Paper SM08

AI-Assisted Metadata Programming: A Transparent Framework for PCS in Lab Data Analysis

Donghua Zeng, BMS, USA
Taniya Muliyil, BMS, USA

Abstract

Statistical programmers often face the challenge of translating complex protocol or SAP text into thousands of lines of conditional code. This manual process is time-consuming, error-prone, and difficult to audit. Traditional approaches to deriving Potentially Clinically Significant (PCS) flags in lab data rely heavily on study-specific coding, which reduces consistency and slows delivery.

This presentation demonstrates an AI-assisted, metadata-driven approach that streamlines programming while enhancing transparency and reproducibility. Free-text rules from protocols and SAPs are translated into Boolean criteria within structured metadata (Excel/TSV). Standard SAS or R macros then apply the metadata to automate PCS derivations. Programmers no longer re-code logic manually; instead, they review and refine metadata tables, ensuring traceability from text → metadata → code → results.

This approach improves quality, shortens validation and study-specific tailoring, and promotes audit readiness. While some customization may still be required for unique protocols, it establishes a scalable framework that elevates programmers from data processors to clinical logic architects. In a pilot example (28 analytes; 77 PCS criteria across Low/High directions), the AI-assisted workflow reduced estimated PCS rule derivation time from ~2–3 days (typical manual coding practice) to <1 hour, while retaining traceability through archived specification and QC evidence files.

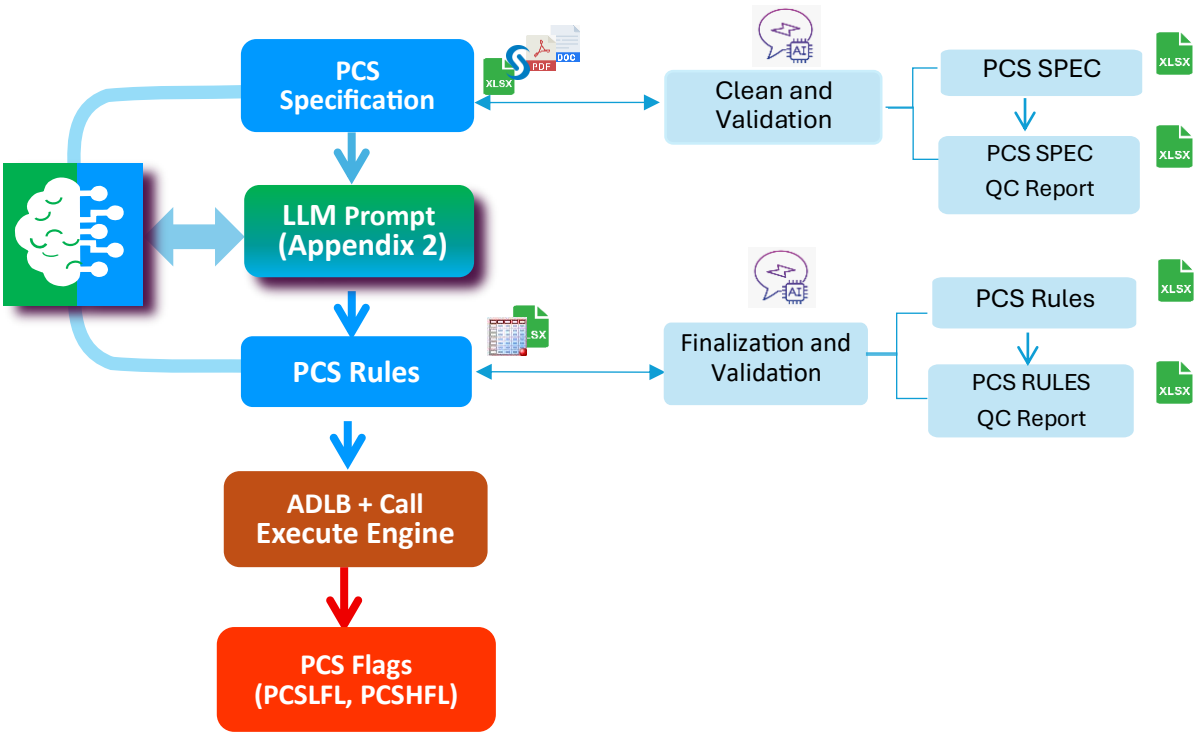
1. Introduction

Potentially Clinically Significant (PCS) or “Marked Abnormality” laboratory results play a critical role in assessing patient safety across clinical trials. Although PCS rules are typically defined in the SAP or Protocol, they are often written in narrative form, making direct translation into programming logic challenging.

Generative AI offers a new opportunity to streamline this process. In particular, **Microsoft Copilot** can help transform human-authored PCS specifications into machine-readable rule components suitable for ADLB derivation when provided with a clear, standardized input. This paper presents a **simplified, AI-assisted PCS derivation framework** that emphasizes preparing a clean, consistent PCS specification table before applying rule generation. Through light pre-cleaning and structured prompting, Copilot can reliably produce BOOLEAN and PRECOND expressions that support standardized PCS flagging. The

resulting workflow reduces manual interpretation, minimizes downstream QC burden, and improves reproducibility across studies.

Figure 1. Mainstream for LLM-assisted PCS derivation with supporting preparation and QC branches



2. SAP/Protocol Specification Table Preparation

PCS (or MA, Marked Abnormality) thresholds are typically scattered across the SAP or Protocol in narrative form. Before any AI-based rule extraction can occur, these specifications must be transformed into a simplified and analysis-ready table. This section describes how the original SAP content is streamlined and precleaned to support accurate and standardized rule generation in later steps.

