Paper ML31

Batch Cleaning of SAS Programs for Submission Readiness: An ML and Python Hybrid Approach

Qisen Luan, BioPier LLC, a Veramed Company, Burlington, United States
Qiong Wei, BioPier LLC, a Veramed Company, Burlington, United States
Lixin Gao, BioPier LLC, a Veramed Company, Burlington, United States

ABSTRACT

Preparing SAS programs to meet regulatory standards (e.g., clarity, conciseness and no irrelevant content, etc.) is critical in clinical trial submissions but often requires extensive manual effort. This paper presents a hybrid ML and Python-based approach for batch cleaning SAS programs to enhance code quality and submission readiness. The workflow combines regex-based comment extraction, alternative strategies for comment classification (from simple heuristics to ML prediction and hybrid ML + human review), Python-based batch cleanup with tracking logs, and post-cleanup validation by re-running programs and comparing outputs against the originals. This ML-assisted and human-in-the-loop approach offers a scalable solution for statistical programmers seeking to efficiently prepare high quality, submission ready SAS programs in clinical trial analysis.

INTRODUCTION

Comments in clinical programming are essential for documenting assumptions, analysis decisions, and data issues. Over time, however, programs accumulate large blocks of narrative comments, debugging notes, and commented-out code that can hinder readability, confuse reviewers, and slow decision-making. Submission-ready programs should be clean, clear, and concise. Traditional manual cleanup depends on individual style, SOP interpretations, and timelines, making it time-consuming, inconsistent, and hard to audit. Our goal is to automate where appropriate, and involve humans where necessary.

END-TO-END WORKFLOW

The workflow has four steps. Each step is modular and has clear documents saved for potential auditing activities.

Step 1. REGEX-BASED COMMENT EXTRACTION

SAS language supports multiple comment styles [4, 5]. Depending on clinical statistical programmer's preferences and programming styles, we commonly see the following comment types in practice:

- Block comments: /* ... */
- Starblock (line-prefixed): lines beginning with * up to ;
- Macro line comments: %* ... ;
- COMMENT statements: comment ... ;

In this step, a Python script (e.g. listcommentallfolder.py) is used to recursively identify all SAS programs (.sas files) and extract SAS comments using a regex-based python approach [1, 2, 3]. Table 1 summarizes the extraction patterns. Block comments are extracted across multiple lines; starblock comments are extracted as contiguous groups of *-prefixed lines; and inline comments are extracted as single-line statements ending with ;. We also merge adjacent block comments separated only by whitespace to preserve logical units. Identified comments are written to a CSV file with program path and name, comment type, comment text and position metadata, context snippet (+/-2 surrounding lines before/after comment text).

Table 1. Regex-Based SAS Comment Pattern

Comment type	Regex pattern	Description
Block comment /* ... */	r'/*.*?*/' with DOTALL	Extracts all text between /* and */, including multi-line blocks.
Starblock (* at line start)	r'^\s**'	Identifies lines starting with optional whitespace followed by *; contiguous runs are grouped into a starblock (minimum length constraint applied to avoid duplicating single-line comments).

Comment type	Regex pattern	Description
Inline comment (*...; or %*...;)	r'(?i)(?:(?<=;)|^)\s*%?*\s*[^;]*;'	Extracts single-line comments that begin with optional whitespace, optional %, then *, then text up to ;.
Comment keyword (comment …;)	r'(?i)(?:(?<=;)|^)\s*comment\b(?!\s*=)\s*[^;]*;'	Extracts 'comment' comments.

Step 2. COMMENT CLASSIFICATION

Once the comments are extracted, an experienced programmer can be designed to review the CSV file and confirm the accuracy of comment extraction. The minimal rule is excluding any false identified comments, the removal of which will change the programming logic and/or even break the program. In the meantime, the programmer will give a preliminary justification regarding the usefulness of those comments and evaluate different strategies to decide whether to keep vs. remove comments.

Approach A. Header-only Keep
This is the simplest approach to implementation. In this approach, we treat the initial block/starblock within the first k lines (e.g., first 15 non-noise lines) that contains certain required header keywords such as program name, author name, date, etc.) as the program header and classify as "keep", For all other extracted comments, we classify as "remove".

Pros: fast, deterministic, submission oriented.
Cons: may remove valuable narrative comments deeper in the file.

