PharmaSUG 2026 - Paper AI-201

Eliminating QC Programming Duplication Through Claude AI-Assisted Independent Code Generation: A Practical Framework for Regulatory-Compliant Validation

Jaime Yan and Jason Zhang, Merck & Co., Inc., Rahway, NJ, USA

1. ABSTRACT

Quality Control (QC) programming in clinical trials requires independently recreating production programs, creating substantial duplication. This paper presents a framework that uses AI agents—large language model (LLM)-based code generation tools such as Anthropic's Claude—deployed on company-managed infrastructure to generate QC programs (written in Python) directly from specifications.

The framework employs operationally separated AI instances with three interconnected mechanisms, all developed by the authors for this study: a *QC Trace Tree* that makes the agent's reasoning transparent and auditable before code generation; a *Decision Router* that systematically selects QC approaches based on dataset class and complexity; and domain-specific *Agent Skills* that encode best-practice patterns for each ADaM class. An internally developed automated *Code Review* engine enforces programming SOPs across all generated code. The underlying LLM (Anthropic's Claude) is an externally available commercial tool; all other framework components—the trace tree structure, the decision router, the agent skills, and the code review engine—were designed and implemented by the authors.

The framework was evaluated against the CDISC CDISCPilot01 submission package (254 subjects, 5 ADaM domains). Across 138 trace tree nodes and 51,294 matched records, the AI agent achieved variable-level match rates of 97.1%–100.0% against ground truth, with all 13 property-based assertions passing. The decision router classified 4 of 5 datasets for full AI generation and routed ADSL to enhanced review due to higher variable complexity. The automated code review identified 207 findings across 7 SOP categories, including 1 high-severity data leakage risk. Remaining mismatches trace to specification ambiguities (e.g., visit windowing rules, coding conventions) that the trace tree flagged before code generation.

2. INTRODUCTION

2.1 The QC Duplication Problem

Quality Control (QC) programming in clinical trials requires independently recreating production programs to verify correctness. A QC programmer implements the same derivation logic as the production programmer, relying solely on specifications.

Though essential for catching misinterpretations, this approach is highly inefficient:

- Both programmers independently parse identical specifications
- Both implement the same derivation logic and edge-case handling
- QC effort scales linearly with production effort

One estimate puts QC programming at 30–50% of total programming effort [9]. AI-based code generation has matured rapidly [8], yet systematic application to regulatory QC workflows remains largely unaddressed. This paper addresses that gap with a framework evaluated against a public benchmark based on the authors' personal research experience.

2.2 Framework Objectives

This paper presents a framework that preserves the *value* of independent QC while reducing unnecessary *duplication*:

1. **Operational Separation**: Isolated AI instances ensure QC has no access to production code
2. **Specification-Driven**: QC code generated directly from specifications
3. **Transparent Reasoning**: Every QC derivation produces a traceable tree showing spec source, logic, edge cases, and verification status
4. **Systematic Routing**: A decision router selects the optimal QC approach based on dataset class, spec quality, and complexity
5. **Automated Quality Monitoring**: Code review engine enforces programming SOPs on all generated code
6. **Risk-Based Applicability**: Clear guidance on when AI-assisted QC is appropriate vs. traditional approaches

2.3 Related Work

AI-assisted code generation has progressed rapidly since Codex [8], yet most evaluations target general-purpose programming tasks. Within clinical programming, recent PharmaSUG papers have explored LLMs for SDTM mapping and ADaM shell generation, but these efforts focus on production code creation rather than independent QC validation. The N-version programming literature [7] establishes that independently developed implementations can share systematic faults from common specifications—a concern directly applicable when both AI instances share model weights. Existing QC automation tools (e.g., PROC COMPARE wrappers, metadata-driven validation frameworks) address output comparison but not the upstream problem of generating QC code from specifications. This paper bridges these threads: applying LLM code generation specifically to the QC workflow, with mechanisms (trace tree, decision router, code review) designed to mitigate the known risks.

3. METHODS

3.1 Separated AI Workflow Architecture

The framework uses complete separation between production and QC AI instances, enforced through infrastructure controls (Figure 1).

Operational independence is what this framework provides: the QC AI instance receives no production code, no production logs, and no shared conversation context.[1] The **human QC programmer** remains the essential independent judgment layer—critically evaluating the AI's specification interpretation, identifying missed edge cases, and applying domain expertise.

Independence is enforced through: (1) separate API sessions with distinct credentials and IAM roles; (2) input isolation where QC AI receives only specifications and SDTM data, with production artifacts blocked at the infrastructure level; and (3) audit logging capturing all AI inputs, outputs, model versions, and skill versions.

