Paper SM05

AI-Optimized AESI Flagging: Streamlined SAS Hash Table Logic with Dynamic CALL EXECUTE and Metadata-Driven Specifications

Hang Zhong, Bristol Myers Squibb, Seattle, WA, USA

Rashad Rasul, Bristol Myers Squibb, Boudry, Switzerland

Jiaqi Hu, Bristol Myers Squibb, Princeton, NJ, USA

ABSTRACT

Traditionally, Adverse Events of Special Interest (AESI) workflows rely on standalone datasets and PROC SQL merges, leading to fragmented logic, performance bottlenecks, and high maintenance costs. This paper presents a metadata-driven approach that eliminates intermediate AESI datasets using SAS hash table logic and CALL EXECUTE for dynamic macro execution. A centralized AESI strategy specification replaces traditional lookup datasets, defining MedDRA strategies—Preferred Terms (PTs), High Level Group Terms (HLGTs), Standardized MedDRA Queries (SMQs), and customized SMQs (cSMQs)—for modular, reviewer-friendly traceability compliant with CDISC Analysis Data Model (ADaM) standards. Pilot implementation achieved an 85% reduction in AESI flagging effort (13 hours to 2 hours per study) with 100% concordance with traditional SQL methods, redirecting programmer capacity to higher-value safety analyses. This approach transforms AESI programming into scalable, compliant, and metadata-driven automation that advances industry best practices for pharmacovigilance.

INTRODUCTION

The Industry Challenge

Traditionally, AESI flagging creates significant maintenance burden for clinical programming departments. When protocol amendments occur or MedDRA versions are updated, extensive refactoring is required (Wu, 2023). Many organizations struggle with AESI definitions spanning multiple legacy studies, often with limited documentation access and unclear specifications. Common industry approaches vary widely: some use standalone MedDRA lookup datasets merged via PROC SQL (Liu & Cai, 2023), others maintain hardcoded macro logic requiring version updates. Still others mix SMQs, FMQs, and CMQs without standardization. This fragmentation redirects programmer effort from high-value safety analyses to repetitive, error-prone maintenance coding—a persistent pain point discussed at PHUSE conferences.

The Traditional Approach: Limitations and Costs

Most organizations create intermediate datasets (e.g., “EOSI”, “AESI_LOOKUP”) by subsetting MedDRA dictionary tables (hierarchy, SMQ subset, content), then merging with ADAE via multiple PROC SQL joins. The typical workflow is:

1. Clinicians provide AESI specifications (often in Word/Excel without structured definitions)
2. Programmers manually query separate MedDRA tables
3. Results are stacked by AESI category and joined to ADAE for each strategy
4. With every MedDRA version change or specification update, this entire process repeats

Key limitations:

- **Maintenance burden:** Every MedDRA upgrade requires manual refactoring across multiple programs
- **Loose traceability:** Specifications and executable code remain poorly coupled
- **Algorithmic inefficiency:** Multiple SQL merges create O(n×m) complexity per study

Objectives and PHUSE Alignment

To address these challenges, we developed a metadata-driven engine aligned with PHUSE Good Programming Practices (GPP) principles and EMA/FDA guidance on AI governance. Our three objectives are:

1. **Architectural:** Eliminate intermediate lookup datasets entirely through memory-resident hash tables

2. **Performance:** Reduce algorithmic complexity from O(n×m) to O(n)
3. **Regulatory:** Ensure 100% specification-to-Define.xml concordance for complete traceability

TIERED METADATA ARCHITECTURE

System Design Overview

We implemented a three-tier architecture separating specifications ("What") from implementation logic ("How"). This ensures 100% concordance between submission documentation and executed code—a regulatory requirement increasingly scrutinized during inspections.

Tier 1 – Specification (Single Source of Truth): Machine-executable Excel/CSV containing AESI definitions, MedDRA strategies, and clinical validation criteria.

Tier 2 – Processing Engine: SAS macros using hash tables and CALL EXECUTE to consume specifications and generate flagged datasets.

Tier 3 – Regulatory Documentation: Define.xml populated automatically from Tier 1, eliminating manual reconciliation errors.

