboration process.

In future practice, organizations can start with rule-stable, highly reusable TFL types and gradually accumulate structured metadata, standard templates, macro libraries, and validation rules. As these engineering assets continue to mature, AI-assisted statistical programming has the potential to evolve from localized efficiency gains into an auditable, reusable, and scalable end-to-end delivery model.

REFERENCES

[1] CDISC. Study Data Tabulation Model (SDTM). Clinical Data Interchange Standards Consortium.

[2] CDISC. Study Data Tabulation Model Implementation Guide (SDTMIG). Clinical Data Interchange Standards Consortium.

[3] CDISC. Analysis Data Model (ADaM). Clinical Data Interchange Standards Consortium.

[4] CDISC. Analysis Data Model Implementation Guide (ADaMIG). Clinical Data Interchange Standards Consortium.

[5] International Council for Harmonisation. ICH E9: Statistical Principles for Clinical Trials.

[6] International Council for Harmonisation. ICH E6(R3): Good Clinical Practice.

[7] U.S. Food and Drug Administration. Considerations for the Use of Artificial Intelligence to Support Regulatory Decision-Making for Drug and Biological Products.

[8] U.S. Food and Drug Administration. Using Artificial Intelligence and Machine Learning in the Development of Drug and Biological Products.

[9] PharmaSUG Proceedings. Papers on clinical trial programming, metadata-driven reporting, and TFL automation.

[10] Company internal standards, Shell Library, macro documentation, codelist, and TFL delivery procedures.

ACKNOWLEDGMENTS

The authors would like to thank the statistics, data management, medical, and statistical programming teams for their continued collaboration in the SAP, ADaM, Shell, and TFL delivery processes. The workflow framework described in this paper can be further adapted to company standards, project complexity, system environment, and regulatory submission requirements.

CONTACT INFORMATION

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

Author Name: Xiaoyu Ding
Company: Keymed Corporation
E-mail: xiaoyuding@keymedbio.com