2.1 Simplified SAP/Protocol PCS Specification Table

The first step is to extract the relevant PCS information from the SAP/Protocol and convert it into a concise table format. Only essential elements are retained, such as:

- **PARAMCD**

- **Low (L) textual rule**
- **High (H) textual rule**
- **Optional comments or qualifiers**

This simplified table removes narrative phrasing and focuses exclusively on the PCS logic needed for downstream processing.
Table 1 and **Table 2** show representative examples of this streamlined specification.

This simplified PCS_SPEC table becomes the standardized input dataset for both pre-cleaning and AI-assisted rule generation.

Table 1. Original Hematology Excerpt (Human-Readable Format)

Clinical Laboratory	Units	Low MA Criterion	High MA Criterion
HEMATOLOGY			
Hemoglobin	g/L	> 20 g/L decrease compared to pre-dose or value ≤ 80 g/L	
Hematocrit	Percent	< 0.75 x pre-dose	
Erythrocytes	$x10^{12}$ c/L	< 0.75 x pre-dose	
Platelet Count	$x10^{9}$ cells/L	< 100 $x10^{9}$ cells/L	
Leukocytes	$x10^{9}$ cells/L	< 0.75 x LLN, or if pre-dose < LLN then use < 0.8 x pre-dose if pre-dose > ULN then use < LLN	> 1.25 x ULN, or if pre-dose < LLN then use > ULN if pre-dose > ULN then use > 1.2 x pre-dose

Table 2. Simplified PCS specification sheet (Excerpt) before AI-assisted process

PARAMCD	Low_text	High_text
HGB	AVAL < (BASE-20); AVAL <= 80	
HCT	< 0.75 x BASE	
RBC	< 0.75 x BASE	
PLAT	< 100	
WBC	< 0.75 x LLN	> 1.25 x ULN
WBC	if BASE < LLN then use < 0.8 x BASE;	if BASE < LLN then AVAL > ULN;
WBC	if BASE > ULN then AVAL < LLN	if BASE > ULN then AVAL > 1.2 x BASE

2.2 AI-Assisted Pre-Cleaning of PCS Text

Even after simplification, the textual content frequently contains elements that are difficult for AI models to interpret directly. Examples include: leftover English connectors, missing comparators, inconsistent

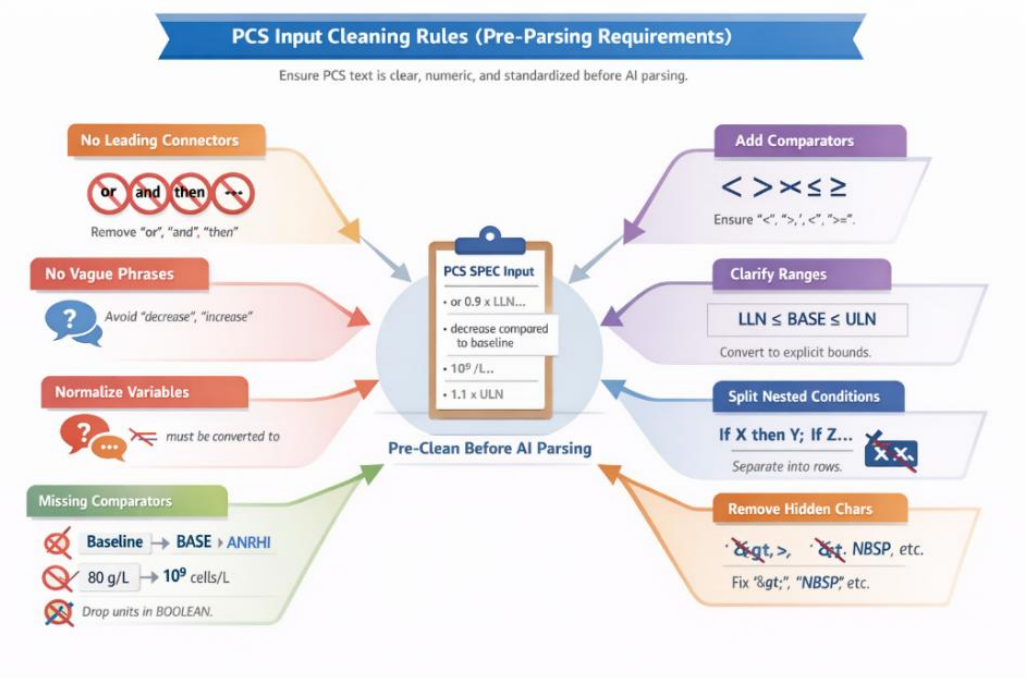
baseline terminology, units embedded in numeric expressions, or hidden non-ASCII characters from PDF copy/paste.

To address this, the simplified PCS_SPEC undergoes a dedicated **AI pre-cleaning step**, during which Copilot inspects each Low/High rule and produces:

1. **PCS_SPEC_clean.xlsx**
    - All expressions normalized (e.g., BASE/ANRLO/ANRHI)
    - Comparators made explicit
    - Units removed from BOOLEAN-relevant expressions
    - Hidden characters and formatting artifacts fixed
    - Leading connectors (“or”, “and”) removed
2. **PCS_SPEC_clean_report.xlsx**
    - Side-by-side comparison of original vs cleaned text
    - QC flags documenting each adjustment (e.g., Drop Units, Missing Comparator, Normalize Boolean)

Figure 2 illustrates the pre-cleaning rules applied at this stage and the standardization required for reliable parsing. This step establishes a standardized foundation for Copilot to generate the final PCS BOOLEAN and PRECOND logic in Section 3.