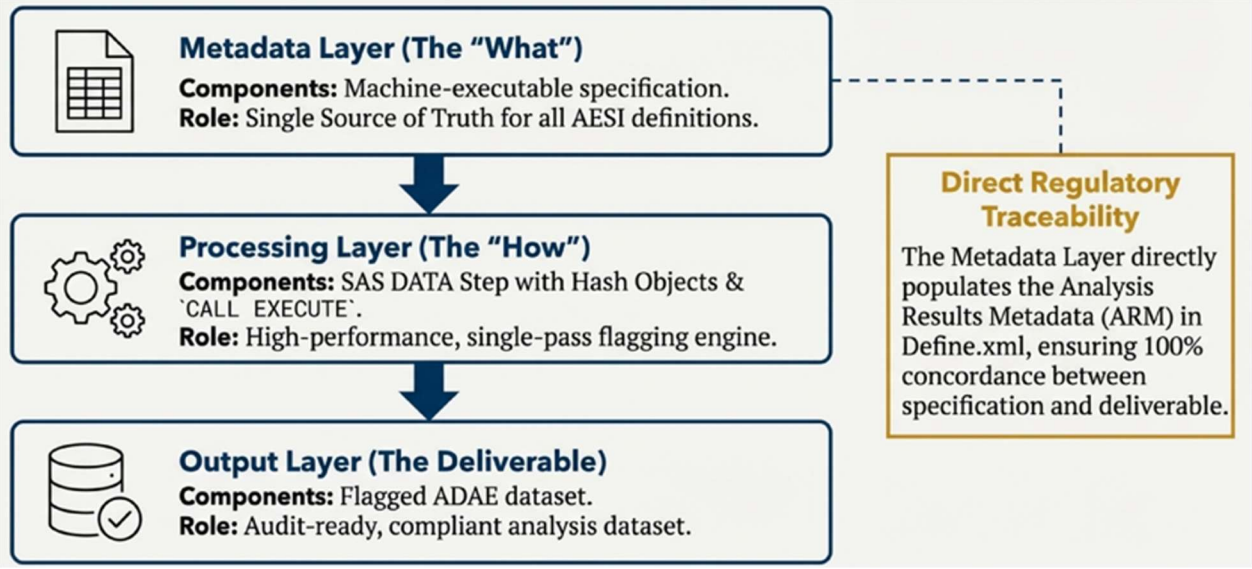

Figure 1: The Three-Tier Metadata-Driven Architecture. The specification acts as the single source of truth, driving the SAS processing engine and directly populating Define.xml metadata for full regulatory traceability.

Specification Design: Clinical Accessibility and Version Control

We chose Excel/CSV as the specification format because medical writers and clinicians can maintain definitions without requiring SAS knowledge. This approach also enables version control via Git, providing audit trails for regulatory submissions. When clinicians update a specification, programmers re-run the macro with zero code changes. Validation and Define.xml metadata auto-update from the same source.

Table 1: Specification-to-Define.xml Mapping

Specification Column	Define.xml Element	Purpose
AESI_VAR	Variable Name	e.g., CQ01NAM
AESI_DESC	Variable Label	Human-readable category description
PROG_CODE + CONDITION	Derivation Logic	Exact MedDRA selection criteria
MEDDRA_VER	External Dictionary Version	Version audit trail

This direct mapping ensures regulatory reviewers can trace AESI definitions from submission metadata directly to the flagged clinical data—a key requirement in recent FDA and EMA inspection observations (EMA/FDA, 2025). For detailed mapping examples, see Supplementary Appendix E.4.

Hash Table Architecture: Single-Pass Efficiency

Unlike earlier macro-based approaches (Liu & Cai, 2023) that require multiple data passes for each AESI, our hash-table method performs all lookups in a single pass. By deriving logic directly from the centralized specification, we eliminate maintaining separate lookup datasets and programs.

Processing Flow:

1. **Read Metadata:** Import the AESI specification (Excel) into SAS as a control table
2. **Initialize Hash Tables:** At _n_=1, load all MedDRA lookup criteria into memory-resident hash objects
3. **Single-Pass Flagging:** For each AE record, perform O(1) hash lookups against all strategies simultaneously
4. **Dynamic Dispatch:** Use CALL EXECUTE to invoke specialized macros for complex logic

