PharmaSUG China 2026 - Paper AA148-2026

# AI-Powered R Shiny Tool for Automated Footnote Comparison Across TLF Packages and Shells

Yuan Sun Sanofi, Chengdu, China

## ABSTRACT

In the preparation of TLF (Tables, Listings, and Figures) packages, a critical step is comparing titles and footnotes between packages and corresponding shells. Usually, footnote verification has been performed manually—a process that is both time-consuming and prone to human error. To enhance the efficiency and accuracy of this quality control process, we developed an AI-powered R Shiny tool that automatically captures and compares footnotes across packages and shells.

This paper presents the design, functionality, and implementation of this validation tool, demonstrating how automation can significantly improve the TLF review workflow in statistical programming. Additionally, we discuss the benefits of AI-driven development in accelerating innovation and improving programmer productivity in statistical programming environments.

## INTRODUCTION

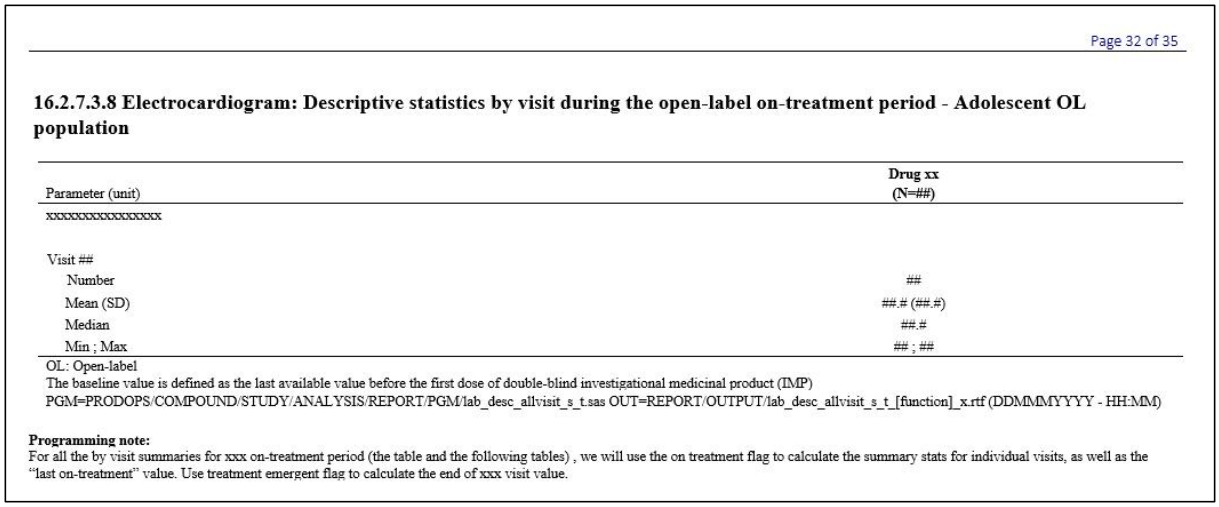
When preparing a Clinical Study Report (CSR), statistical programmers are required to compile outputs into structured packages and submit them as addenda in accordance with regulatory agency delivery requirements. Each package encompasses multiple outputs organized under a common theme — such as adverse events, efficacy, and so on. A critical validation step prior to delivery is ensuring that the footnotes in each output are consistent with those specified in the mock shells provided by statisticians.

When footnotes are manually created rather than automatically generated from mock shells, discrepancies between shells and outputs are prone to occur. Traditionally, statistical programmers and statisticians review footnotes through a manual, line-by-line comparison between shells and packages — a process that is both time-consuming and susceptible to human error, as subtle mismatches can easily be overlooked during visual inspection.

To improve the efficiency and accuracy of this review process, we developed an R Shiny–based tool that automatically extracts footnotes from both shells and output packages, performs comparison, and highlights inconsistencies for further inspection. This work was funded by Sanofi.

## LAYOUT OF SHELL AND PACKAGE

We constructed a dummy shell (referred to as Shell throughout this paper) and a corresponding dummy package (referred to as TLF package). Notably, Shell contains a greater number of outputs than TLF package, which better reflects real-world scenarios. Figure 1 and Figure 2 illustrate representative pages from Shell and TLF, respectively.