Figure 2. PCS Specification Pre-Cleaning Workflow by AI



2.3 Validation Protocol — PCS Specification

Purpose
This section formally defines the validation protocol used to confirm that the cleaned PCS specification table (PCS_SPEC_clean.xlsx) faithfully preserves SAP/Protocol intent and is suitable as stable input for AI-assisted rule parsing.

Key points (keep concise in main text):

- Validation is performed using an **AI-assisted checklist prompt**.
- Validation output is an **audit-ready evidence table**.
- **Human review is targeted** only to rows flagged by the validation step.

Implementation details, prompts, and evidence artifacts are provided in **Appendix 1.**

3. AI Prompt for Parsing PCS Rules

Once the PCS specification table has been simplified and pre-cleaned, the next step is to convert the standardized textual rules into structured PCS logic. This is achieved by using **Microsoft Copilot** with a standardized prompt designed to extract BOOLEAN, PRECOND, PCSSEQ, and a criteria description from each Low/High rule.

The use of AI at this stage shifts the workflow away from manual interpretation and large SAS QC programs toward a reproducible, metadata-driven process. Because the input text has already been normalized (Section 2), Copilot can consistently translate each PCS expression into a machine-readable format suitable for direct application in ADLB.

3.1 AI Prompt Framework

The PCS prompt follows a fixed structure and instructs Copilot to:

- Read each PARAMCD and its corresponding Low/High text
- Convert standardized terms (BASE, ANRLO, ANRHI, AVAL) into numeric logic
- Produce BOOLEAN expressions using explicit comparators
- Generate PRECOND statements for required variable availability
- Assign PCSSEQ in the order rules appear
- The prompt is designed to produce reproducible outputs under controlled conditions (fixed prompt version and file-based inputs), producing consistent BOOLEAN and PRECOND

outputs suitable for downstream execution and review. Controlled conditions include a versioned prompt, file-based inputs (rather than pasted text), and archived intermediate artifacts to support traceability and repeatability

See Appendix 2 for full Prompt.

3.2 AI-Generated Outputs

Copilot returns two aligned deliverables:

1. **PCS_RULES_clean.xlsx**
    - Final BOOLEAN and PRECOND expressions
    - PCSSEQ ordering
    - Ready for direct evaluation in ADLB
2. **PCS_RULES_clean_report.xlsx**
    - Documentation of AI-applied adjustments
    - QC flags indicating unit removal, comparator inferences, or normalization steps
    - A traceable audit trail for reviewers

These two files represent the transition from textual SAP specifications to an executable PCS ruleset.

3.3 Validation Protocol — PCS Rule Parsing (See Appendix 3)

Although the AI produces fully structured Boolean logic, a brief human check remains essential. The reviewer ensures that:

- The direction (Low/High) aligns with clinical intent
- Comparators (<, >, <=, >=) match SAP intent
- BASE/LLN/ULN relationships are correct
- PCSSEQ represents expected rule ordering

Because the input text has been precleaned and the prompt is standardized, this validation step is minimal compared to traditional QC workflows. QC outcomes are severity-stratified (PASS, PASS_WITH_WARNINGS, REVIEW); only REVIEW findings require human correction.

3.4 AI-Assisted Meta-Approach Summary and Efficiency Impact

This section summarizes the operational impact of the proposed AI-assisted meta-approach for PCS rule derivation.

Traditional PCS programming requires manual interpretation of SAP/Protocol text, iterative rule coding, and repeated full-dataset QC cycles. In contrast, the proposed approach uses AI to perform structured rule extraction and validation, allowing human effort to be focused on targeted review of flagged items rather than exhaustive manual construction.

Table 3 provides a high-level comparison of the manual and AI-assisted workflows, highlighting elapsed time and human effort across major steps. The intent of this summary is to illustrate relative efficiency gains rather than to prescribe absolute timing, which may vary by study complexity and organizational practice.

In the pilot example evaluated in this study, the PCS specification comprised 28 analytes, resulting in 77 distinct PCS rule criteria when both Low and High directions were considered.

Table 3. Summary of PCS Rule Derivation Workflow and Relative Time Requirements

Step	Manual approach	AI-assisted approach
PCS spec preparation	1 day	10 min
Rule derivation	1–2 days (SAS coding)	5 min
QC & review	0.5–1 day	10 min
Total elapsed time*	**~2–3 days**	**<1 hour**

*Note: Times are estimated based on the pilot example study (28 analytes; 77 criteria) and current study coding practice; actual effort may vary by study complexity and organizational workflow.

4. Final Application of PCS Specifications to ADLB

The final step in the workflow is the direct application of the validated PCS specification (PCS_RULES) to the ADLB dataset. This is performed using a dynamic SAS programming technique based on CALL EXECUTE, which assembles and runs rule-specific logic at runtime. Because the PRECOND and BOOLEAN expressions are stored as literal text within PCS_RULES, CALL EXECUTE provides a flexible and powerful way to evaluate each PCS condition against the ADLB data.

4.1 Dynamic Rule Execution Using CALL EXECUTE

The following SAS code iterates through each rule in PCS rules and generates a data step that applies all PCS logic to each ADLB record. For each PARAMCD and direction (Low/High), the PRECOND and BOOLEAN expressions are evaluated sequentially in ascending PCSSEQ order, and intermediate results are accumulated.