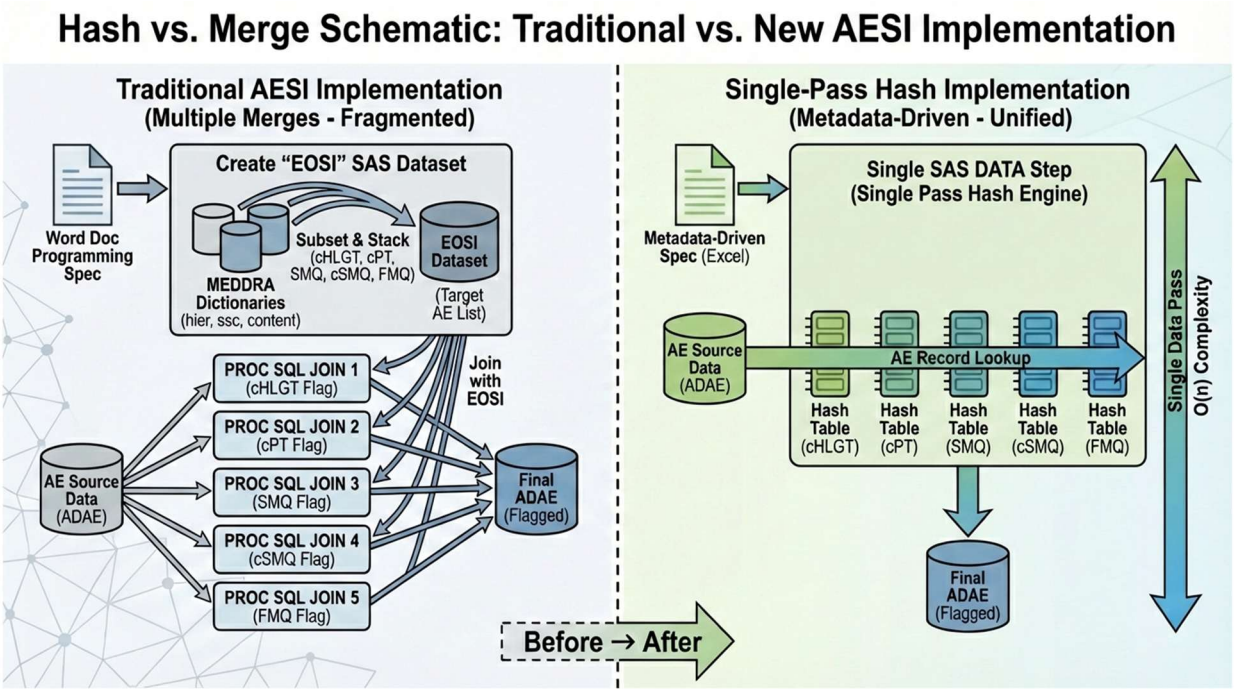

Figure 2: Hash vs. Merge Schematic. Left: Traditional approach creates an intermediate "EOSI" dataset by subsetting and stacking from MedDRA dictionaries (hier, ssc, content), then merges with ADAE via multiple PROC SQL joins. Right: Hash table approach loads criteria directly from MedDRA dictionaries into memory, performing all lookups in a single data pass with O(n) complexity.

Performance Implications:

- **Traditional approach:** 5 AESI strategies = 5 data passes = O(5n)
- **Hash table approach:** 5 AESI strategies = 1 data pass = O(n)

For the AESI "Hypertension" (SMQ_CODE 20000147), the lookup table contains approximately 60-80 Preferred Terms. With hash tables, this entire set is checked for each record in constant time, versus SQL requiring a full merge with sorting for each strategy.

```
/* Concept: One Pass, Multiple Lookups */
data adae_flagged;
    length SMQ02NAM $200;
    call missing(SMQ02NAM);
```

```
    if _n_ = 1 then do;
        declare hash h_smq02(dataset: "work.lkup_smq02");
        h_smq02.definekey('PT_NAME');
        h_smq02.definedata('SMQ02NAM');
        h_smq02.definedone();
    end;
    set adam.adae;
    if h_smq02.find(key: aedecod) = 0 then; /* SMQ02NAM populated */
run;
```

For detailed code patterns and the complete variable naming convention (FL vs NAM suffix), see Appendix A of the Supplementary Material.

AI-ASSISTED DEVELOPMENT AND GxP GOVERNANCE

The hash table architecture is fully implementable using manual SAS programming. However, we accelerated development and ensured regulatory compliance by providing AI tools with a curated "skills file"—a machine-readable document containing explicit rules, forbidden patterns, and code examples.