Page 32 of 35

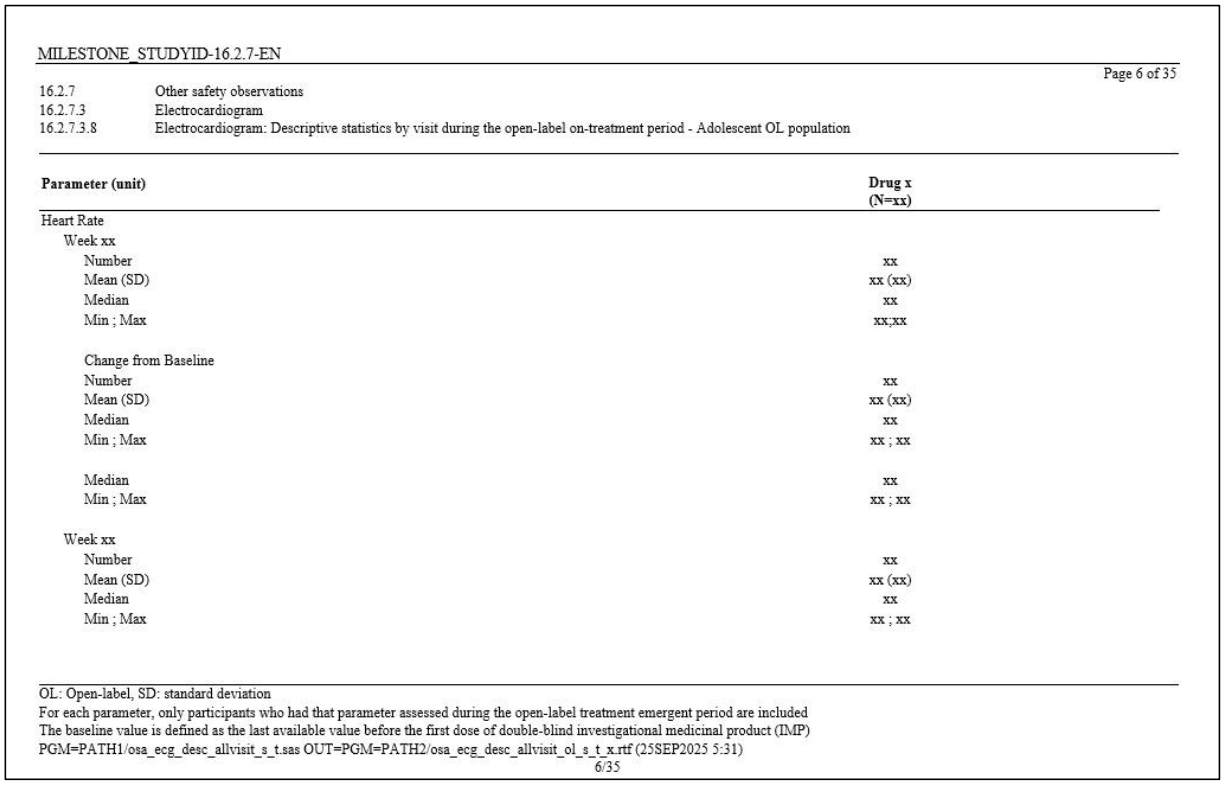
**16.2.7.3.8 Electrocardiogram: Descriptive statistics by visit during the open-label on-treatment period - Adolescent OL population**

| Parameter (unit) | Drug xx (N=##) |
|---|---|
| xxxxxxxxxxxxxxx | |
| Visit ## | |
| Number | ## |
| Mean (SD) | ##.# (##.#) |
| Median | ##.# |
| Min ; Max | ## ; ## |

OL: Open-label
The baseline value is defined as the last available value before the first dose of double-blind investigational medicinal product (IMP)
PGM=PRODOPS/COMPOUND/STUDY/ANALYSIS/REPORT/PGM/lab_desc_allvisit_s_t.sas OUT=REPORT/OUTPUT/lab_desc_allvisit_s_t_[function]_x.rtf (DDMMMYYYY - HH:MM)

**Programming note:**
For all the by visit summaries for xxx on-treatment period (the table and the following tables) , we will use the on treatment flag to calculate the summary stats for individual visits, as well as the "last on-treatment" value. Use treatment emergent flag to calculate the end of xxx visit value.

**Figure 1. Layout of output in Shell**



MILESTONE_STUDYID-16.2.7-EN

Page 6 of 35

16.2.7 Other safety observations
16.2.7.3 Electrocardiogram
16.2.7.3.8 Electrocardiogram: Descriptive statistics by visit during the open-label on-treatment period - Adolescent OL population

| Parameter (unit) | Drug x (N=xx) |
|---|---|
| Heart Rate | |
| Week xx | |
| Number | xx |
| Mean (SD) | xx (xx) |
| Median | xx |
| Min ; Max | xx;xx |
| Change from Baseline | |
| Number | xx |
| Mean (SD) | xx (xx) |
| Median | xx |
| Min ; Max | xx ; xx |
| Median | xx |
| Min ; Max | xx ; xx |
| Week xx | |
| Number | xx |
| Mean (SD) | xx (xx) |
| Median | xx |
| Min ; Max | xx ; xx |

OL: Open-label, SD: standard deviation
For each parameter, only participants who had that parameter assessed during the open-label treatment emergent period are included
The baseline value is defined as the last available value before the first dose of double-blind investigational medicinal product (IMP)
PGM=PATH1/osa_ecg_desc_allvisit_s_t.sas OUT=PGM=PATH2/osa_ecg_desc_allvisit_ol_s_t_x.rtf (25SEP2025 5:31)
6/35

**Figure 2. Layout of output in TLF package**

## CRITICAL MODULES OF THIS TOOL

The automated footnote comparison process can be decomposed into five critical steps:

1. **Reading** the Word files of Shell and TLF packages
2. **Extracting** the title and footnotes from each output in both Shell and TLF packages
3. **Comparing** the footnotes across corresponding outputs
4. **Summarizing** the comparison results and exporting them as an Excel file

To enhance accessibility, we developed an R Shiny-based web application that integrates all four steps into a streamlined workflow. Users can upload files and obtain comparison results through a simple point-

and-click interface, without requiring coding expertise. To achieve this user-friendly implementation, an additional critical component was required:

5. **Developing** the user interface (UI) and interactive functionality

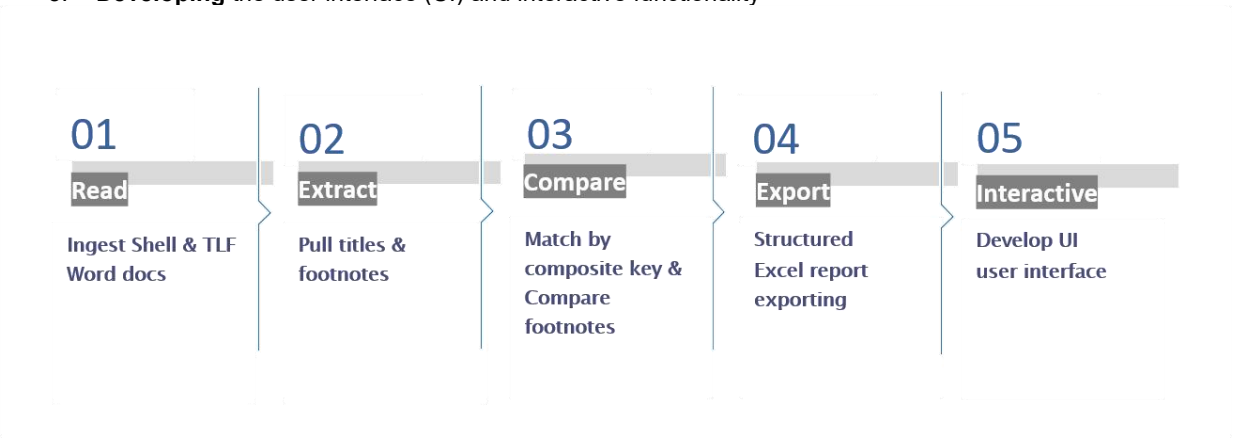



**Figure 3. Process of the critical modules**

In this section, we present the technical details of core module underlying this tool.

## EXTRACTING THE TITLE AND FOOTNOTES

The most critical component of this tool is accurately capturing footnotes from both Shell and TLF packages. Since their document structures differ significantly, we developed two distinct extraction algorithms.

### TLF package structure and extraction logic

TLF packages follow a standardized structure: titles are placed in the header section of each page, and footnotes in the footer section (Figure 4). For multi-page tables, each page contains both elements. Our algorithm captures title and footnote from every page and removes duplicates.