See Appendix 4 for SAS Codes (CALL EXECUTE process)

All generated rules are compile-validated prior to full ADLB execution; any non-executable rule is flagged as REVIEW and blocks release until resolved.

When converting PCS_RULES_clean.xlsx to a SAS dataset (e.g., PCS_RULES.sas7bdat) using SAS Transport (XPT v5), XPT-friendly naming conventions may be applied (e.g., 8-character variable name constraints).

Key variables are:

- PARAMCD (unchanged)
- PCSSEQ (unchanged, unique PCS criteria sequential number)
- DIREC (renamed from DIRECTION; Low/High indicator)
- PCSDESC *(renamed from SOURCE_TEXT; rule text / description)*
- PRECOND *(condition for rule applicability)*
- BOOLEAN *(PCS Boolean condition)*

Table 4 Typical PCS_RULE metadata table (except)

PARAMCD	DIREC	PCSSEQ	PCSDESC	PRECOND	BOOLEAN
HGB	L	1	AVAL < (BASE-20)	not missing(AVAL) and not missing(BASE)	AVAL < (BASE-20)
WBC	H	1	AVAL > 1.25 x ULN	not missing(AVAL) and not missing(ANRHI)	AVAL > 1.25 * ANRHI

4.2 PCS Results (Example Interpretation)

- _PCSLY_ accumulates all *positive* (Y-flag) low-direction hits
- _PCSLN_ accumulates all *negative* evaluations
- PCSLFL reflects the worst outcome:
    - Y if *any* condition is met
    - N if all available/eligible conditions are negative

Key examples:

- Subject 101, Visit 1 (WBC)
    - No low direction hits → PCSLFL = N
- Subject 101, Visit 2 (WBC)
    - _PCSLY_ = Y2 → PCSLFL = Y
- Subject 101, Visit 3 (WBC)
    - _PCSLY_ = Y2 | Y1 → PCSLFL = Y

This matches the clinical interpretation of PCS rules: any worsening triggers a flag. High direction is processed similarly.

Table 5. Low Direction PCS Results (simulated)

Obs	SUBJID	PARAMCD	VISIT	AVAL	BASE	ANRLO	ANRHI	_PCSLY_	_PCSLN_	PCSLFL
1	101	WBC	visit1	18.0	2.9	3	10		N2|N1	N
2	101	WBC	visit2	2.3	2.9	3	10	Y2	N1	Y
3	101	WBC	visit3	1.8	2.9	3	10	Y2|Y1		Y
4	103	SODIUM	visit1	132.0	.	135	145		N1	N
5	104	SODIUM	visit1	90.0	.	135	145	Y1		Y

(only critical variables shown)

Table 6. High Direction PCS Results(simulated)

Obs	SUBJID	PARAMCD	VISIT	AVAL	BASE	ANRLO	ANRHI	_PCSHY_	_PCSHN_	PCSHFL
1	101	WBC	visit1	18	2.9	3	10	Y2|Y1		Y
2	101	WBC	visit2	2.3	2.9	3	10		N2|N1	N
3	101	WBC	visit3	1.8	2.9	3	10		N2|N1	N
4	103	SODIUM	visit1	132	.	135	145		N1	N
5	104	SODIUM	visit1	90	.	135	145		N1	N

(only critical variables shown)

4.3 Summary of ADLB Integration

- PCS logic is generated from clean and standardized rule text.
- The prompt is designed to be with SAS execution, requiring no additional SAS QC repair code.
- The method integrates directly with existing ADLB programming workflows.
- Final PCS outputs (PCSLFL, PCSHFL) can be used for listings, summaries, or downstream safety reviews.

5. Future Expectations

The approach described in this paper demonstrates how PCS derivation can shift from manual, SAS-intensive workflows toward automated, text-driven rule generation supported by AI. As sponsors refine SAP authoring practices and adopt standardized PCS specification tables, the pre-cleaning and AI-parsing framework can be readily integrated into broader metadata-driven pipelines.

Future enhancements may include:

- Embedding PCS specification templates directly in SAP/Protocol authoring systems
- Automating pre-cleaning as part of the document-to-metadata ingestion process

- Integrating Copilot with sponsor-level rule libraries for cross-study consistency
- Extending the approach to other domains where SAP instructions are expressed narratively (eg, visit windows, treatment emergence, baseline definitions)
- Incorporating validation dashboards that visualize AI-generated BOOLEAN and PRECOND logic for rapid reviewer confirmation

As AI models continue to evolve, we anticipate increasingly stable and fully traceable rule generation across studies, enabling faster delivery timelines and reducing redundant QC effort.

6. Conclusions

This paper outlines a streamlined, AI-assisted framework for converting PCS specifications into executable programming logic. By standardizing SAP/Protocol text through a focused pre-cleaning step, Copilot can consistently generate BOOLEAN and PRECOND expressions that support standardized PCS flagging with minimal manual intervention.

The resulting workflow reduces reliance on large SAS QC programs, improves reproducibility, and aligns PCS derivation with modern metadata-driven practices. This approach provides a practical and scalable path toward automated PCS rule implementation across clinical studies.

REFERENCES

SDTMIG v3.2 https://www.cdisc.org/standards/foundational/sdtm
ADaMIG v1.3 https://www.cdisc.org/standards/foundational/adam

Acknowledgments

The authors would like to thank BMS management, Rashad Rasul for supporting the project and providing the necessary resources for its execution. Special thanks to Yang Liu for the effort to test AI Prompt.