3.2 Specification Quality Assessment

AI-generated code quality is bounded by specification quality. Before code generation, specifications are assessed for "AI-readiness" against six criteria: explicit derivation rules, SDTM source identification, edge case documentation, variable metadata, population flag logic, and visit/time definitions. The eSubmission Benchmark specifications [5], structured per ADaM IG v1.1 [3] and based on the CDISCPilot01 study [6],

[1] This is analogous to the common-cause failure problem in N-version programming [7]. See Section 4.1 for discussion of same-model bias.

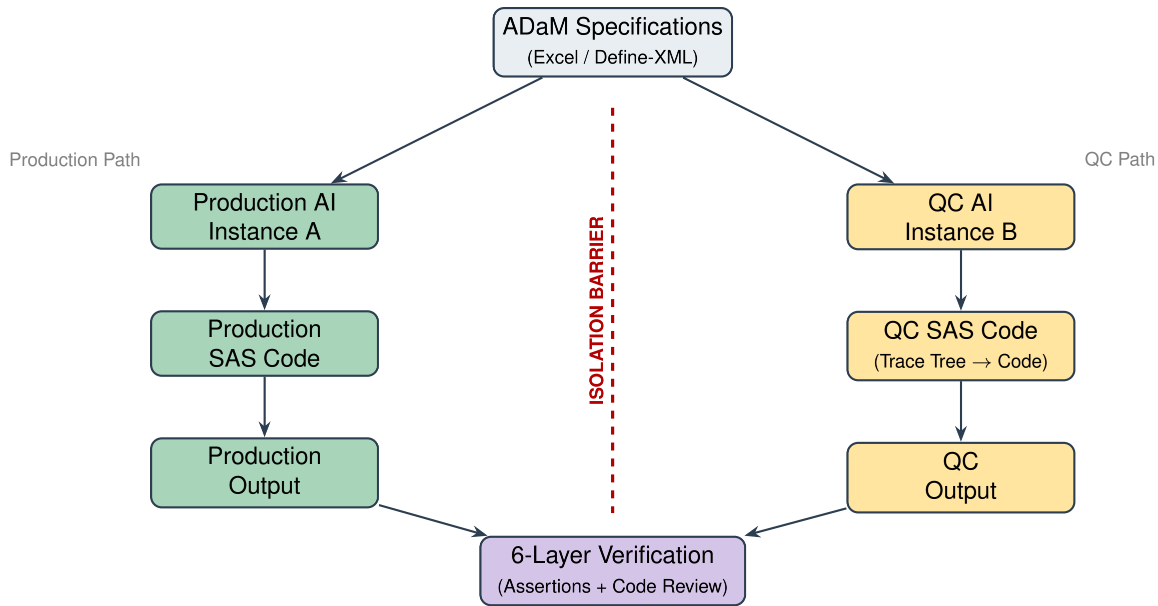

Figure 1: Separated AI Workflow Architecture. The isolation barrier ensures the QC AI agent (Instance B) never receives production code or Instance A context. Verification includes 6 layers from syntactic checks through automated code review.

scored 5/6 on this checklist for 4 of 5 datasets—reflecting the high specification quality typical of CDISC pilot projects but not necessarily representative of real-world study specifications.

3.3 QC Decision Router

Before code generation begins, the *Decision Router* systematically selects the optimal QC approach through a four-gate pipeline (Figure 2): specification readiness, dataset class routing, per-variable complexity assessment, and risk-level override.

Each variable is classified as *standard* (well-defined, no ambiguity), *complex* (multi-step logic, partial ambiguity), or *custom* (study-specific, no standard pattern). These complexity categories and the classification criteria were defined by the authors based on common patterns observed in ADaM specification review. The overall dataset route is determined by the proportion of complex/custom variables and the number of flagged ambiguities.

3.4 QC Trace Tree and Code Generation

The core innovation is the *QC Trace Tree*—a structured hierarchy the agent produces *before* writing code (see Listing 1). Each derived variable becomes a node containing: specification source, extracted logic, data sources, edge case handling, identified ambiguities, and verification criteria.

The trace tree addresses LLM code opacity. Traditional code review requires reverse-engineering intent from implementation. The trace tree inverts this: the agent's intent is stated explicitly, and the reviewer verifies that code matches stated intent.