Approach B. Pre-Defined Rules (Heuristics)
Extended from approach A, we can manually define a more complex labeling rule sets based on strong signals (headers, separators, TODO/QC/Changing Log, decisions, etc.) and code-like features (SAS keywords, macro calls, etc.) to mark each comment as "keep", "remove", or "review". Table 2 shows an example of the pre-defined labeling rules.

Table 2. Pre-defined Comment Labeling Rules

Rule ID	Condition (evaluated in order)	Decision	Typical example	Implementation (summary)
R1	Comment is empty after .strip()	Remove	"", " "	if not comment_raw.strip():
R2	Program header block (contains all required header keywords)	Keep	Standard header block	is_program_header(); header blocks excluded from extracted set
R3	After removing SAS comment delimiters (/* */, leading *, %*, trailing ;) the remaining content is empty	Remove	/* */, /* */	strip_comment_delimiters() then is_effectively_empty()
R4	Decoration-only separators (all non-empty lines are only *, -, =)	Remove	**********, -----, =====	is_decoration_block() requiring **all** lines match
R5	Whitelist markers present	Review	TODO, FIXME, REVISION HISTORY, QC, CHANGE, UPDATE	WHITELIST_REGEX.search(text)
R6	Macro invocation statement (including multi-line %mymacro(...);)	Remove	%predata(in=dm); or multi-line %mpgbreak2(...);	MACRO_CALL_ANYWHERE_REGEX with DOTALL + non-greedy
R7	if ... then ... ; appears (DATA-step conditional statement)	Remove	if cmongo1=1 then pedate='Ongoing';	IF_THEN_STMT_REGEX (requires semicolon)
R8	No SAS "code indicators" and no SAS keyword match	Review	Pure narrative text	Check absence of ; = ( ) % & : and SAS_KEYWORD_REGEX
R9	SAS structural keywords present	Remove	proc, data, run;, %macro, proc sql	SAS_KEYWORD_REGEX.search(text)
R10	Macro-language constructs / macro variables in code-like contexts	Remove	%macro, %if, %let, macro calls with &	Explicit %macro/%if/%let patterns; conservative handling for &
R11	Issue/number reference pattern present	Review	#100, #999	ISSUE_REF_REGEX.search(text) evaluated early

Rule ID	Condition (evaluated in order)	Decision	Typical example	Implementation (summary)
R12	Statement-like signals: dataset refs (lib.ds), assignments, common SAS verbs, run;/quit;, unbalanced parentheses	Remove	work.adsl, x=y;, set adsl;, unmatched (	DATASET_LIKE_REGEX, ASSIGNMENT_REGEX, common verbs, RUN_QUIT_REGEX, paren-balance
R13	PROC REPORT define statement with slash options	Remove	define pdatemed / order style(...);	DEFINE_SLASH_REGEX.search(text)

Each rule has a rule ID and the classifier applies rules in a fixed order. This precedence is essential. For example, the known "keep" cases (program headers, decoration-only separators, whitelisted markers, issue references) are evaluated early to minimize false positives, while high-signal code patterns (macro invocations, if...then...;, PROC REPORT define ... /) are evaluated before broader keyword and syntax heuristics. Decisions are written to an updated CSV file with decision, reason (rule ID) appended to the CSV file generated in step 1.

Pros: transparent, explainable, easy to refine.
Cons: can miss edge cases, require maintenance as coding styles can vary significantly among programmers, vendors.