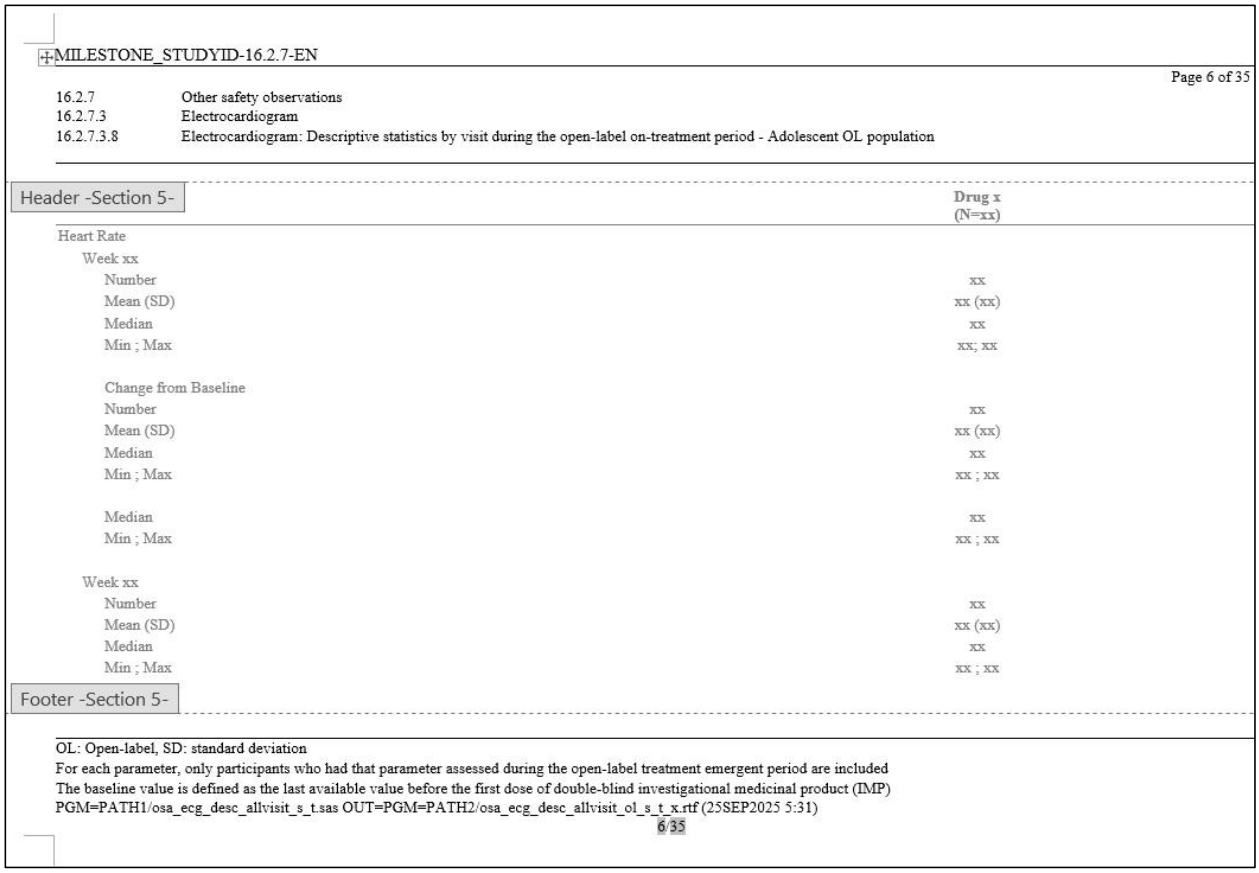



**Figure 4. The location of the title and footnote for TLF package**

## Shell Structure and Extraction Logic

Shell documents have a more complex structure: both titles and footnotes are embedded in the document body with non-standardized positions, and multiple outputs may appear on a single page. Additionally, individual tables can span multiple pages with inconsistent formatting—for example, a table distributed across three pages may have the title on page 1, footnote on page 3, and neither on page 2.

However, we identified consistent formatting patterns: titles always appear above the table's top border, and footnotes always appear below the bottom border. Each footnote terminates with a program identifier "PGM=XXX". Using these patterns, we developed a table border-based positioning method to extract titles and footnotes.

The R packages **xml2** and **stringr** are the primary tools used to read, parse, clean, and extract the required information from Word files. Below we present the critical functions employed in this implementation.

| Package | Function | Purpose |
|---|---|---|
| xml2 | read_xml() | Read document.xml, _rels, header/footer, and other XML files |
| | xml_ns() | Retrieve Word namespace (e.g., w: prefix) |
| | xml_find_first() / xml_find_all() | XPath-based node search (paragraphs w:p, tables w:tbl, rows w:tr, etc.) |
| | xml_children() | Traverse child nodes under w:body |
| | xml_name() | Determine node type (p / tbl / sectPr) |

| | | |
|---|---|---|
| | xml_text() | Extract node text content |
| | xml_attr() / xml_attrs() | Read attributes (style val, relationship Id, bookmark name, etc.) |
| stringr | str_squish() | Compress whitespace (in clean_text()) |
| | str_replace_all() | Normalize Unicode symbols (e.g., ≥→>=, full-width colons) |
| | str_match() | Match title numbering patterns (16.x.x...), PGM= lines |
| | str_detect() | Detect page numbers, TOC styles, "Programming note:", footnote prefixes |
| | str_count() / str_sub() / str_locate() | Calculate title hierarchy levels, string segmentation |

**Table 1. The main functions used in R packages xml2 and stringr**

## COMPARING THE FOOTNOTES ACROSS CORRESPONDING OUTPUTS

Once titles and footnotes are extracted, the records are stored in a structured Excel file. Each output occupies a single row, with titles and footnotes stored in individual cells (Figure 5). The exported Excel file follows a fixed schema of **N rows × 15 columns**, where:

- **N** represents the total number of outputs
- **15 columns** consist of: Table ID, Title 1–4, and Footnote 1–10

This standardized format is applied consistently to both Shell and TLF exports.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | tableid | title1 | title2 | title3 | title4 | footnote1 | footnote2 | footnote3 | footnote4 | footnote5 | footnote6 | footnote7 | footnote8 | footnote9 | footnote10 |
| 2 | Table_1 | 16.2.7 Other safety observations | 16.2.7.1 PCSA | 16.2.7.1.1 PCSAs for vital sign | | PCSA: Potentially clinically significant abnormality | Adult criteria to be used when adolescents grown into adults during the study | | | | | | | | |
| 15 | Table_14 | 16.2.7 Other safety observations | 16.2.7.3 Electrocardiogram | 16.2.7.3.9 Electrocardiogram: Listing of all abnormality (PCSA) for participants with events onset during the OL and LTE treatment emergent period - Adolescent OL and LTE population | | DB: Double-blind, OL: Open-label, LTE: Long-term extension, PCSA: Potentially clinically significant abnormality | aFor sex: M=Male; F=Female | bFor race: W=White; B=Black or African American; A=Asian; AI=American Indian or Alaska Native; PI=Native Hawaiian or Other Pacific Islander; O=Other; NR= Not Reported | cRelative day to start date of investigational medicinal product | Flags definition: B: baseline; P: Post-treatment; E: Last on-treatment value; -: < 50 beats/min for heart rate; --: < 40 beats/min for heart rate; ---: < 30 beats/min for heart rate; | +: > 90 beats/min for heart rate, > 200 msec for PR Interval, > 110 msec for QRS Duration, > 500 msec for QT Interval, > 450 msec for QTcF Interval; | ++: > 100 beats/min for heart rate, > 220 msec for PR Interval, > 120 msec for QRS Duration, > 480 msec for QTcF Interval; | +++: > 120 beats/min for heart rate, > 240 msec for PR Interval, > 500 msec for QTcF Interval; | D: decrease from baseline >= 20 beats/min for heart rate; I: increase from baseline >= 20 beats/min for heart rate, increase from baseline >= 25% for PR Interval, increase from baseline >= 25% for QRS Duration, increase from baseline [30-60] msec for | |

**Figure 5. Title and footnote extracted from shell**

## FOOTNOTE COMPARISON

The two exported files are then merged into a single dataset using the title as the merge key, enabling systematic cross-comparison of footnotes between Shell and TLF packages.