Contact Information

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

Author Name: Donghua Zeng
Company Name: Bristol Myers Squibb, USA
Company Address: 3551 Lawrenceville Rd., Princeton, NJ 08540 USA
Email: donghua.zeng@bms.com

Author Name: Taniya Muliyil
Company Name: Bristol Myers Squibb, USA
Company Address: 3551 Lawrenceville Rd., Princeton, NJ 08540 USA
Email: taniya.muliyil@bms.com

APPENDICES

Appendix 1. Validation Checklist — PCS Specification (Input Validation) + AI Prompt

A1.1 Purpose

This checklist verifies that the simplified and cleaned PCS specification table (**PCS_SPEC_clean.xlsx**) faithfully preserves the SAP/Protocol intent and is suitable as stable input for downstream rule parsing and SAS derivation.

A1.2 Evidence Artifacts (Audit-Ready)

Store the following files as **audit-ready evidence**:

- **PCS_SPEC_clean.xlsx** (final cleaned input table)
- **PCS_SPEC_CLEAN_REPORT.xlsx** (AI-assisted validation report + reviewer notes)

A1.3 AI-Assisted Validation Prompt (Copilot) — "PCS_SPEC Input Review"

How to Run (≈1 minute)

- Upload or open **PCS_SPEC_clean.xlsx** in a Copilot-enabled environment (Excel Copilot or Copilot Chat).
- Load the prompt below.
- Save the resulting table output as **PCS_SPEC_CLEAN_REPORT.xlsx** (or paste into Excel and save).
- Perform human review only on rows that Copilot indicates need attention.

Prompt

TITLE: PCS_SPEC Pre-clean + Report Prompt (PCS_SPEC_CLEAN_PROMPT_v1.0)

ROLE
You are a clinical programming QC assistant. Clean PCS specification text for parsing readiness and produce a side-by-side report.

INPUT
I will provide a table with columns:
- PARAMCD
- Low_text
- High_text

TASK
For each row:
1) Create cleaned versions of Low_text and High_text by applying these rules:
- Example: < → <, > → >
- Remove non-ASCII or hidden whitespace (e.g., NBSP)

- Normalize spacing (single spaces)
- Replace multiplication symbol "×" with "x"
- If the text starts with a comparator (<, <=, >, >=), prefix with "AVAL "
- Remove trailing semicolons and standardize "; " spacing
- Do NOT invent thresholds or change clinical meaning

2) Flag whether any non-ASCII characters were present in the original text.

3) Summarize what was changed.

OUTPUT
Return a single table with EXACTLY these columns (tab-separated):
- PARAMCD
- Low_text_original
- Low_text_clean
- High_text_original
- High_text_clean
- NonASCII_in_original (Y/N)
- Changes_applied (short text)

RETURN FORMAT
Return ONLY the table with a header row. No extra commentary.

A1.4 Human Confirmation (Targeted Review)

The human reviewer examines only the rows flagged as needing attention, focusing on:

- **Rule granularity:** multi-rule analytes remain separate rows; no unintended merges
- **Directionality:** Low vs High categories remain correct
- **Comparator / threshold fidelity:**
  < vs <=, > vs >=, and all numeric multipliers remain unchanged
- **Baseline intent:** "BASE" usage correctly reflects SAP meaning

Reviewer records one of the following outcomes:

- **PASS** — accepted as is
- **FIXED** — minor correction applied and re-checked
- **ESCALATE** — requires SAP/clinical confirmation

A1.5 Acceptance Criteria

Appendix 1 validation is considered complete when:

- All rows are **PASS**, or all **REVIEW/FAIL** items have documented resolution; and
- The final **PCS_SPEC_clean.xlsx** and **PCS_SPEC_CLEAN_REPORT.xlsx** are archived as audit-ready evidence.

Appendix 2. PCS Rule Generation Prompt

Prompt

TITLE: PCS Rule Generation Prompt (PCS_RULE_GEN_FINAL)

ROLE
You are a clinical programming assistant. Convert cleaned PCS rule text into structured rule components suitable for SAS evaluation in ADLB.

INPUT
Use the Excel table PCS_SPEC_clean with columns:
- PARAMCD
- Low_text
- High_text

DEFINITIONS
- PRECOND defines when a rule applies.
- BOOLEAN defines the executable AVAL comparison.
- PCSSEQ enforces stable ordering within each PARAMCD + DIRECTION.

NORMAL RANGE MAPPING (Required)
- In PRECOND and BOOLEAN only, map:
LLN -> ANRLO
ULN -> ANRHI
- Keep SOURCE_TEXT unchanged for traceability.

REQUIRED OUTPUT (EXCEL ONLY)
Create an Excel table in a NEW worksheet named PCS_RULES_clean with EXACT columns:
PARAMCD | DIRECTION | PCSSEQ | SOURCE_TEXT | PRECOND | BOOLEAN | VARIABLES
Return only the table. Do NOT output TSV. Do NOT add commentary.

PARSING RULES
A) Direction:
- Low_text non-blank => DIRECTION=L
- High_text non-blank => DIRECTION=H

B) Multi-rule cell:
- If SOURCE_TEXT has multiple AVAL comparisons separated by ";", split into separate output rows.
- Each output row must contain exactly ONE BOOLEAN expression.