Approach C: Supervised ML Prediction
In this approach, we train a supervised text classifier to predict whether an extracted comment should be kept (useful) or removed (not useful). Compared with pre-defined rules, a supervised model can adapt to diverse programming styles and reduces manual review by learning patterns directly from labeled examples.
Training data and labels: We maintain a labeled comment dataset in CSV format (e.g., {comment_text, keep? (Y/N)}) curated from prior projects and periodically expanded with newly reviewed comments. Labels are assigned by experienced programmers using a consistent guideline (e.g., "keep" for comments documenting key assumptions, analysis decisions, data issues, or maintainability notes; "remove" for decorative separators, debugging remnants, and commented-out code). To control the primary operational risk (incorrectly removing a truly useful comment), the downstream decision is implemented as a triage into KEEP / REMOVE / REVIEW rather than a pure binary outcome.
Features: Each comment is converted to numerical features using (a) TF-IDF character n-grams (e.g., 3–5) to capture punctuation and "code-ness" signals (such as ;, %, &, run, proc), and (b) TF-IDF word/token n-grams (e.g., 1–2 or 1–3) to capture short phrases commonly associated with useful narrative (e.g., "purpose", "assumption", "QC", "derive"). Optionally, dense sentence embeddings can be concatenated with TF-IDF to capture semantic similarity for longer narrative comments. When available, lightweight metadata (comment type, file position, surrounding code snippet) can be included to improve robustness for short or ambiguous comments.
Model training and validation: We train a linear classifier suitable for high-dimensional sparse text features, such as Logistic Regression (with class weights to handle imbalance) or Linear SVM. Performance is assessed using stratified cross-validation. Because downstream workflow uses probability thresholds for triage, we calibrate the model output to obtain an estimated P(useful) (e.g., via Platt scaling or isotonic regression).
Practical considerations: Supervised ML performance depends on the diversity and consistency of labeled data. Short comments, mixed blocks (narrative + commented-out code), and organization-specific jargon can be challenging; these are precisely the cases that benefit from the REVIEW path and continuous learning. To support auditability, we recommend logging model version, training data version, thresholds, and inference timestamp together with each decision, and periodically monitoring review rate and error patterns to detect style drift across studies or vendors.

Pros: Quick to execute, reduce overall manual review load.
Cons: short comments and mixed blocks are challenging; requires labeled data; risk of silent drift; explainability lower than pre-defined rules.

Approach D: Hybrid ML + Human Review
Instead of forcing every comment into keep/remove, we use probability bands to route low-confidence cases to human review. This approach combines approach B (a small, confirmed set of pre-defined rules) to catch obvious cases, then apply approach C (ML) to the remainder and use probability bands to triage:

- KEEP if P(useful) ≥ 1-threshold
- REMOVE if P(useful) ≤ threshold
- REVIEW otherwise

This triage design prioritizes avoiding false removal of useful documentation while still reducing the overall manual workload. Predictions are written back to a scored CSV with useful_prob, useful_pred_label, and an optional review_flag for the review queue.

Pros: adapts to evolving comment styles; reduces repetitive review; integrates naturally with probabilistic triage. best balance of speed, accuracy, and governance; naturally enables continuous learning and improvement of the comment classifier.
Cons: initial setup requires human effort, accuracy and human review effort depends on diversity of comment styles.

Step 3. PYTHON-BASED BATCH CLEANUP & LOGGING

After the output comments classification CSV file is sufficiently reviewed by a designed programmer and/or study leader to confirm the final action, a downstream Python script (e.g., clean_sas_file_csv.py) reads the CSV file, rewrites each .sas file to a new output folder (preserve folder structure). Comments labeled 'remove' are removed, while comments labeled 'keep' are retained. Along with cleaned programs, a tracking log (CSV) with file path, line ranges removed, action reason (rule ID or model score) and action timestamp is generated. The output codebase therefore represents a submission-ready version with minimized commented-out code while preserving necessary documentation required for maintainability and compliance. In this step, we do not alter the original programs to ensure reversible and traceable changes.

Step 4. VALIDATION AND QUALITY CONTROL

In this step, we rerun all cleaned programs and compare the results with the original ones to identify any artifacts.

- SDTM/ADaM programs that produce SDTM/ADaM datasets: use proc compare with strict criteria to compare the datasets.
- TFL programs that produce analysis results: use proc compre with strict criteria to compare QC data, use RTF and/or PDF compare tools to compare the exact output.

If the outcome is 100% match, we will accept the cleaned programs. Otherwise, an investigation on the comment classification and comment classification CSV files is required.

CASE STUDY

We illustrate our workflow using a representative CSR analysis codebase and compare these different comment classification approaches.