First, records from the Shell and TLF parsed datasets are aligned by title1–title4. An exact match is attempted by comparing a normalized composite key built from all four title fields. If a matching Shell row is found that row is paired with the corresponding TLF row for comparison.

If no exact match is found, a fuzzy match is applied: the full title text is compared using Levenshtein-based similarity, and the Shell row with the highest similarity score is selected as the best match. Each

TLF row is therefore paired with at most one Shell row (or left unmatched if no Shell data are available).

Two diff granularities are supported: word-level and character-level.

In **word-level** mode, text is tokenized into three types: (1) alphanumeric tokens (including underscores), (2) consecutive whitespace, and (3) individual punctuation or symbols. Differences are reported at the token level, which is useful for detecting changes in words, phrases, or punctuation blocks within a sentence.

In **character-level** mode, text is split into individual characters. This approach is more granular and sensitive: it can highlight a single-character change within a word, or subtle differences in spaces and symbols. Word-level comparison is generally easier to read for sentence-level changes; character-level comparison is better for spotting minor edits.

In addition to footnotes, titles are also compared and highlighted when the paired Shell and TLF titles are not identical—especially for rows matched by fuzzy alignment rather than an exact 100% match. The same word-level or character-level setting applies to both title and footnote comparisons.

## COMPARISON RESULTS AND OUTPUT

The comparison results are exported as an Excel file with the following structure:

- **Fixed columns**: Table ID, Title 1–4.
- **Dynamic columns**: Adjacent column pairs such as shell_footnote1 and tlf_footnote1, showing the Shell and TLF versions side by side. Only footnotes with detected differences are included; identical footnotes are omitted to keep the output concise.
- **Summary columns** (appended at the end): Difference Status: "Identical" if no differences are found in that row, or "Different" if any title or footnote cell differs. Difference Cells: the number of cells in that row containing a detected difference.

Different types of mismatches are highlighted in distinct colors for easy identification: green for additions, orange for deletions, and yellow for modifications. The Difference Status column uses conditional formatting (green for Identical, red for Different).

As illustrated in Figure 6 below, columns F and G display the mismatched Footnote 1 from the Shell and TLF packages, respectively, while the Difference Status column indicates the comparison outcome as *"Different"* for the corresponding output.

| A | B | C | D | E | F | G | H | I | N | O |
|---|---|---|---|---|---|---|---|---|---|---|
| tableid | title1 | title2 | title3 | title4 | shell_footnote1 | tlf_footnote1 | shell_footnote2 | tlf_footnote2 | difference_status | difference_cells |
| Table_1 | 16.2.7 Other safety | 16.2.7.3 | 16.2.7.3.2 QTcF Interval: | | | | | | Identical | 0 |
| Table_2 | 16.2.7 Other safety | 16.2.7.3 | 16.2.7.3.5 | | | | | | Identical | 0 |
| Table_3 | 16.2.7 Other safety | 16.2.7.3 | 16.2.7.3.7 Plot of mean | | Footnote: | OL: Open-label, | OL: Open-label | | Different | 3 |
| Table_4 | 16.2.7 Other safety observations | 16.2.7.3 Electrocardiogram | 16.2.7.3.8 Electrocardiogram: Descriptive statistics by visit during the open-label on-treatment period - Adolescent OL population | | OL: Open-label | OL: Open-label, SD: standard deviation | The baseline value is defined as the last available value before the first dose of double-blind investigational medicinal product (IMP) | For each parameter, only participants who had that parameter assessed during the open-label treatment emergent period are included | Different | 4 |
| Table_5 | 16.2.7 Other safety observations | 16.2.7.3 Electrocardiogram | shell: 16.2.7.3.9 Electrocardiogram: | shell: tlf: 16.2.7.3.10.1 | | | | | Different | 4 |

**Figure 6. The excel file of comparison result**

## USER INTERFACE DEVELOPMENT

The user interface is built with Shiny, bslib, and DT. bslib provides the Bootstrap 5 theme and layout; Shiny supplies interactive controls; and DT displays the comparison table. Together with other module functions assemble these into a three-tab workflow: File Upload, Footnote Compare and Compare Result - where users upload Shell and TLF .docx files, choose word- or character-level diff granularity, run the comparison, and download results.

## TOOL USAGE AND RESULT VISUALIZATION

To use this tool, follow these prerequisites and setup steps. Download the complete project folder to your local computer. Ensure R Studio is installed on your system and verify that all required R packages are installed in your R environment. Once the prerequisites are met, you can launch the tool using either method: 1). Command line approach: Execute the following code in R Studio (shown as Figure 7). 2). Direct file approach: Open and run the app.R file directly in R Studio.

```
# 1. Set working directory to the project root
setwd("C:/Users/Public/R-Shiny-DocX-Comparer")

# 2. Use the project's local .R-lib (required before loading shiny)
.libPaths(c(normalizePath(".R-lib", winslash = "/", mustWork = TRUE), .libPaths()))

# 3. Launch the Shiny app
shiny::runApp("app.R")
```

**Figure 7. R code of using this tool**

The web application consists of three pages: File upload, Footnote Compare and Compare Result.

### FILE UPLOAD

The File Upload page contains two dedicated upload sections for the Shell and TLF files, respectively. Users can either select files through the file browser or drag and drop files directly from their local directory. Once the upload is complete, the **Preview** button allows users to verify the uploaded file content online before proceeding.