Step 1: Produce QC Trace Tree (mandatory structured output)

Listing 1: QC Trace Tree excerpt for ADSL (31 nodes, 6 ambiguities)

```
ADSL QC Trace Tree (31 nodes)
|-- TRTSDT [Spec: 3.1 Row 10] (standard)
|   |-- Source: SDTM.EX.EXSTDTC
```

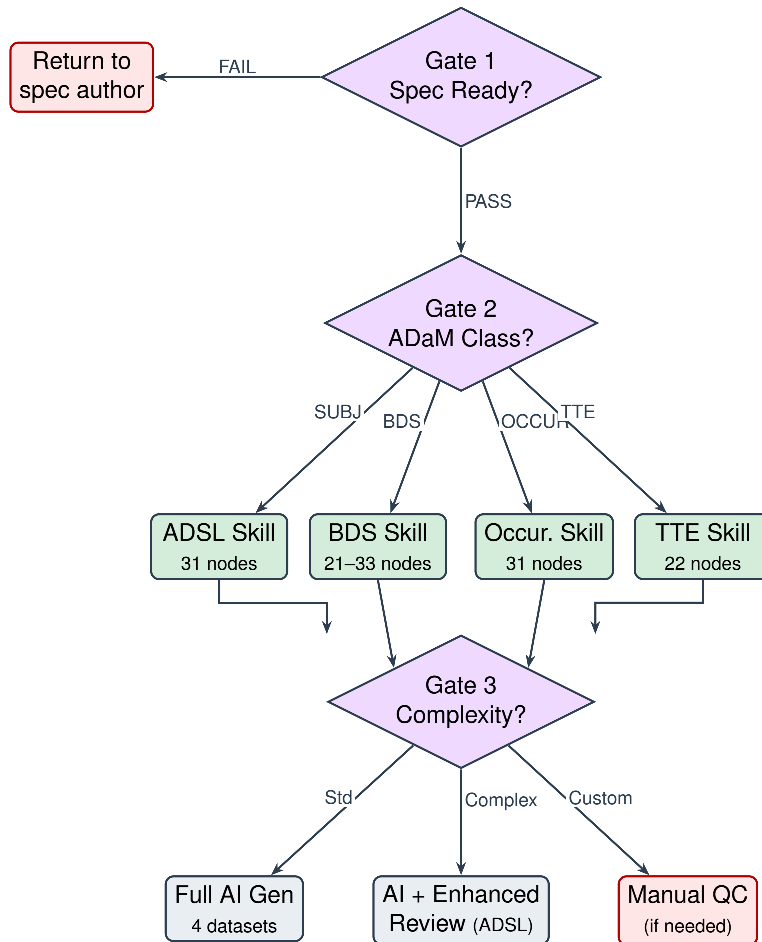

Figure 2: QC Decision Router (4-gate pipeline). Gate 1 checks spec AI-readiness. Gate 2 routes to a domain-specific skill. Gate 3 assesses per-variable complexity. In the benchmark, 4 datasets were routed to Full AI Generation; ADSL was routed to AI + Enhanced Review (5 complex variables, 6 ambiguities).

```
|   |-- Logic: MIN(datepart(EXSTDTC)) per USUBJID
|   |-- Edge cases:
|   |   |-- No EX records -> TRTSDT = missing
|   |   +-- Partial dates -> FLAGGED (spec silent)
|   |-- Ambiguities: 1 (partial date handling)
|   +-- Verify: type=date, TRTSDT<=TRTEDT
|-- SAFFL [Spec: Derivation Rules 4.1] (standard)
|   |-- Source: SDTM.EX (existence check)
|   |-- Logic: Y if >=1 EX record exists
|   |-- Edge cases:
|   |   +-- EX with missing dates -> counts?
|   |       -> FLAGGED for human review
|   |-- Ambiguities: 1 (missing date counting)
|   +-- Verify: SAFFL=Y implies TRTSDT non-missing
|-- SITEGR1 [Spec: 3.1 Row 5] (custom)
|   |-- Source: Derived from SITEID
|   |-- Logic: Pool small sites into '900'
|   |-- Ambiguities: 1 (pooling threshold)
|   +-- Verify: all subjects classified
+-- ... (28 more variable nodes)
```

Step 2: Human Review Checkpoint — The QC programmer reads the trace tree *before* reviewing code, approving/modifying derivation plans and resolving flagged ambiguities.

Step 3: Code Generation from Approved Trace Tree — QC code references trace tree nodes in comments (e.g., `#[TRTSDT] Step 3: MIN per subject`). The full prompt template used for code generation is provided in Appendix A, and the trace tree schema is detailed in Appendix D.