SETUP AND PROFILE

- Source CSR Codebase: ~160 SAS programs (ADaM: ~15, TFL: ~52, macros: ~82, other: ~11)
- Example SAS program before cleaning:

Before Cleaning

```
dm 'log;clear;out;clear;';
/*********************
Protocol Number: XXX
Program Name: q-admh.sas
Program Purpose: qc ADMH
Date Created: 2025-XX-XX
Programmer: ABC XYZ etc.
*******/
%include "..\..\macros\setup.sas";
data qc1;
    length usubjid $40;
    merge sdtm.mh(in=a) adam.adsl(keep=&adslvars);
            by usubjid;
            if a;
run;
/*This is a valid comment in SAS*/
data final;
            set qc1;
/*
            array oo[2] mhstdtc mhendtc;
            array ll[2] astdtcl aendtcl;
            do i=1 to 2;
            if length(oo[i])=7
        then ll[i]='--'||substr(put(input(catx('-
',oo[i],'01'),yymmdd10.),date9.),3);
```

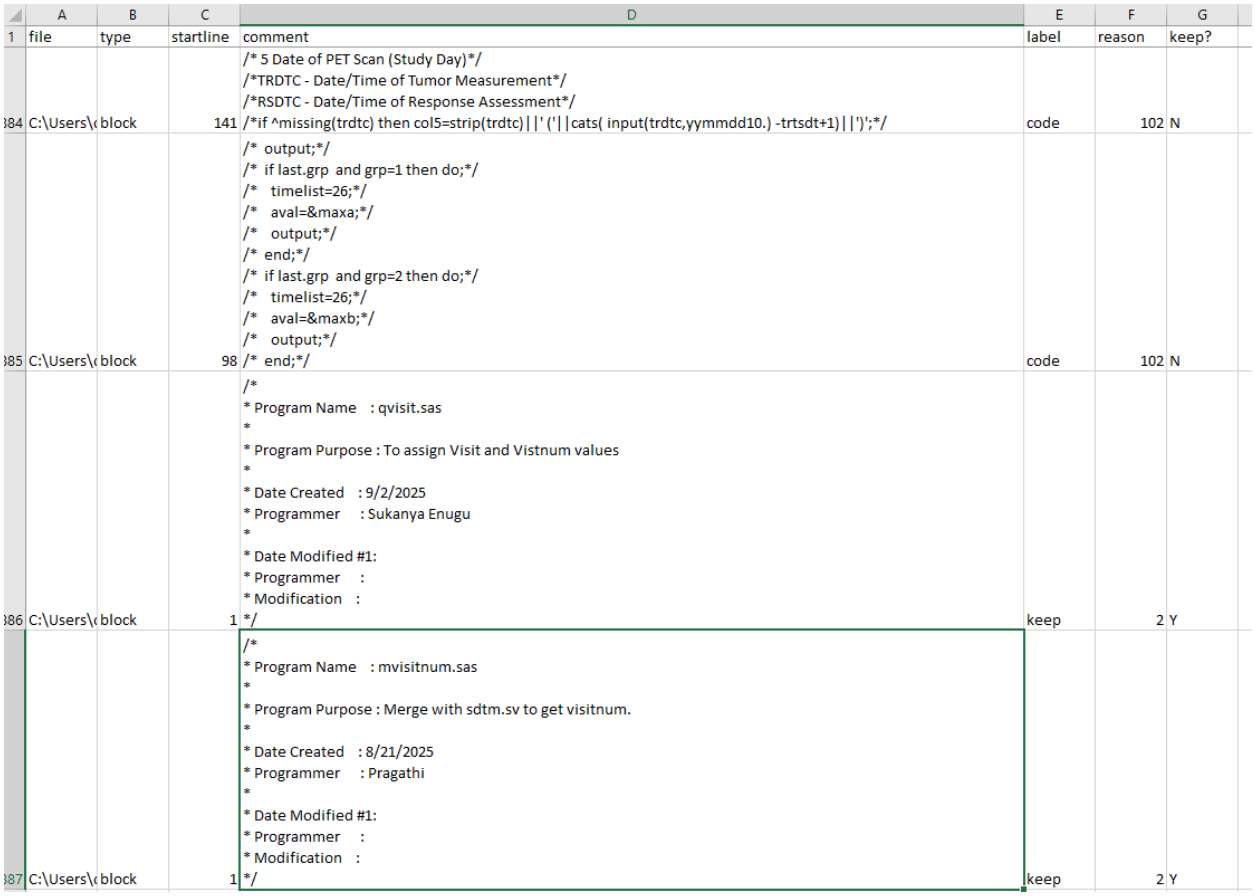
Before Cleaning

end;
*/
/*%somemacro;*/
run;
proc compare data=admh&keep comp=qc&keep listall
method=absolute criterion=0.000001;
id &id;
run;