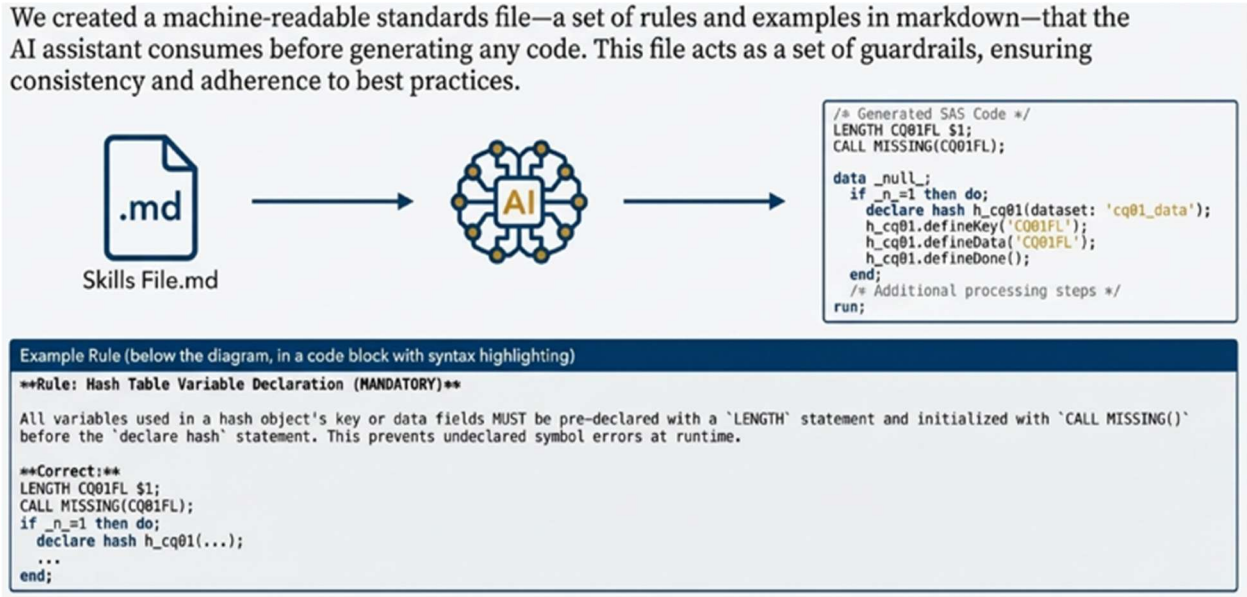

Figure 3: The AI "Skills File" Process. The machine-readable markdown file acts as a set of guardrails, constraining the AI model output to adhere to specific SAS coding patterns and corporate standards.

Governance Framework:

1. **AI Assistance:** AI generates initial hash table code, SAS macro definitions, and validation logic, guided by the skills file
2. **Programmer Review:** All generated code undergoes independent review for logic correctness and regulatory compliance
3. **Standard QC:** All code—AI-generated and manual—follows corporate SOPs for validation, including PROC COMPARE for concordance with traditional SQL methods
4. **Skills File Evolution:** When programmers discover errors, they document patterns and update the Quick Fix Index and rule sections

Risk Mitigation: Without structured guidance, AI-generated code had errors in ~71% of initial trials. These errors stemmed from subtle semantic issues: checking NAM variables for 'Y' instead of using FL variables, missing CALL MISSING statements, using reserved macro variable names. The skills file approach constrains outputs to established patterns, aligning with PHUSE GPP principles on human oversight and standardized processes.

This approach satisfies EMA/FDA guidance on human-centric AI use in clinical trial programming: programming remains the human responsibility; AI augments generation, not decision-making.

RESULTS

Primary Benefit: Architectural Simplification and Maintenance Reduction

The principal benefit is **eliminating the intermediate AESI lookup dataset entirely**. This architectural change delivers measurable advantages:

Table 2: Programming Hours by Activity (Pilot Study Data, 4-8 AESI Strategies)

Activity	Traditional (hours)	Metadata-Driven (hours)	Reduction (hours)
Specification Creation	3.0	0.5	2.5
AESI Dataset Programming	4.0	0.0	4.0
QC/Validation	4.0	1.0	3.0
Documentation	2.0	0.5	1.5
Total Per Study	**13.0**	**2.0**	**11.0**

This 85% reduction in per-study effort translates to:

- **Programmer Time Savings:** 11 hours redirected to higher-value safety analysis programming per study
- **Reduced Maintenance Burden:** No synchronization required between lookup datasets and ADAE flagging code; changes require only specification updates
- **Simplified Validation:** Single source of truth eliminates specification-code reconciliation errors

Cross-functional benefits include improved cycle time for regulatory amendments (clinical teams can modify AESI definitions independently) and faster submission timelines for protocol updates.