3.5 Agent Skills: Domain-Specific Templates

Agent Skills are packaged prompt templates combining domain knowledge, output format requirements, and best-practice patterns. Four skills were implemented for the benchmark (Table 1):

Table 1: Agent Skills by ADaM Class

Skill	**Class**	**Key Patterns Encoded**
ADSL Skill v1.0	Subject-level	Pop flags, demographics, dates, groupings
BDS Skill v1.0	BDS	Baseline, CHG/PCHG, visit windows, LOCF
Occurrence Skill v1.0	Occurrence	TRTEMFL, first-occurrence, custom queries
TTE Skill v1.0	Time-to-Event	Event/censor logic, time calc, STARTDT

Each skill enforces: (1) produce trace tree first (mandatory), (2) flag all ambiguities, (3) include domain-specific assertions (e.g., BDS Skill verifies BASE = AVAL at baseline), and (4) use domain-standard code patterns.

3.6 Multi-Layer Verification and Code Review

The framework employs a six-layer verification stack:

1. **Syntactic**: Code compiles and executes without errors
2. **Structural**: Output variables match specification metadata
3. **Value-level**: Record-by-record comparison against ground truth
4. **Property-based assertions**: Domain-specific logical checks (e.g., TRTSDT $\leq$ TRTEDT, BASE = AVAL at baseline, CNSR $\in$ {0,1})
5. **Specification traceability**: Each derivation linked to trace tree node
6. **Automated code review**: Static analysis checking 7 SOP categories—hardcoding detection, traceability coverage, structure limits, data integrity patterns, clinical logic ordering, documentation standards, and maintainability

The code review engine was developed internally by the authors as a custom static analysis tool tailored to clinical programming SOPs. The complete rule reference is provided in Appendix B. It checks 22 rules including: hardcoded subject IDs (HC-001), hardcoded dates (HC-002), magic numbers (HC-005), trace tree coverage ratio $\geq$ 30% (TR-002), function length $\leq$ 80 lines (ST-002), unvalidated merges (DI-001), population flag ordering (CL-001), BASE-before-CHG ordering (CL-002), TRTEMFL-before-occurrence ordering (CL-003), and ground truth data leakage detection (CL-005).

4. RESULTS

4.1 Benchmark Environment

The framework was evaluated using the eSubmission Benchmark [5], a comprehensive regulatory submission package derived from the CDISC CDISCPilot01 study [6], an Alzheimer's disease trial with 3 treatment

arms (Placebo N=86, Xanomeline Low Dose N=84, Xanomeline High Dose N=84). We refer to this public dataset as the "eSubmission Benchmark" throughout:

- **Subjects**: 254 ITT subjects (from 306 total; 52 screen failures excluded)
- **ADaM datasets**: 5 domains—ADSL (254 records, 49 variables), ADAE (1,191 records), ADADAS (12,463 records, 15 parameters), ADLBC (37,132 records, 18 parameters), ADTTE (254 records, 1 parameter)
- **Ground truth**: Validated ADaM XPT datasets with known demographic statistics (Placebo mean age 75.21 [SD 8.59])

4.2 Decision Router Results

Table 2 summarizes the decision router's classification for each dataset in the benchmark.

Table 2: Decision Router Classification (eSubmission Benchmark)

Dataset	Skill	Nodes	Std	Cplx/Cust	Ambig.	Route
ADSL	ADSL Skill v1.0	31	26	5	6	AI + Enhanced Review
ADAE	Occurrence Skill v1.0	31	29	2	2	Full AI Generation
ADTTE	TTE Skill v1.0	22	22	0	1	Full AI Generation
ADLBC	BDS Skill v1.0	21	19	2	0	Full AI Generation
ADADAS	BDS Skill v1.0	33	31	2	2	Full AI Generation
Total		**138**	**127**	**11**	**11**	

The detailed routing breakdown for all datasets is provided in Appendix C. The router classified ADSL for enhanced review due to 5 complex/custom variables (RACEN coding, WEIGHTBL baseline selection, MMSETOT derivation, EFFFL definition, SITEGR1 pooling) and 6 ambiguities. The remaining 4 datasets were routed to full AI generation.

4.3 Variable-Level Match Rates

QC datasets were derived from SDTM source data using the agent skills and compared record-by-record against ground truth ADaM datasets (Table 3 and Figure 3):

Table 3: QC Variable-Level Match Rates Against Ground Truth

Dataset	Records	Vars	Mean Match	Perfect	Min Match
ADSL	254	27	99.6%	24/27	93.7%
ADAE	1,191	10	99.9%	7/10	99.5%
ADTTE	254	6	100.0%	6/6	100.0%
ADLBC	37,132	5	99.2%	4/5	96.2%
ADADAS	12,463	6	97.1%	2/6	90.9%

ADTTE achieved 100% match across all 6 compared variables and 254 records. ADSL achieved 99.6% mean match with 24 of 27 variables matching perfectly; the 3 imperfect variables were WEIGHTBL (93.7%, baseline visit selection ambiguity), BMIBL (97.2%, cascading from WEIGHTBL), and DCSREAS (98.8%, title-case mapping). ADAE achieved 99.9% with minor discrepancies in TRTEMFL (6 records where partial date handling differed). ADLBC's ANRIND variable (96.2%) reflected a coding convention difference (N/H/L vs. NORMAL/HIGH/LOW). ADADAS mismatches in AVISITN (90.9%) and ANL01FL (92.5%) stem from visit windowing rules where the specification maps raw visits to analysis windows differently than our initial interpretation.