Comments Extraction and Classification

- Extracted comments: 1118 (block comments: 638; starblock comments: 251, inline comments: 229)
- Preliminary comments justification: 154/1118=10.286% useful (Approach A), 670/1118=59.993% useful (Approach B)

Figure 1. Example Comment Extraction and Classification Worksheet



	A	B	C	D	E	F	G
1	file	type	startline	comment	label	reason	keep?
384	C:\Users\(	block	141	/* 5 Date of PET Scan (Study Day)*/ /*TRDTC - Date/Time of Tumor Measurement*/ /*RSDTC - Date/Time of Response Assessment*/ /*if ^missing(trdtc) then col5=strip(trdtc)||' ('||cats(input(trdtc,yymmdd10.) -trtsdt+1)||')';*/	code	102	N
385	C:\Users\(	block	98	/* output;*/ /* if last.grp and grp=1 then do;*/ /* timelist=26;*/ /* aval=&maxa;*/ /* output;*/ /* end;*/ /* if last.grp and grp=2 then do;*/ /* timelist=26;*/ /* aval=&maxb;*/ /* output;*/ /* end;*/	code	102	N
386	C:\Users\(	block	1	/* * Program Name : qvisit.sas * * Program Purpose : To assign Visit and Vistnum values * * Date Created : 9/2/2025 * Programmer : Sukanya Enugu * * Date Modified #1: * Programmer : * Modification : */	keep	2	Y
387	C:\Users\(	block	1	/* * Program Name : mvisitnum.sas * * Program Purpose : Merge with sdtm.sv to get visitnum. * * Date Created : 8/21/2025 * Programmer : Pragathi * * Date Modified #1: * Programmer : * Modification : */	keep	2	Y

Table 3. Summary of classification results

Approach	Description	Classification Accuracy (on labeled set)	Human Review Effort	Pros	Cons
A	Keep programs header, remove others	100%	Minimal	Fast, deterministic	Over-removal risk; may drop valuable narrative
B	Pre-defined rules	98% for long comments	Moderate (short comments defer to human to decide)	Transparent, explainable	Low flexibility; high maintenance; missed edge cases

Approach	Description	Classification Accuracy (on labeled set)	Human Review Effort	Pros	Cons
C	ML (TF-IDF/Emb + Logistic/SVM)	82%	Low-to-Moderate (4 defer to human to decide per 100 comments)	Adapts to data; reduces review	Needs labeled data; short comments are hard to classify
D	Hybrid (Rules→ML→Review)	100%	Target: Lowest	Best balance; scalable; auditable	Requires initial setup

CLEANUP EXECUTION AND TRACKING

After lead programmers review the comment extraction and classification CSV and finalize decisions (column G in Figure 1), we input this worksheet into a Python program to generate cleaned SAS programs in a new folder. For example, in the cleaned SAS program below, blue highlights retained useful comments, and yellow which highlighted the comment-out codes are successfully removed. The Python execution log (Figure 2) and the comment classification worksheet (Figure 1) together document the process for internal checks and audits.