Computational Performance

Hash table lookups operate in O(1) constant time, reducing end-to-end complexity from O(n×m) to O(n). For typical clinical datasets (100,000+ adverse events), absolute runtime differences are modest (seconds saved per study). However, architectural simplification and eliminated maintenance provide far greater value. Performance gains become more pronounced for large real-world evidence datasets.

Validation and Regulatory Compliance

We verified concordance using PROC COMPARE: AESI flags produced by the hash-based approach matched 100% with those from conventional PROC SQL methods on four test studies (100,000+ records per study). This cross-validation confirmed that the new method maintains correctness while improving performance and maintainability.

Regulatory Alignment: The specification-driven Define.xml concordance ensures full traceability, a key requirement in recent FDA and EMA inspection observations. This addresses documentation gaps commonly cited in Establishment Inspections.

IMPLEMENTATION CONSIDERATIONS AND SCALABILITY

Prerequisites: Implementation requires MedDRA dictionary datasets (hierarchy and SMQ content tables) in accessible SAS locations. First-time setup for the macro framework and specification template requires 8–16 hours. ROI is realized from the second study, with full cost recovery typically within 2–3 studies depending on number of AESIs and data volume.

Scope and Limitations: Results are based on a single-organization pilot. Results may vary depending on existing infrastructure, number of AESI strategies, and study complexity. For distributed development teams using AI assistance, the skills file format continues to evolve. Emerging open standards offer improved modularity and interoperability.

Applicability: The metadata-driven architecture is generalizable to clinical programming beyond AESI flagging (e.g., SMQ-based derived variables, custom medical history flags).

CONCLUSION

This metadata-driven approach addresses documented industry challenges with:

1. **Architectural simplification:** SAS hash tables replace intermediate datasets, reducing logical fragmentation
2. **Performance improvement:** Complexity reduced from O(n×m) to O(n)
3. **Effort reduction:** 85% decrease in per-study programming hours (13 to 2 hours)
4. **Regulatory traceability:** 100% specification-to-Define.xml concordance enabling end-to-end auditability

The metadata-driven architecture and elimination of redundant processing are generalizable to clinical programming beyond AESI flagging. They align with PHUSE GPP principles for standardization, risk-based validation, and transparent documentation. When combined with structured AI governance (skills files), this approach shows how organizations can safely use automation for accelerated development while maintaining full regulatory compliance and human accountability.

REFERENCES

1. CDISC. (2023). ADaM Implementation Guide v1.3. Available at https://www.cdisc.org/
2. PhUSE. (2023). Good Programming Practices (GPP) Working Group. Available at https://www.phuse.global/
3. SAS Institute. (2023). SAS 9.4 Language Reference: Concepts. Available at https://documentation.sas.com/
4. Whitlock, I. (2016). “Hash Objects from the Ground Up.” SAS Global Forum.
5. Dorfman, P. (2015). “Data Management Solutions Using Hash Objects.” SUGI 30.
6. MedDRA MSSO. (2025). Introductory Guide to MedDRA Version 28.0.
7. FDA. (2023). Guidance for Industry: Safety Reporting Requirements for INDs. Available at https://www.fda.gov/
8. Liu, Y. and Cai, J. (2023). “Let the Data Drive it and Set your Hands Free - A Macro to Create Indicators for Special Interest AEs.” PharmaSUG 2023, Paper QT-175.
9. Wu, C. (2023). “Why FDA Medical Queries (FMQs) for Adverse Events of Special Interest?” PharmaSUG 2023, Paper DS-036.
10. EMA/FDA. (2026). European Medicines Agency and U.S. Food and Drug Administration. **Guiding Principles of Good AI Practice in Drug Development**. January 2026. Available at: https://www.fda.gov/media/189581/download

ACKNOWLEDGMENTS

We thank our colleagues in Global Biometrics & Data Science (GBDS) for their support during the development of this work. We particularly acknowledge valuable feedback from Xiaohan Zou, Zhouzhou (Joe) Xi, and Lei Zhang.