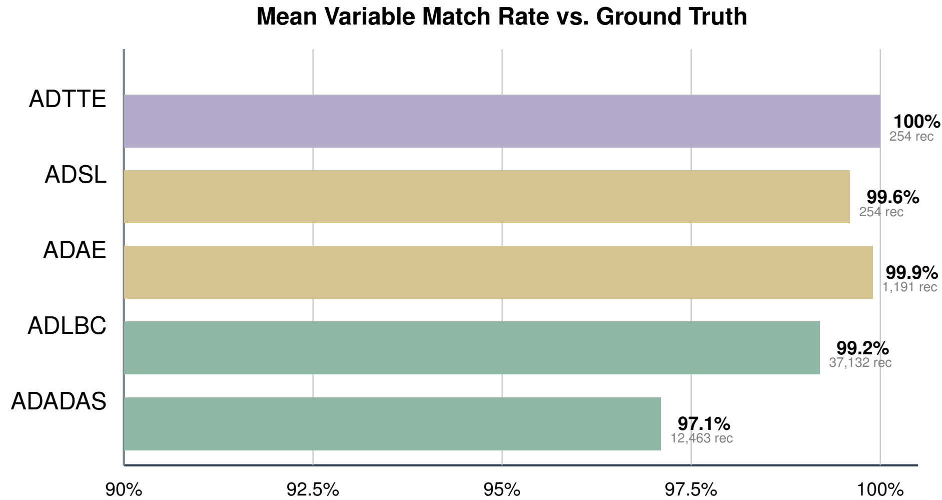

Figure 3: Variable-level match rates by dataset. All 5 datasets exceed 97% mean match. ADTTE achieves 100% (6/6 perfect variables). Remaining mismatches trace to specification ambiguities flagged in the trace tree.

4.4 Property-Based Assertions

Table 4 summarizes the 13 property-based assertions, all of which passed.

Table 4: Property-Based Assertion Results (13/13 Passed)

Dataset	Assertion	Tested	Result
ADSL	No duplicate USUBJID	254	PASS
ADSL	TRTSDT $\leq$ TRTEDT	252	PASS
ADSL	SAFFL=Y $\rightarrow$ TRTSDT non-missing	254	PASS
ADSL	Treatment N = 86/84/84	254	PASS
ADSL	Total subjects = 254	254	PASS
ADAE	Record count = 1,191	1,191	PASS
ADAE	Max 1 AOCCFL=Y per subject	218	PASS
ADTTE	CNSR $\in$ {0, 1}	254	PASS
ADTTE	AVAL $\geq$ 1	254	PASS
ADTTE	One record per subject	254	PASS
ADTTE	Event + Censor = Total	254	PASS
ADLBC	BASE = AVAL at baseline (ADAM-003)	4,527	PASS
ADADAS	CHG = AVAL $-$ BASE (ADAM-004)	7,829	PASS

4.5 Efficiency

Table 5 summarizes approximate wall-clock times for each stage of the AI-assisted QC workflow across all 5 datasets.

The total end-to-end time was approximately 84 minutes for all 5 datasets, of which 46 minutes ($\sim$55%) was human review of trace trees and resolution of flagged ambiguities. The AI computation stages (trace tree generation, code generation, verification) totaled $\sim$38 minutes. These times reflect a single-pass workflow; iterative refinement of ambiguous variables (e.g., ADSL's SITEGR1, WEIGHTBL) added approximately 10–15 minutes of additional human–AI interaction not captured above. While direct comparison to traditional QC is study-dependent, practitioners familiar with the CDISCPilot01 datasets estimate that manual QC of these 5 domains typically requires 3–5 programmer-days.