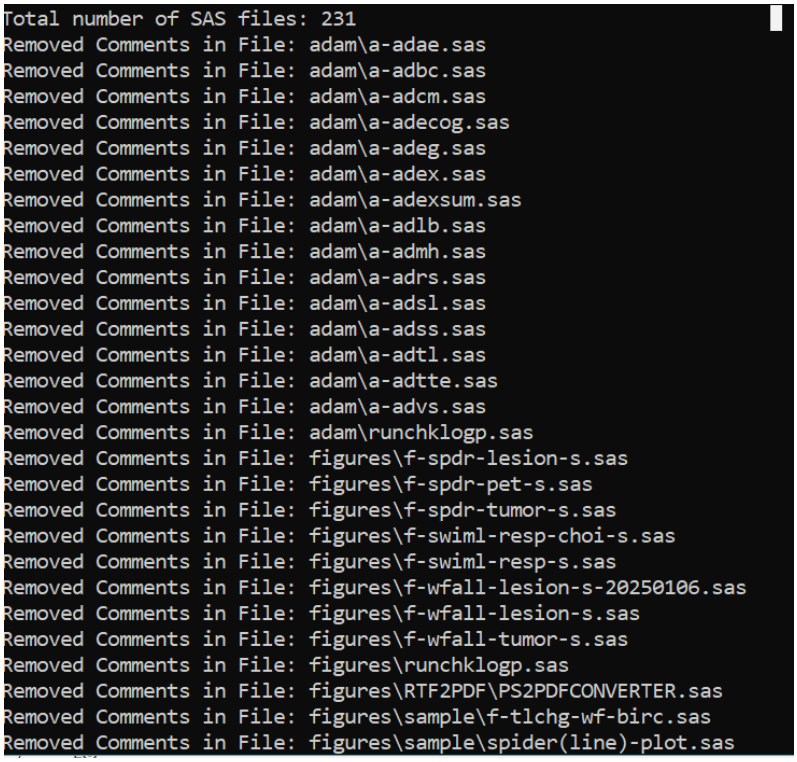
After Cleaning

```
dm 'log;clear;out;clear;';
/*********************
Protocol Number: XXX
Program Name: q-admh.sas
Program Purpose: qc ADMH
Date Created: 2025-XX-XX
Programmer: ABC XYZ etc.
*******/
%include "..\..\macros\setup.sas";
data qc1;
    length usubjid $40;
    merge sdtm.mh(in=a) adam.adsl(keep=&adslvars);
            by usubjid;
            if a;
run;

/*This is a valid comment in SAS*/

data final;
            set qc1;
run;
proc compare data=admh&keep comp=qc&keep listall
method=absolute criterion=0.000001;
    id &id;
run;
```

Figure 2. Example Python Tracking Log



Validation and QC

We execute the cleaned SAS programs once again and evaluate the outputs through various methodologies. For ADaM programs, we employ PROC COMPARE with strict criteria to verify that datasets generated by separate copies of the programs are identical, as illustrated in Figure 3. For TFL programs, our internal RTF comparison toolkit is utilized to assess the content on each page of individual RTF files, with Figure 4 presenting an example of the comparison results.