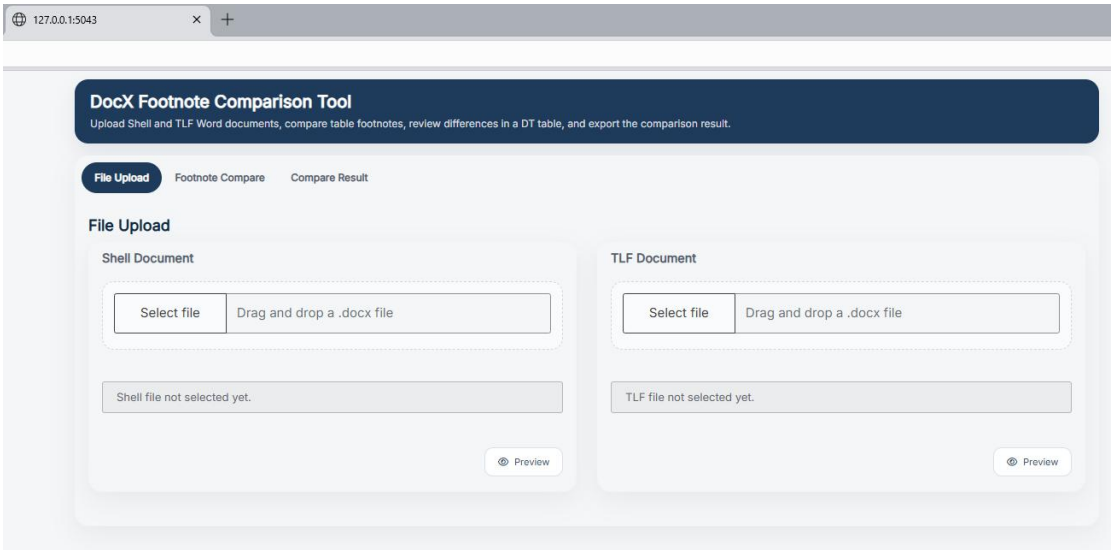



**Figure 8. Layout of file upload page**

### FOOTNOTE COMPARE

After successfully uploading both files, users can navigate to the Footnote Compare page and click the "Run Footnote Compare" button to initiate the automated comparison process. You can select either word-level or character-level button as you wish before executing comparison.

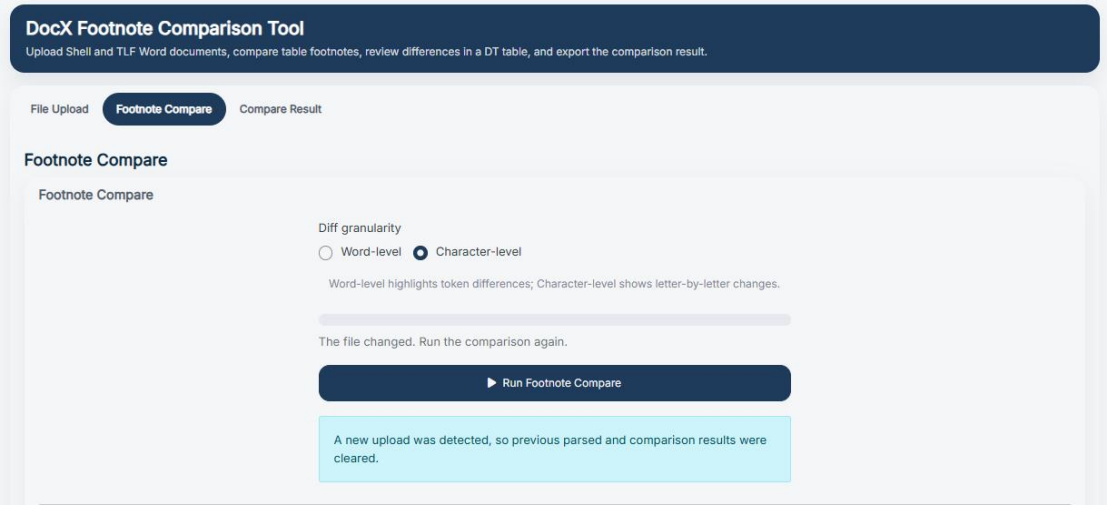



**Figure 9. Layout of footnote compare page**

## COMPARE RESULT

Upon completion of the comparison, users can navigate to the Compare Result page to review the results interactively online or download them as an Excel file for offline review. The Excel file is shown as Figure 6 above. What's more, this page also includes a built-in search function, enabling users to filter and locate specific outputs of interest efficiently.