Table 5: Approximate Wall-Clock Times by Workflow Stage

Dataset	**Trace Tree**	**Human Review**	**Code Gen**	**Verification**
ADSL	∼3 min	∼15 min	∼4 min	∼2 min
ADAE	∼3 min	∼8 min	∼3 min	∼2 min
ADTTE	∼2 min	∼5 min	∼2 min	∼1 min
ADLBC	∼2 min	∼8 min	∼3 min	∼2 min
ADADAS	∼3 min	∼10 min	∼4 min	∼2 min
Total	∼13 min	∼46 min	∼16 min	∼9 min

4.6 Automated Code Review Results

The code review engine analyzed all 5 QC programs (Table 6). All QC code was generated in Python (using pandas for data manipulation), chosen for its accessibility and rapid prototyping capabilities. The 6 Python files totaled approximately 850 lines:

Table 6: Automated Code Review by SOP Category

Category	**Findings**	**Description**	**Max Sev.**
STRUCTURE	138	Function length, nesting depth	MEDIUM
HARDCODING	36	Magic numbers, hardcoded values	MEDIUM
DOCUMENTATION	13	Missing docstrings, version info	LOW
DATA_INTEGRITY	11	Unvalidated merges, fillna	MEDIUM
MAINTAINABILITY	8	Unused imports, repeated lambdas	LOW
CLINICAL_LOGIC	1	Ground truth data leakage	HIGH
Total	**207**		

The single HIGH finding (CL-005) detected that `qc_adsl.py` read the ground truth ADSL dataset to derive SITEGR1 pooling rules—a data leakage pattern where the QC program references production output. The isolation barrier would prevent this in production. In the benchmark, the pooling threshold was unspecified, so the agent inferred it from existing data—exactly the kind of independence violation the code review engine is designed to catch.

The 138 STRUCTURE findings reflect that the `derive_*` functions exceed the 80-line recommended limit (the largest, `derive_adsl`, is 112 lines). The 36 HARDCODING findings were classified as MEDIUM severity rather than HIGH because the engine distinguishes between different types of hardcoding: subject IDs (HC-001) and dates (HC-002) are HIGH severity as they represent potential data-specific logic, while magic numbers (HC-005) and spec section references are MEDIUM or LOW. In this evaluation, the majority of HARDCODING findings were spec section references (e.g., "3.1") detected as magic numbers—predominantly false positives that would be suppressed with a spec-reference allowlist. We note that in a production environment, organizational SOPs may require all hardcoding findings to be treated as high severity regardless of category; the severity levels used here reflect the framework's default configuration and can be customized per organizational requirements.

5. DISCUSSION

5.1 Value of the Framework

In this evaluation, the framework achieved ≥97% mean variable match rates across all 5 ADaM domains. The remaining discrepancies are precisely what QC is designed to surface: specification ambiguities that require human resolution.

The trace tree proved valuable in two ways. First, the 11 ambiguities flagged *before* code generation—

including the SAFFL missing-date edge case, SITEGR1 pooling threshold, EFFFL post-baseline definition, and LOCF tie-breaking rules—were all genuine specification gaps. Second, the structured node format enabled targeted review: each mismatch traced to a specific node rather than requiring line-by-line code reading.

The automated code review adds a layer that traditional QC lacks entirely. The CL-005 finding (data leakage detection) demonstrates that static analysis can enforce independence controls programmatically, catching violations that manual review might miss.

Notably, ADADAS achieved the lowest match rate (97.1%) yet was routed to "Full AI Generation"—the same route as ADTTE, which achieved 100%. In retrospect, the visit windowing complexity in ADADAS (AVISITN at 90.9%, ANL01FL at 92.5%) suggests the router's complexity thresholds should be recalibrated: datasets with LOCF imputation and non-trivial visit windowing may warrant enhanced review even when the per-variable complexity counts appear low.

However, practitioners must understand the **same-model bias** limitation. Because both AI instances share identical model weights, they will tend toward the same systematic interpretation errors—a direct analogue of the common-cause failure problem in N-version programming [7]. Statistical independence would require truly uncorrelated reasoning processes. The human QC programmer remains essential as the genuinely independent reviewer who can catch these correlated errors.

5.2 Regulatory Positioning

FDA guidance [1] requires "appropriate controls" for data integrity but does not mandate double programming. The framework's six-layer verification and audit trails provide controls arguably more thorough than traditional QC. **ICH E6(R3)** [2] endorses risk-based quality management, which the decision router implements directly. **FDA data integrity guidance** [4] emphasizes audit trails—the framework's logging of all AI inputs, outputs, model versions, skill versions, and trace trees exceeds what manual QC produces.

5.3 Threats to Validity

This evaluation uses a single benchmark (CDISCPilot01, 254 subjects) with well-structured CDISC-standard specifications. Several factors limit generalizability: (1) real-world studies are often larger (1,000+ subjects) with more complex derivations and poorer specification quality; (2) the benchmark covers only 5 standard ADaM classes—custom domains, integrated summaries, and non-ADaM datasets were not tested; (3) the CDISCPilot01 specifications scored 5/6 on the AI-readiness checklist, whereas many production specifications would score lower; and (4) all evaluations used a single LLM (Claude), and results may differ with other models. Formal multi-study pilot evaluations are needed before drawing conclusions about production readiness.