CONTACT INFORMATION

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

Author Name: Hang Zhong

Company: Bristol Myers Squibb

Address: Seattle, WA, USA

Email: hang.zhong@bms.com

AI-Optimized AESI Flagging: Streamlined SAS Hash Table Logic with Dynamic CALL EXECUTE and Metadata-Driven Specifications

Hang Zhong, Bristol Myers Squibb, Seattle, WA, USA

Rashad Rasul, Bristol Myers Squibb, Boudry, Switzerland

Jiaqi Hu, Bristol Myers Squibb, Princeton, NJ, USA

INTRODUCTION

This supplementary material provides detailed SAS code patterns, algorithmic analysis, and implementation references for readers seeking technical depth. The main paper introduces the metadata-driven AESI architecture; this document supports implementation. Each appendix is self-contained.

LIST OF ABBREVIATIONS

Abbreviation	Definition
AESI	Adverse Events of Special Interest
ADaM	Analysis Data Model
ARM	Analysis Results Metadata
CDISC	Clinical Data Interchange Standards Consortium
cHLGT	Company-defined High Level Group Term strategy
CMQ	Company-defined Customized MedDRA Query
cPT	Company-defined Preferred Term strategy
cSMQ	Customized Standardized MedDRA Query
EOSI	Events of Special Interest
FMQ	FDA Medical Query
GPP	Good Programming Practices
GxP	Good Practice (regulatory compliance)
HLGT	High Level Group Term
HLT	High Level Term
ISS	Integrated Summary of Safety
MedDRA	Medical Dictionary for Regulatory Activities
PT	Preferred Term
SMQ	Standardized MedDRA Query
SOC	System Organ Class

APPENDIX A: DETAILED SAS CODE PATTERNS

Workflow Overview

The metadata specification (Appendix E) is imported via PROC IMPORT and processed using CALL EXECUTE to invoke macros for each AESI definition. Each macro creates a lookup dataset. A final DATA step instantiates hash tables from these lookup datasets and flags records in a single pass.

Modular Development Structure: - 00_setup.sas – Environment and library assignment - 01_load_metadata.sas – Excel/CSV specification import - 02_hash_definitions.sas – CALL EXECUTE-driven macro dispatch - 03_aesi_macros.sas – Reusable lookup table generation macros - 04_main_driver.sas – Hash table instantiation and flagging - 05_validation.sas – Concordance testing via PROC COMPARE

Production Deployment: A streamlined %create_aesi_flag() macro consolidates core logic, producing identical output flags.

A.1 Hash Object Instantiation Pattern

```
data work.adae_aesi;
    /* Pre-declare all variables with explicit lengths */
    length PT_NAME $200 HLGT_NAM $200 CQ01NAM $200;
    call missing(PT_NAME, HLGT_NAM, CQ01NAM);

    /* Hash loaded once per DATA step */
    if _n_ = 1 then do;
        declare hash h_cq01(dataset: "work.lkup_cq01");
        h_cq01.definekey('PT_NAME');
        h_cq01.definedata('HLGT_NAM');
        h_cq01.definedone();
    end;

    set adam.adae;

    /* O(1) lookup: populate NAM variable if match found */
    CQ01NAM = '';
    if h_cq01.find(key: aedecod) = 0 then do;
        CQ01NAM = HLGT_NAM;
    end;
run;
```

Critical Points: - Pre-declare all data variables with LENGTH before declare hash - Use CALL MISSING() to initialize variables before hash definition - The .find() method returns 0 on success, non-zero on key not found - Hash objects are most efficient for 100+ rows; for small lookups (< 10 rows), FORMAT-based lookups may suffice

A.2 Production Macro: Specification-Driven Lookup Table Creation

The production macro processes the Excel specification and creates strategy-specific lookup tables using PROC SQL:

Specification Variables Used:

Variable	Purpose
AESI_VAR	Target variable name (e.g., CQ01NAM) used to name lookup table
STRATEGY	Source table type: cHLGT/cPT → xxx_hier, cSMQ → xxx_ssc, SMQ → content
PROG_CODE	Primary WHERE clause filter (MedDRA codes or criteria)
CONDITION	Additional filter combined with AND logic

Code Pattern:

```
/* Construct WHERE clause from PROG_CODE and CONDITION */
%if %length(&condition) > 0 and &condition ne 1=1 %then %do;
    %let where_cond = (&codes) and (&condition);
%end;
%else %do;
    %let where_cond = &codes;
%end;

/* Example: cHLGT strategy creates lookup from hierarchy table */
%if %upcase(&strategy) = CHLGT %then %do;
    proc sql;
        create table work._lkup_&base as
        select distinct PT_NAME, HLGT_NAM
        from _meddra.xxx_hier
        where &where_cond;
    quit;
%end;
```