PRE-PROCESSING (Required) — Fix any HTML escapes
Before interpreting SOURCE_TEXT, PRECOND, or BOOLEAN, convert any HTML escape sequences:
- Replace "<=" with "<="
- Replace ">=" with ">="
- Replace "<" with "<"
- Replace ">" with ">"
- Replace "&" with "&"

Apply this cleanup to SOURCE_TEXT, PRECOND, and BOOLEAN before all other steps.

C) Extract PRECOND/BOOLEAN:
- Simple: if SOURCE_TEXT begins with "AVAL <" / "AVAL >" / "AVAL <=" / "AVAL >=":
PRECOND = blank
BOOLEAN = SOURCE_TEXT
- Conditional: if SOURCE_TEXT matches "If <condition> then <aval_statement>":
PRECOND = <condition>
BOOLEAN = <aval_statement>
- Convert multiplication in BOOLEAN: "x" -> ""*
- Convert "BASE = missing" or "BASE is missing" to: missing(BASE)

D) Apply LLN/ULN mapping in executable fields:
- Replace LLN with ANRLO in PRECOND/BOOLEAN
- Replace ULN with ANRHI in PRECOND/BOOLEAN

E) SAS RANGE NORMALIZATION (Required)
- Replace chained comparison like "ANRLO <= BASE <= ANRHI" with:
"BASE >= ANRLO AND BASE <= ANRHI"

F) PRECOND MISSING-GUARDS (Required)

PRECEDENCE (Required)
- If SOURCE_TEXT indicates BASE missing ("BASE = missing" or "BASE is missing"):
1) PRECOND must include missing(BASE)
2) PRECOND must NOT include not missing(BASE) under any circumstance
3) Add only other needed guards: not missing(AVAL) and not missing(ANRLO/ANRHI) as applicable

Otherwise (non-BASE-missing rules):
- Always add: not missing(AVAL)
- If PRECOND/BOOLEAN references BASE, add: not missing(BASE)
- If PRECOND/BOOLEAN references ANRLO, add: not missing(ANRLO)
- If PRECOND/BOOLEAN references ANRHI, add: not missing(ANRHI)

G) VARIABLES
- VARIABLES must list tokens used in PRECOND/BOOLEAN after mapping:
(AVAL, BASE, ANRLO, ANRHI as applicable) in pipe-delimited form.

H) PCSSEQ
- PCSSEQ starts at 1 within each PARAMCD + DIRECTION and increases sequentially.
- Preserve input order from PCS_SPEC_clean; split parts by ";" get consecutive PCSSEQ.

RETURN FORMAT
Return ONLY the PCS_RULES_clean Excel table with header row. No additional commentary.

Do not perform any transformation other than the explicit steps listed above. If a rule cannot be expressed using only those steps, output it unchanged and let QC flag it.

Appendix 3. Validation Checklist — PCS Rules (Rule Parsing + QC) + AI Prompt

A3.1 Purpose

This checklist verifies that the AI-generated PCS rules table (**PCS_RULES_clean.xlsx**) correctly represents the cleaned PCS specification (**PCS_SPEC_clean.xlsx**) and produces SAS-executable rule components (PRECOND/BOOLEAN) with stable ordering (PCSSEQ).

A3.2 Evidence Artifacts (Audit-Ready)

Store the following files as audit-ready evidence:

- **PCS_RULES_clean.xlsx** (final parsed rule table with DIRECTION, PCSSEQ, SOURCE_TEXT, PRECOND, BOOLEAN, VARIABLES).
- **PCS_RULES_CLEAN_REPORT.xlsx** (QC report with PASS / PASS_WITH_WARNINGS / REVIEW and issue flags).

A3.3 AI-Assisted Generation Prompt (Copilot) — "PCS Rule Generation"

How to Run (≈1 minute)

- Upload/open **PCS_SPEC_clean.xlsx** in a Copilot-enabled environment (Excel Copilot recommended).
- Load the **Generation Prompt** (Appendix 2).
- Save the resulting output table as **PCS_RULES_clean.xlsx**.

Prompt Note (Important): Do not paste PCS_SPEC table rows into Copilot chat. Always load the Excel file and let Copilot read the table directly. If prompts are stored in Word, copy into a plain-text editor (Notepad) before pasting into Copilot to avoid silent operator encoding (e.g., < → <). It is recommended **M365 Copilot** users loading the PROMPT.txt and PCS_SPEC excel table into **Chatbox**. Avoid directly pasting.

A3.4 AI-Assisted QC Prompt (Copilot) — "PCS Rule Validation"

How to run (≈1 minute)

- *Upload/open **PCS_SPEC_clean.xlsx** and **PCS_RULES_clean.xlsx** in the Copilot-enabled environment.*
- *Load the **QC Prompt** below (severity-based).*
- *Save the resulting output table as **PCS_RULES_CLEAN_REPORT.xlsx**.*
- *Perform human review only for rows flagged **REVIEW** (ERROR-level issues).*

Prompt

TITLE: PCS Rule Validation Prompt (PCS_RULE_QC_FINAL_v2) — Severity-Based

ROLE
You are an independent QC reviewer. Validate that PCS_RULES_clean correctly represents PCS_SPEC_clean and is executable in SAS.
Do NOT over-flag: only logic-breaking issues should trigger REVIEW. Minor hygiene issues should be WARN only.