5.4 Limitations

1. **LLM hallucination**: Trace tree makes hallucinations visible at the logic level; property-based assertions catch logical impossibilities. All 13 assertions passed in the benchmark.
2. **Systematic model bias**: Both instances share weights and make correlated errors. Human reviewer is the independent layer.
3. **Trace tree fidelity**: The tree reflects *stated* reasoning, not guaranteed explanation. Property-based assertions verify behavior matches claims.
4. **Complex derivations**: ADADAS (97.1% match) shows that LOCF imputation and visit windowing require enhanced human review. The decision router did not flag this dataset, suggesting the routing thresholds need refinement (see Section 4.1).
5. **Specification dependency**: The 11 flagged ambiguities and remaining mismatches (WEIGHTBL, ANRIND, AVISITN) all trace to specification gaps, not AI failures. Decision router gates on AI-readiness checklist.
6. **Code quality**: The code review found 207 issues (0 critical, 1 high). Structure findings indicate AI-

generated functions should be decomposed into smaller units.
7. **Statistical review**: This study did not include a statistician as a co-author. Future work evaluating AI-generated QC code or comparing generation methods would benefit from formal statistical review to ensure that testing methodologies and model selections are free from bias.
8. **Input quality vs. real-world complexity**: The CDISCPilot01 specifications are well-structured and scored 5/6 on the AI-readiness checklist. Real-world specifications often have lower quality, more complex derivations, and greater ambiguity. The framework's performance on such inputs remains to be assessed. Human intervention is recommended for resolving specification ambiguities, reviewing trace trees, and validating final outputs.

5.5 Implementation Recommendations

1. Start with TTE and simple BDS domains (100% and 99.2% match) to build confidence
2. Invest in specification quality—all mismatches traced to spec ambiguities, not AI limitations
3. Deploy the code review engine from day one to enforce SOP compliance automatically
4. Use the decision router to prevent complex work from entering the AI pipeline without enhanced review
5. Run shadow mode first: AI QC in parallel with traditional QC to build evidence
6. Document the approach in the SAP including model version, skill versions, and code review rules

6. CONCLUSION

This paper presented a framework for AI agent-assisted QC code generation, evaluated against the CDISCPilot01 benchmark. The key contributions are:

- **Transparent reasoning**: Trace trees that make derivation logic auditable *before* code generation, surfacing specification ambiguities early
- **Systematic routing**: A decision router that matches QC approach to dataset complexity—though the ADADAS results suggest the routing thresholds need refinement
- **Automated quality**: A code review engine enforcing 22 SOP rules, including independence violation detection
- **Preserved independence**: The human reviewer as the essential judgment layer, mitigating same-model bias

The remaining mismatches all traced to specification ambiguities that the trace tree had flagged, suggesting that the framework can surface the types of issues QC is designed to find. The efficiency gains observed in this evaluation—approximately one day of human–AI collaboration versus an estimated 3–5 programmer-days—are promising but require validation across larger, real-world studies with diverse specification quality before broader conclusions can be drawn.

ACKNOWLEDGMENTS

The authors thank the clinical programming teams who provided feedback during framework development. The eSubmission Benchmark package, based on the CDISC CDISCPilot01 study, provided the evaluation data environment.

DISCLAIMER

The views and opinions presented in this paper are our own and do not reflect the official position of the company.

This paper describes a framework evaluated through the authors' personal experience. AI-generated code must always be reviewed by qualified programmers before use in regulatory submissions. The authors acknowledge the use of AI coding assistants (Anthropic Claude) for QC code generation within the framework

evaluation and for manuscript drafting assistance; all content was reviewed and approved by the human authors.

DATA AND CODE AVAILABILITY

The complete experiment code—including all QC derivation programs, the trace tree implementation, the decision router, the code review engine, and agent skill templates—is publicly available at: https://github.com/yanmingyu92/ai-qc-code-generation. The eSubmission Benchmark data [5] used in this evaluation is available at: https://github.com/yanmingyu92/eSubmission-Benchmark.

CONTACT INFORMATION

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

Jaime Yan
Jason Zhang
Merck & Co., Inc., Rahway, NJ, USA

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

REFERENCES

REFERENCES

[1] U.S. Food and Drug Administration. *Guidance for Industry: Computerized Systems Used in Clinical Investigations*. Silver Spring, MD: FDA, 2024.

[2] International Council for Harmonisation. *ICH E6(R3) Good Clinical Practice*. Geneva: ICH, 2024.