A.3 Variable Naming Convention: FL vs NAM Suffixes

ADaM standards use paired suffixes for categorical flags:

Suffix	Purpose	Variable Length	Flagged Value	Unflagged Value
FL	Binary flag	$1	'Y'	'' (blank)
NAM	Category name	$200	Text description	'' (blank)

Production Pattern:

```
/* Extract base variable name: CQ01NAM (7 chars) minus 3 = CQ01 */
base = substr(AESI_VAR, 1, length(AESI_VAR) - 3);

/* NAM variable receives category description, not 'Y' */
if h_cq01.find(key: aedecod) = 0 then do;
    CQ01NAM = "&aesi_desc";  /* e.g., "Urinary Events" */
end;
```

Comparison Semantics:

```
/* FL variables: always compare to 'Y' */
where CQ01FL = 'Y';

/* NAM variables: always compare to non-blank */
where CQ01NAM ne '';
```

Critical Learning from AI Development: This semantic distinction was a frequent source of errors when AI-generated code treated NAM as binary flags. The skills file explicitly documents: "NAM suffix = 3 characters; comparison must use ne "."

APPENDIX B: ALGORITHMIC COMPLEXITY ANALYSIS

Complexity Comparison

Approach	Per-Strategy Complexity	m Strategies	Total Complexity	Behavior
PROC SQL Merge	O(n log n)	5	O(5n log n)	Non-linear degradation with strategies
Hash Table	O(n)	5	O(n)	Linear single pass

For a typical ISS dataset with 100,000+ records and 4-8 AESI strategies: - **Traditional SQL:** 4-8 data passes with sorting overhead (~1-2 minutes per study) - **Hash table:** Single in-memory pass (~10-15 seconds per study)

Memory Efficiency

Lookup tables are typically small (50-500 rows per strategy, <10 MB total). The ADAE dataset is streamed row-by-row via SET statement—never loaded entirely into memory. This approach scales to datasets of any size. For memory optimization, the HASHEXP= option can tune internal bucket allocation.

Example: For 8 strategies with 100 PTs each, memory usage ≈ 1-2 MB combined across all hashes.

APPENDIX C: ERROR RISK CLASSIFICATION MATRIX

Risk Level	Error Type	Description	Detection	Mitigation
LOW	Syntactic	Missing semicolons, invalid syntax	High (SAS Log)	Automated log scanning, basic syntax checking
MEDIUM	Environmental	Hardcoded paths, missing library assignments, namespace collisions	Medium (Runtime)	Standards file with prohibited patterns, linting
HIGH	Semantic	Code executes but logic is flawed (e.g., checking NAM for 'Y' instead of FL)	Low (Silent failure)	Independent programmer review, PROC COMPARE validation

Typical scenario: AI-generated code compiles and runs without errors but produces silent logic failures. Flagging never occurs because the condition check is wrong. These escape automated QC unless explicit validation logic (Section A.1) is employed.

APPENDIX D: SKILLS FILE STRUCTURE AND GOVERNANCE

The skills file is a machine-readable markdown document organized into five sections:

11. **Quick Fix Index:** Common errors and corrections discovered by programmers
12. **Pre-Flight Checks:** Mandatory rules enforced before code review
13. **Hash Table Patterns:** Code templates with rationale
14. **Namespace Protection:** Forbidden variable/macro names and replacements
15. **Variable Semantics:** FL vs NAM, CALL MISSING() requirements, LENGTH declarations

Mandatory Rules (Pre-Flight):

16. Before ANY declare hash, pre-declare all data variables with explicit LENGTH statements
17. Initialize all data variables with CALL MISSING() before hash definition
18. Forbidden macro variables: &root, &curdir, &path, &project, &study (replace with &aesi_root, &aesi_cur, etc.)
19. FL variables (length $1) must use = 'Y' in comparisons; NAM variables (length $200) must use ne '' in comparisons

Continuous Improvement Protocol:

When programmers find errors during validation: 1. Document the error pattern and root cause 2. Add to Quick Fix Index with corrected example code 3. Update Pre-Flight Checks or Patterns section to prevent recurrence 4. Subsequent AI sessions inherit this institutional knowledge

Effectiveness Data:

- **Without skills file:** ~71% of AI-generated code had subtle semantic errors requiring manual correction
- **With skills file:** ~8% error rate; remaining errors are logical (not syntactic or semantic), requiring domain knowledge review

This aligns with emerging research on "test-time compute" where structured guidance at generation time significantly improves domain-specific code quality.