INPUTS
You will be given TWO Excel tables:
(1) PCS_SPEC_clean with columns: PARAMCD, Low_text, High_text
(2) PCS_RULES_clean with columns: PARAMCD, DIRECTION, PCSSEQ, SOURCE_TEXT, PRECOND, BOOLEAN, VARIABLES

IMPORTANT WARNING
Do NOT copy/paste PCS_SPEC text into chat. Always open/upload the Excel files so operators (<, >, <=, >=) are not altered by encoding.

REQUIRED OUTPUT (EXCEL ONLY)
Create an Excel table in a NEW worksheet named PCS_RULES_CLEAN_REPORT with EXACT columns:
PARAMCD | DIRECTION | PCSSEQ | SOURCE_TEXT | PRECOND | BOOLEAN | VARIABLES | QC_STATUS | ISSUES_FLAGGED | QC_NOTES
Return only the table. Do NOT output TSV. Do NOT add commentary.

SEVERITY MODEL
Classify issues into two tiers:

ERROR (must trigger QC_STATUS=REVIEW)
- SOURCE_TEXT_MISMATCH
- PCSSEQ_ORDER
- BASE_MISSING_CONTRADICTION
- BASE_MISSING_RULE_ERROR
- CHAINED_COMPARISON_NOT_SAS
- BOOLEAN_NOT_EXECUTABLE
- BOOLEAN_HAS_X
- LLN_NOT_MAPPED
- ULN_NOT_MAPPED

WARN (should NOT trigger REVIEW; only PASS_WITH_WARNINGS)
- MISSING_GUARD_AVAL
- MISSING_GUARD_BASE
- MISSING_GUARD_ANRLO
- MISSING_GUARD_ANRHI
- VARIABLES_INCOMPLETE
- DUPLICATE_PRECOND_CLAUSE (e.g., "missing(BASE) AND missing(BASE)")
- STYLE_ONLY (optional)

QC_STATUS RULE

- REVIEW if any ERROR exists
- PASS_WITH_WARNINGS if no ERROR but at least one WARN exists
- PASS if no issues exist

VALIDATION CHECKS (per row in PCS_RULES_clean)

A) Source traceability
- Confirm SOURCE_TEXT matches the corresponding PCS_SPEC_clean cell text for that PARAMCD and DIRECTION.
If mismatch: ISSUE=SOURCE_TEXT_MISMATCH (ERROR).

B) PCSSEQ checks
- Within each PARAMCD + DIRECTION:
- PCSSEQ must start at 1.
- PCSSEQ must increase by 1 with no gaps and no duplicates.
If any problem: ISSUE=PCSSEQ_ORDER (ERROR).

C) Executability blockers (SAS)
- If PRECOND contains both "missing(BASE)" and "not missing(BASE)":
ISSUE=BASE_MISSING_CONTRADICTION (ERROR).
- If PRECOND contains chained comparisons like "ANRLO <= BASE <= ANRHI"
or any "<= BASE <=" / ">= BASE >=" pattern:
ISSUE=CHAINED_COMPARISON_NOT_SAS (ERROR). Expected: "BASE >= ANRLO AND BASE <= ANRHI".
- BOOLEAN must not contain narrative words (e.g., "use", "then"):
If present: ISSUE=BOOLEAN_NOT_EXECUTABLE (ERROR).
- BOOLEAN must use "" for multiplication (no " x "):*
If present: ISSUE=BOOLEAN_HAS_X (ERROR).

D) Normal range mapping
- If SOURCE_TEXT contains LLN, then PRECOND/BOOLEAN must contain ANRLO and must not retain LLN.
If not: ISSUE=LLN_NOT_MAPPED (ERROR).
- If SOURCE_TEXT contains ULN, then PRECOND/BOOLEAN must contain ANRHI and must not retain ULN.
If not: ISSUE=ULN_NOT_MAPPED (ERROR).

E) Missing-guard checks (WARN by default)
- PRECOND should include "not missing(AVAL)".
If missing: ISSUE=MISSING_GUARD_AVAL (WARN).
- If PRECOND or BOOLEAN references BASE and PRECOND does NOT include "missing(BASE)",
then PRECOND should include "not missing(BASE)".
If missing: ISSUE=MISSING_GUARD_BASE (WARN).
- If PRECOND or BOOLEAN references ANRLO, PRECOND should include "not missing(ANRLO)".
If missing: ISSUE=MISSING_GUARD_ANRLO (WARN).
- If PRECOND or BOOLEAN references ANRHI, PRECOND should include "not missing(ANRHI)".
If missing: ISSUE=MISSING_GUARD_ANRHI (WARN).

F) BASE-missing rule correctness
- If SOURCE_TEXT indicates BASE missing ("BASE = missing" or "BASE is missing"):
PRECOND must include "missing(BASE)" and must NOT include "not missing(BASE)".
If violated: ISSUE=BASE_MISSING_RULE_ERROR (ERROR).

G) VARIABLES completeness (WARN)
- VARIABLES should include tokens used in PRECOND/BOOLEAN (at minimum AVAL; plus BASE/ANRLO/ANRHI when present).

If incomplete: ISSUE=VARIABLES_INCOMPLETE (WARN).

H) Duplicate PRECOND clause (WARN)
- If PRECOND contains duplicate identical clauses (e.g., "missing(BASE) AND missing(BASE)"):
ISSUE=DUPLICATE_PRECOND_CLAUSE (WARN).

QC_NOTES
- For REVIEW: one short sentence stating the exact fix needed.
- For PASS_WITH_WARNINGS: one short sentence stating the optional cleanup (if desired).
- For PASS: leave blank.