Figure 3. Example Validation Result: QC datasets with PROC COMPARE

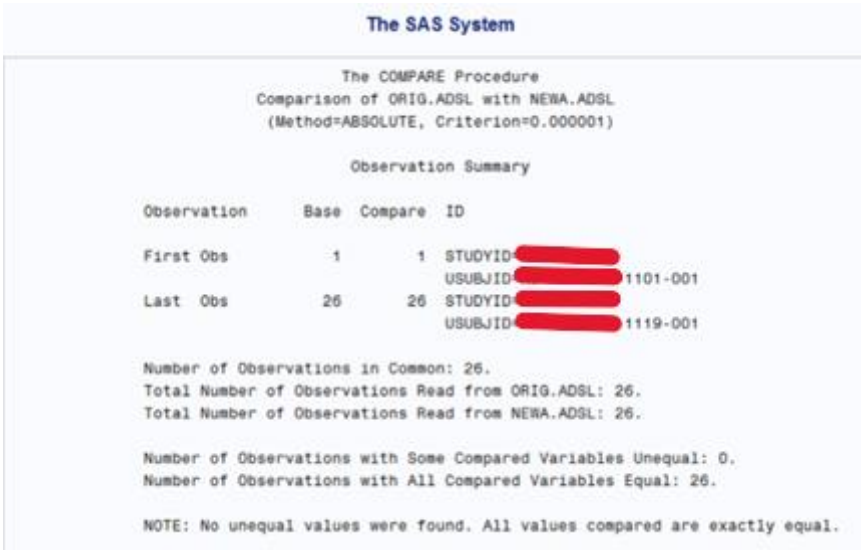
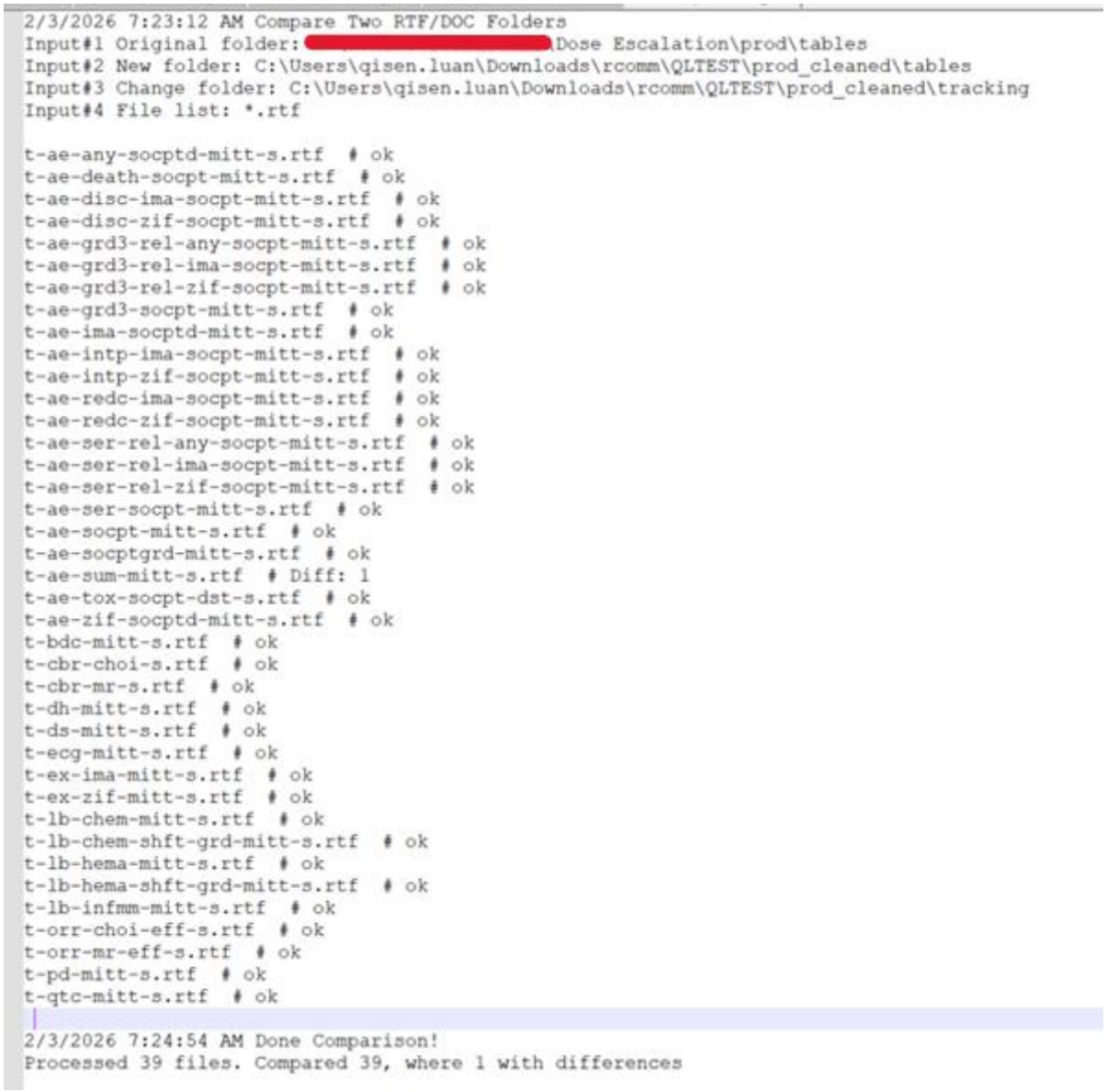