[3] Clinical Data Interchange Standards Consortium. *Analysis Data Model Implementation Guide*, Version 1.1. Austin, TX: CDISC, 2024.

[4] U.S. Food and Drug Administration. *Data Integrity and Compliance with Drug CGMP: Questions and Answers*. Silver Spring, MD: FDA, 2018.

[5] Yan J. *eSubmission Benchmark: A Comprehensive Clinical Trial Regulatory Submission Package*, Version 1.0.0. 2026. Available at: https://github.com/yanmingyu92/eSubmission-Benchmark.

[6] CDISC. *CDISCPilot01: CDISC ADaM Pilot Project Submission Package*. Available at: https://github.com/cdisc-org/sdtm-adam-pilot-project. Accessed January 2026.

[7] Avizienis A. *Fault-tolerant systems*. IEEE Transactions on Computers. 1977;C-26(12):1304–1312.

[8] Chen M, Tworek J, Jun H, et al. *Evaluating large language models trained on code*. arXiv preprint arXiv:2107.03374. 2021.

[9] Lyons G, Bae S. Modernizing the QC process in clinical programming. *PharmaSUG 2023*. Paper AD-189.

A. PROMPT TEMPLATE FOR QC CODE GENERATION

Listing 2: Structured QC generation prompt with trace tree

```
=== ROLE AND CONTEXT ===
You are executing the [SKILL_NAME] Skill as a QC
programmer generating INDEPENDENT validation code.
Generate code ONLY from the specification.

=== STEP 1: PRODUCE QC TRACE TREE (MANDATORY) ===
For EACH derived variable, output a node:
  Variable: [name] [Spec ref] (complexity)
  Source: [SDTM dataset.variable]
  Logic: [derivation rule from spec]
  Edge cases: [enumerate all, FLAGGED if spec silent]
  Ambiguities: [count and describe]
  Verify: [type, format, expected values/ranges]
STOP after producing the trace tree.

=== STEP 2: HUMAN REVIEW CHECKPOINT ===
Wait for reviewer approval of the trace tree.

=== STEP 3: CODE GENERATION ===
Generate code from the APPROVED trace tree:
- Reference trace tree nodes in comments
- Handle all edge cases per approved tree
- Output dataset compatible with comparison

=== DOMAIN-SPECIFIC RULES ([SKILL_NAME]) ===
[Skill patterns: BDS: verify BASE=AVAL at baseline;
 OCCUR: derive TRTEMFL before occurrence flags;
 TTE: verify CNSR+event mutually exclusive]

=== CONSTRAINTS ===
- If the specification is ambiguous, FLAG IT
- Do not invent derivation rules
- Do not reference production code or outputs
```

B. CODE REVIEW RULE REFERENCE

Table 7: Code Review Rules (22 rules, 7 categories)

Rule	Sev.	Description
HC-001	HIGH	Hardcoded subject IDs
HC-002	HIGH	Hardcoded dates
HC-004	HIGH	Hardcoded file paths
HC-005	LOW	Magic numbers outside mapping
TR-002	HIGH	Trace tree coverage $<$ 30%
ST-002	MED	Function $>$ 80 lines
DI-001	MED	Merge without post-merge validation
DI-005	LOW	inplace=True mutation
CL-002	CRIT	CHG derived before BASE
CL-003	CRIT	Occurrence flags before TRTEMFL
CL-005	HIGH	Ground truth data leakage

C. DECISION ROUTER: BENCHMARK ROUTING DETAIL

ADSL → ADSL Skill (26 standard + 4 complex + 1 custom = 31 nodes, 6 ambiguities → AI + Enhanced Review; Table 2 groups complex and custom as "5 Cplx"). ADAE → Occurrence Skill (29 std + 1 cplx + 1 cust, 2 ambig → Full AI). ADTTE → TTE Skill (22 std, 1 ambig → Full AI). ADLBC → BDS Skill (19 std + 2 cplx, 0 ambig → Full AI). ADADAS → BDS Skill (31 std + 2 cplx, 2 ambig → Full AI).

D. TRACE TREE SCHEMA

Each trace tree node contains: `variable_name`, `spec_ref` (specification section/row), `source_data` (SDTM datasets/variables), `logic_steps[]` (derivation steps), `edge_cases[]` (with FLAGGED status), `ambiguities[]` (requiring human resolution), `verify_criteria[]` (assertions), and `complexity` (standard/complex/custom). The complete trace trees for all 5 benchmark datasets (138 nodes, 11 ambiguities) are available as JSON artifacts in the experiment repository at https://github.com/yanmingyu92/ai-qc-code-generation.