RETURN FORMAT
Return ONLY the PCS_RULES_CLEAN_REPORT Excel table with header row. No additional commentary.

A3.5 Human Confirmation (Targeted Review)

The human reviewer examines only rows flagged as **REVIEW**, focusing on logic-breaking or SAS-executability issues:

- **BASE missing contradiction:** PRECOND must not contain both missing(BASE) and not missing(BASE).
- **Chained comparison not SAS:** convert ANRLO <= BASE <= ANRHI into BASE >= ANRLO AND BASE <= ANRHI.
- **Mapping checks:** if SOURCE_TEXT uses LLN/ULN, executable fields must use ANRLO/ANRHI as specified.
- **BOOLEAN executability:** must contain executable comparison logic only, no narrative text, and use * for multiplication.
- **PCSSEQ sanity:** confirm sequential ordering within PARAMCD + DIRECTION as expected.

Reviewer records one of the following outcomes:

- **PASS** — accepted as is
- **FIXED** — corrected and re-checked (regenerate QC report)
- **ESCALATE** — requires SAP/clinical clarification

(PASS_WITH_WARNINGS rows do not require correction; they represent conservative or optional cleanup flags.)

A3.6 Acceptance Criteria

Appendix 3 validation is considered complete when:

- All **REVIEW** items (ERROR-level) have documented resolution (PASS or FIXED), and
- The final **PCS_RULES_clean.xlsx** and **PCS_RULES_CLEAN_REPORT.xlsx** are archived as audit-ready evidence.

Appendix 4 Dynamic Build of ADLB Rule Application

```
data _null_;
    set pcs_rules end = eof;
    if _n_ = 1 then do;
        call execute('data adlb_1;');
        call execute('   set adlb;');
        call execute('   length _PCSLY_  _PCSLN_ _PCSHY_  _PCSHN_  $50 PCSLFL  PCSHFL $1 _PCSLYC_  PCSLCRi PCSHCRi $200 ;');
        call execute('    _PCSLY_ = "";  _PCSLN_ = ""; _PCSHY_ = ""; _PCSHN_ = ""; PCSLCRi = ""; PCSHCRi = "" ;');
    end;
    call execute('   if strip(PARAMCD) = "'||strip(PARAMCD)||'" then do;');  /*TESTCD*/
    call execute('      if "'||upcase(strip(DIREC))|| '" in ("L" "LOW") then do;');  /*Direc*/
    call execute('         if AVAL > .Z then do ;') ;  /*nonmissing AVAL*/
    call execute('            if '||strip(PRECOND) || ' then do ;') ;
    call execute('               if '||strip(BOOLEAN)|| ' then do ;');  /*algorithm   AVAL   */
    call execute('                  _PCSLQ_ = "Y"||strip(' ||strip(put(PCSSEQ, best.))|| ') ;');
    call execute('                  _PCSLY_ = catx("|", _PCSLQ_, _PCSLY_) ;');
    call execute('                  _PCSLYC_ = strip(_PCSLQ_) ||": "||strip("' ||strip(pcsdesc)|| '");') ;
    call execute('                  PCSLCRi  = catx("|",  _PCSLYC_,  PCSLCRi) ;');
    call execute('               end;');
    call execute('               else do ;');
    call execute('                  _PCSLN_ = catx("|", "N"||strip(' ||strip(put(PCSSEQ, best.))|| '), _PCSLN_) ;');
    call execute('               end;');
    call execute('            end;');
    call execute('         end;');
    call execute('         if find(_PCSLY_, "Y", "i") = 0  and find(_PCSLN_, "N", "i") then PCSLFL = "N"; ') ;  /*N*/
    call execute('         else if find(_PCSLY_, "Y", "i") then PCSLFL = "Y"; ') ;  /*Y*/
    call execute('      end;');  /*Direc low*/

    call execute('      if "'||upcase(strip(DIREC))|| '" in ("H" "HIGH") then do;');  /*Direc*/
    call execute('         if AVAL > .Z then do ;') ;  /*AVAL*/
    call execute('            if '||strip(PRECOND) || ' then do ;') ;
```

```
    call execute('               if '||strip(BOOLEAN)|| ' then do ;');  /*algo   AVAL  */
    call execute('                  _PCSHQ_ = "Y"||strip(' ||strip(put(PCSSEQ, best.))|| ') ;');
    call execute('                  _PCSHY_ = catx("|", _PCSHQ_, _PCSHY_) ;');
    call execute('                  _PCSHYC_ = strip(_PCSHQ_) ||": "||strip("' ||strip(pcsdesc)|| '");') ;
    call execute('                  PCSHCRi  = catx("|",  _PCSHYC_,  PCSHCRi) ;');
    call execute('               end;');
    call execute('               else do ;');
    call execute('                  _PCSHN_ = catx("|", "N"||strip(' ||strip(put(PCSSEQ, best.))|| '), _PCSHN_) ;');
    call execute('               end;');
    call execute('            end;');
    call execute('         end;');
    call execute('         if find(_PCSHY_, "Y", "i") = 0  and find(_PCSHN_, "N", "i") then PCSHFL = "N"; ') ;  /*N*/
    call execute('         else if find(_PCSHY_, "Y", "i") then PCSHFL = "Y"; ') ;  /*Y*/
    call execute('      end;');  /*Direc High*/
    call execute('   end;');  /*CD*/
    if eof then call execute ('run;');
run;
```