Figure 4. Example Validation Result: Compare RTF *via* RTF Compare Toolkit



```
2/3/2026 7:23:12 AM Compare Two RTF/DOC Folders
Input#1 Original folder: [redacted]Dose Escalation\prod\tables
Input#2 New folder: C:\Users\qisen.luan\Downloads\rcomm\QLTEST\prod_cleaned\tables
Input#3 Change folder: C:\Users\qisen.luan\Downloads\rcomm\QLTEST\prod_cleaned\tracking
Input#4 File list: *.rtf

t-ae-any-socptd-mitt-s.rtf  # ok
t-ae-death-socpt-mitt-s.rtf  # ok
t-ae-disc-ima-socpt-mitt-s.rtf  # ok
t-ae-disc-zif-socpt-mitt-s.rtf  # ok
t-ae-grd3-rel-any-socpt-mitt-s.rtf  # ok
t-ae-grd3-rel-ima-socpt-mitt-s.rtf  # ok
t-ae-grd3-rel-zif-socpt-mitt-s.rtf  # ok
t-ae-grd3-socpt-mitt-s.rtf  # ok
t-ae-ima-socptd-mitt-s.rtf  # ok
t-ae-intp-ima-socpt-mitt-s.rtf  # ok
t-ae-intp-zif-socpt-mitt-s.rtf  # ok
t-ae-redc-ima-socpt-mitt-s.rtf  # ok
t-ae-redc-zif-socpt-mitt-s.rtf  # ok
t-ae-ser-rel-any-socpt-mitt-s.rtf  # ok
t-ae-ser-rel-ima-socpt-mitt-s.rtf  # ok
t-ae-ser-rel-zif-socpt-mitt-s.rtf  # ok
t-ae-ser-socpt-mitt-s.rtf  # ok
t-ae-socpt-mitt-s.rtf  # ok
t-ae-socptgrd-mitt-s.rtf  # ok
t-ae-sum-mitt-s.rtf  # Diff: 1
t-ae-tox-socpt-dst-s.rtf  # ok
t-ae-zif-socptd-mitt-s.rtf  # ok
t-bdc-mitt-s.rtf  # ok
t-cbr-choi-s.rtf  # ok
t-cbr-mr-s.rtf  # ok
t-dh-mitt-s.rtf  # ok
t-ds-mitt-s.rtf  # ok
t-ecg-mitt-s.rtf  # ok
t-ex-ima-mitt-s.rtf  # ok
t-ex-zif-mitt-s.rtf  # ok
t-lb-chem-mitt-s.rtf  # ok
t-lb-chem-shft-grd-mitt-s.rtf  # ok
t-lb-hema-mitt-s.rtf  # ok
t-lb-hema-shft-grd-mitt-s.rtf  # ok
t-lb-infmm-mitt-s.rtf  # ok
t-orr-choi-eff-s.rtf  # ok
t-orr-mr-eff-s.rtf  # ok
t-pd-mitt-s.rtf  # ok
t-qtc-mitt-s.rtf  # ok

2/3/2026 7:24:54 AM Done Comparison!
Processed 39 files. Compared 39, where 1 with differences
```

CONCLUSION

A fully automated, ML-only cleanup risks deleting valuable documentation or missing subtle code artifacts. Conversely, a manual-only approach does not scale and lacks traceability for auditing. The human-in-the-loop hybrid workflow achieves the right balance: ML-assist provides scalability, speed, and consistency; human review ensures overall accuracy, protects critical context, and builds a continuously improving system. With end-to-end logging and rigorous output equivalence checks, the process yields submission-ready SAS programs while reducing human effort and saving time.

REFERENCES

[1] James, J. (n.d.). SAS-Comment-Remover [Source code]. GitHub. Retrieved February 10, 2026, from https://github.com/A142763/SAS-Comment-Remover

[2] Katam, J. (2019). Understanding regular expression and its application in SAS using PRX (PERL regEx) functions (Paper CT12) [Conference paper]. PhUSE EU Connect 2019: The Clinical Data Science Conference, Amsterdam, Netherlands. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2019/Connect/EU/Amsterdam/PAP_CT12.pdf

[3] PyCQA. (2025, October 27). eradicate (Version 3.0.1) [Source code]. GitHub. Retrieved February 10, 2026, from https://github.com/PyCQA/eradicate

[4] SAS Institute Inc. (n.d.). SAS comments. SAS Help Center. Retrieved February 10, 2026, from https://documentation.sas.com/doc/en/pgmsascdc/v_070/lepg/p129pt9a1g2fgxn19d1nyaqck4gq.htm

[5] SAS User. (2022, August 5) How to add comments in SAS (7 proven ways). Learn SAS Code. https://learnsascode.com/sas-comments/

CONTACT INFORMATION

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

Author Name: Qisen Luan, Qiong Wie, Lixin Gao
Company: BioPier LLC, a Veramed Company
Address: 80 Blanchard Rd Ste 104, Burlington, MA 01803
Work Phone: (781) 354-1924
Email: qluan@biopier.com, qwei@biopier.com, lgao@biopier.com
Website: https://biopier.com/