APPENDIX E: MACHINE-EXECUTABLE SPECIFICATION STRUCTURE

E.1 Specification Variables and Definitions

Variable	Length	Description
AESI_DESC	$200	Human-readable description (e.g., "Urinary Events")
AESI_VAR	$20	Target variable name in ADAE (e.g., CQ01NAM)
CONCEPT	$100	Clinical concept being flagged
STRATEGY	$10	Lookup strategy: cHLGT, cPT, cSMQ, SMQ
SOURCE	$20	MedDRA dictionary dataset identifier
PT_VAR	$20	Key matching variable (typically "PT_NAME")
CONDITION	$500	Additional WHERE clause filter
PROG_CODE	$1000	Primary programming code for hash WHERE clause
PROG_NAME	$200	Description of returned category values
MEDDRA_VER	8	Numeric MedDRA version for audit trail

E.2 Strategy Types with Examples

Strategy	Source Dataset	Example PROG_CODE
cHLGT	xxx_hier	HLGT_COD in ('10004994', '10018188') AND PRIM_SOC = 'Y'

Strategy	Source Dataset	Example PROG_CODE
cPT	xxx_hier	PT_COD in (‘10000001’, ‘10000002’)
cSMQ	xxx_ssc	SMQ_ID = ‘E2B.R2’ AND SCOPE = 2
SMQ	content	SMQ_CODE = 20000147 AND STATUS = ‘A’

E.3 Specification Import and Validation

```
/* Import specification from Excel with data validation */
proc import datafile="&aesi_spec_path"
    out=work.aesi_spec
    dbms=xlsx replace;
    range="Specifications$A1:J999";
run;

/* Quick validation: check required columns exist */
%macro check_columns;
    proc sql noprint;
        select count(*) into :col_count
        from dictionary.columns
        where libname="WORK" and memname="AESI_SPEC"
        and upcase(name) in ("AESI_VAR", "AESI_DESC", "PROG_CODE");
    quit;

    %if &col_count ne 3 %then %do;
        %put ERROR: Required columns missing from specification;
        %abort;
    %end;
%mend;
%check_columns;
```

E.4 Regulatory Traceability: Define.xml Population

The specification directly populates Analysis Results Metadata (ARM) in Define.xml:

Specification Column	Define.xml Element	Define.xml Role
AESI_VAR	Variable/Name	Target variable in ADAE dataset
AESI_DESC	Variable/Label	Variable label/definition in submission
PROG_CODE + CONDITION	Derivation/Formula	Complete MedDRA selection logic
SOURCE	Origin/Type	“External Dictionary Reference”
MEDDRA_VER	Origin/Version	MedDRA version

Traceability Workflow:

20. Programmer updates Excel specification
21. Macro processes specification → SAS code and Define.xml metadata simultaneously
22. No manual Define.xml editing; all changes flow from specification
23. PROC COMPARE validation ensures code output matches ARM documentation

This achieves 100% specification-to-data concordance, addressing compliance gaps commonly cited in FDA/EMA inspections.

APPENDIX F: REGULATORY ALIGNMENT WITH EMA/FDA AI PRINCIPLES

The AESI flagging approach aligns with the joint EMA/FDA “Guiding Principles on the Use of Artificial Intelligence in Drug and Biological Product Development” (2025):

EMA/FDA Principle	Our Implementation	Mechanism
Human-Centric Design	All AI-generated code undergoes independent programmer review	Code review SOPs; no AI-generated code deployed without sign-off

EMA/FDA Principle	Our Implementation	Mechanism
Risk-Based Approach	Tiered validation: development < test < production environments	Concordance testing proportionate to risk level
Standards Compliance	Skills file enforces PHUSE GPP and GxP coding rules	Automated checks; skills file updates with discovered patterns
Transparent Documentation	All code outputs documented; no hidden AI transformations	Plain SAS code; no intermediate transformations
Life Cycle Management	Version control (Git) for specification and code; revalidation planned with MedDRA updates	Audit trail of changes; pre-defined revalidation triggers

CONTACT INFORMATION

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

Author Name: Hang Zhong

Company: Bristol Myers Squibb

Address: Seattle, WA, USA

Email: hang.zhong@bms.com