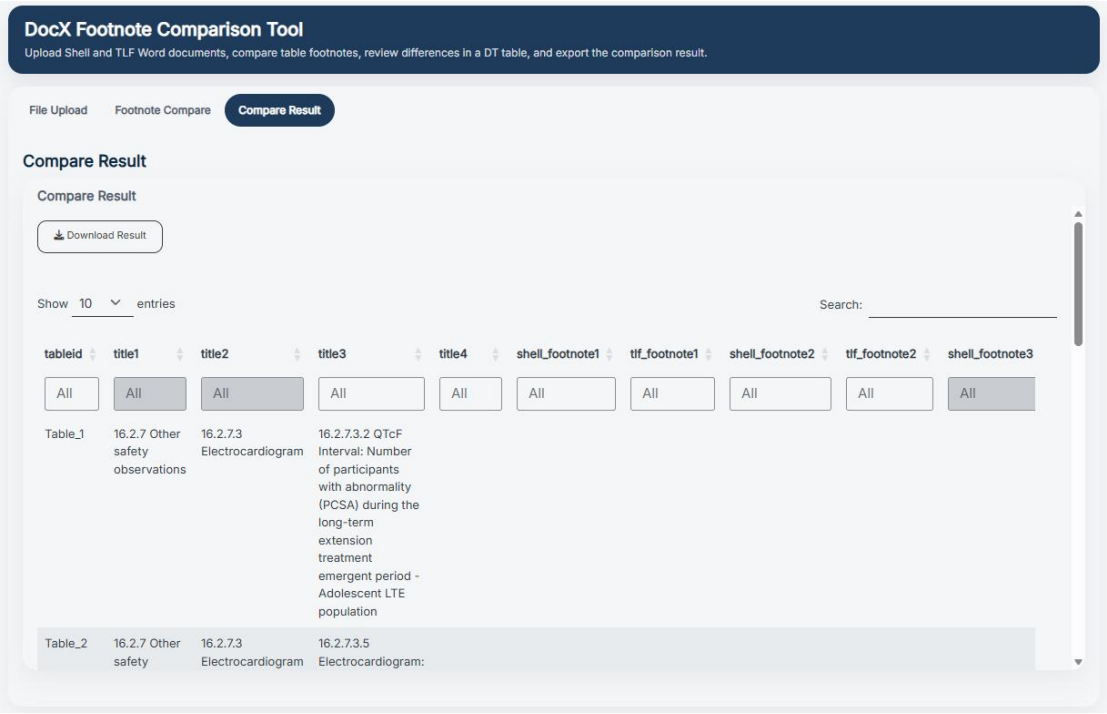



**Figure 10. Layout of compare result page**

## ADDITIONAL FEATURE: TITLE AND FOOTNOTE EXTRACTION

The Footnote Compare page also provides download sections for exporting all captured titles and footnotes from both Shell and TLF packages.

This feature enables programmers to extract titles and footnotes from Shell before generating TLF

outputs, eliminating the need for subsequent footnote comparison and streamlining the workflow.

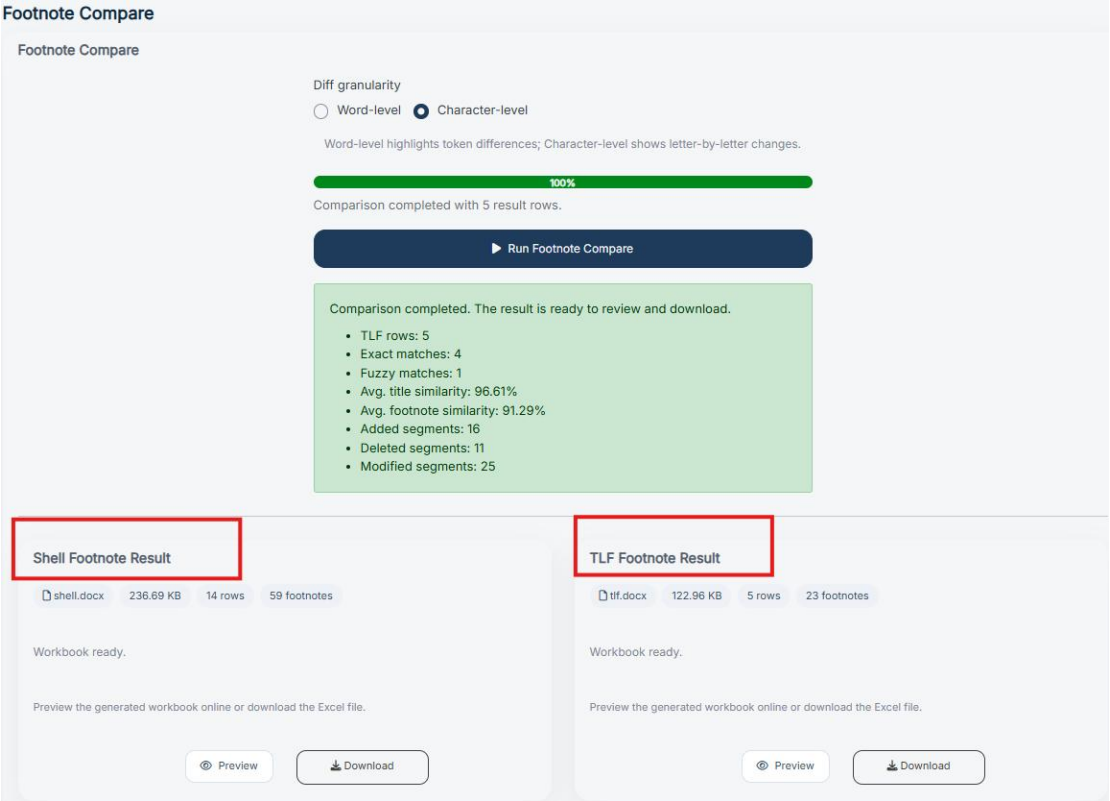



**Figure 11.The function of exporting results in footnote compare page**

## AI-ASSISTED DEVELOPMENT APPROACH

This tool was developed entirely with the assistance of AI agents. After designing the overall architecture, we consulted AI to recommend the most suitable programming language for our requirements and working environment. Based on AI's suggestion, R Shiny was selected as the optimal framework to achieve our objectives.

### DEVELOPMENT APPROACH

Given our limited familiarity with text processing packages such as xml2, and the complexity of extracting footnotes from Shell and TLF documents with their varied structural patterns, we delegated the coding implementation to AI. However, successful AI-assisted development required careful preparation and structured guidance:

1. **Clear project architecture**: We defined the complete tool structure upfront and instructed AI to create distinct modules aligned with the logical workflow.
2. **Precise prompt engineering**: We crafted detailed prompts for each module, clearly specifying the expected functionality and outputs.
3. **Template and test file preparation**: We prepared representative template files and comprehensive test cases to validate AI-generated code.
4. **Iterative testing and validation**: After AI completed each module, we conducted thorough testing to ensure it met our criteria. Only when all modules were executed successfully did we integrate them into the final tool.

This modular, test-driven approach ensured the resulting tool was both efficient and stable.

### LESSONS LEARNED: WORKING EFFECTIVELY WITH AI

AI can be a powerful enabler for implementing ideas, particularly when working with unfamiliar

programming languages or frameworks. Prior coding expertise in a specific language is no longer a prerequisite—developers can communicate requirements through natural language, and AI can iteratively refine the implementation based on feedback. This democratizes software development, allowing domain experts to become effective developers.

However, successful AI collaboration requires:

- **Strong conceptual clarity**: A clear understanding of what you want to achieve and how the components should interact
- **Logical thinking**: The ability to decompose complex problems into well-defined modules and workflows
- **Effective prompt engineering**: Practice in articulating requirements precisely to minimize ambiguity and iteration cycles

In our experience, the ability to manage overall system logic and architecture is the most critical skill when working with AI. As AI coding assistants continue to evolve, investing time in developing prompt engineering skills will significantly enhance productivity and development outcomes.

AI accelerated implementation, but human oversight remained essential for architecture, business rules, test design, and regulatory appropriateness. Domain experts still need strong system-level thinking and effective prompt writing; AI complements rather than replaces statistical programming judgment.

## CONCLUSION

This paper presents an R Shiny-based automated tool for validating footnote consistency between mock shells and output packages in Clinical Study Report preparation. By automating extraction, comparison, and reporting of discrepancies, the tool addresses a critical bottleneck in statistical programming workflows that traditionally relied on manual, error-prone review processes.

The tool delivers three main contributions:

1. **Dual extraction algorithms**: Tailored parsing strategies for Shell and TLF documents, using table borders and pattern recognition to accurately capture titles and footnotes.
2. **Intelligent matching and comparison**: Hierarchical matching (exact followed by fuzzy) combined with word-level and character-level comparison strategies for high-precision discrepancy detection.
3. **User-friendly automation**: An intuitive web interface that eliminates technical barriers, enabling programmers to validate footnotes efficiently without coding expertise.

The tool significantly reduces validation time, minimizes human error, and accelerates CSR delivery. The color-coded Excel reports facilitate rapid identification and resolution of mismatches. Additionally, the tool supports proactive footnote extraction during output generation, preventing discrepancies before they occur.

This project demonstrates effective AI-assisted development through clear architectural design, modular decomposition, and precise prompt engineering—enabling domain experts to implement sophisticated solutions without deep programming expertise.

Potential enhancements include support for additional document formats, integration with version control systems, and batch processing capabilities for multiple studies.

## REFERENCES

[1]Qi, W. (2025). Tool to extract titles and footnotes from TFL shell [Conference paper]. PharmaSUG China 2025, Shanghai, China.

[2] Wang, L., & Tong, J. (2025). An efficient R Shiny app for spelling validation in pharmaceutical clinical study documentation review [Conference paper]. PharmaSUG China 2025.

[3]CDISC. (2023). Analysis results standard (ARS) version 1.0. Clinical Data

Interchange Standards Consortium. https://www.cdisc.org/standards/foundational/ars

[4]Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 10(8), 707-710.
[5]U.S. Food and Drug Administration. (2022). E6(R3) good clinical practice: Integrated addendum to ICH E6(R2) - Draft guidance for industry. https://www.fda.gov/regulatory-information/search-fda-guidance-documents/e6r3-good-clinical-practice-integrated-addendum-ich-e6r2
[6]White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. arXiv preprint arXiv:2302.11382. https://arxiv.org/abs/2302.11382
[7]Wickham, H. (2023). stringr: Simple, consistent wrappers for common string operations (Version 1.5.0) [Computer software]. https://CRAN.R-project.org/package=stringr
[8]Wickham, H., Hester, J., & Ooms, J. (2023). xml2: Parse XML (Version 1.3.5) [Computer software]. https://CRAN.R-project.org/package=xml2
[9]Xiao, N., & Xu, Q. (2018). Reproducible research and automation in clinical trial reporting. Pharmaceutical Statistics, 17(5), 521-530. https://doi.org/10.1002/pst.1869

## CONTACT INFORMATION

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

Yuan Sun
17F, Building D, Zhonghai International Community, High-Tech Zone, Chengdu, P.R.C
Yuan Sun was contracted by Sanofi for this work
Yuan1.Sun@sanofi.com