PharmaSUG 2026 - Paper AI-305

Improving AI SAS-to-R Code Migration via an Intermediate Design Document Layer

Junze Zhang and Amy Zhang, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

The pharmaceutical industry is increasingly migrating clinical programming workflows from SAS to R. SAS and R have resemblance with one another and perform similar functions, but they also differ in some fundamental ways. “Direct” AI translation from SAS to R often reverts to block-by-block literal conversion (Bosak & Varney, 2025; Cheng et al., 2025). This can produce non-idiomatic R code with higher maintenance burden and subtle logic defects, especially when SAS programs rely on sort-dependent BY-group semantics (e.g., first./last., retain, and merge) and SAS-specific missing-value behavior. We propose a design-first, two-step workflow that inserts an intermediate design document layer between SAS and R. First, a large language model (LLM) analyzes SAS programs and generates a structured design document capturing purpose, inputs, outputs, transformation steps, and key assumptions (e.g., ordering requirements for baseline/last-value derivations, join keys and expected cardinality, missing-value handling, deduplication rules, and validation checks). A high-level human review, rather than in-depth line-by-line checks, refines the design to ensure requirement alignment. Then, the LLM generates R code directly from the reviewed design document, enabling a vernacular-driven implementation that is more idiomatic, modular, and testable. We demonstrate this approach using previously developed macro code examples and compare their results to those from a direct translation. Outcomes are evaluated by output alignment with SAS, number of iteration cycles, and manual revision rate. This intermediate design layer decouples understanding from implementation, reduces literal-translation artifacts, and improves migration reliability for regulated clinical deliverables.

INTRODUCTION

BACKGROUND: WHY DIRECT SAS TO R TRANSLATION FAILS

SAS and R differ in several fundamental ways that make literal, block-by-block translation unreliable. We list some instances where direct translation from SAS to R may fail. First, SAS macros are a code-generation mechanism (text substitution that produces SAS code), whereas R functions are runtime abstractions (Ling & Wang, 2025); direct translation can misinterpret “generation logic” as “business logic,” leading to verbose and hard-to-maintain R code. Second, many SAS programs rely on sort-dependent BY-group semantics (Danner & Sarkar, 2023). For example, first./last., retain, and merge behavior implicitly depends on prior proc sort. In R, joins do not require sorting, but clinical rules such as baseline/last non-missing selection and windowing still depend on time order and must be expressed explicitly (e.g., arrange() + group_by()), which direct translation may omit. Third, the SAS DATA step often encodes row-wise, stateful processing (e.g., retain) that R typically implements more clearly with vectorized or declarative transformations. Naive translation tends to recreate SAS “state machines” using loops, producing code that may run but is not idiomatic or easily testable. Finally, missing-value semantics differ. SAS procedures often ignore ‘.’ by default in summaries, while R functions frequently propagate NA unless na.rm=TRUE is specified (Cheng et al., 2025). This is especially sensitive for “last non-missing baseline” rules. Together, these mismatches explain why direct AI translation often yields syntactically correct but semantically fragile R code.

A direct SAS to R conversion is similar to literal human-language translation: it preserves surface structure but loses intent. When the output code mirrors the source language’s constructs (macros, implicit ordering, stateful data steps) rather than expressing the underlying requirement, the result often feels unnatural in the target language and is more prone to hidden logic drift. A reliable migration therefore needs a way to separate “what the program must do” from “how SAS happened to implement it.”

PROPOSED METHOD

OVERVIEW OF TWO-STEP PIPELINE

Because of these challenges, our proposed workflow decomposes SAS-to-R migration into two distinct phases that separate comprehension from implementation. In Step 1, a LLM ingests one or more SAS programs, and produces a structured design document that captures the program's purpose, inputs and outputs, transformation sequence, and explicit assumptions. This document becomes a concise, tool-agnostic specification that captures logic tied to SAS-specific semantics (e.g., BY-group ordering, missing-value precedence, and merge behavior). In Step 2, the LLM generates an R implementation directly from the reviewed design document, rather than from the original SAS code. This design-first approach focuses the model on "what" to build by giving clear requirements and constraints, before addressing "how" to express those requirements idiomatically in R.

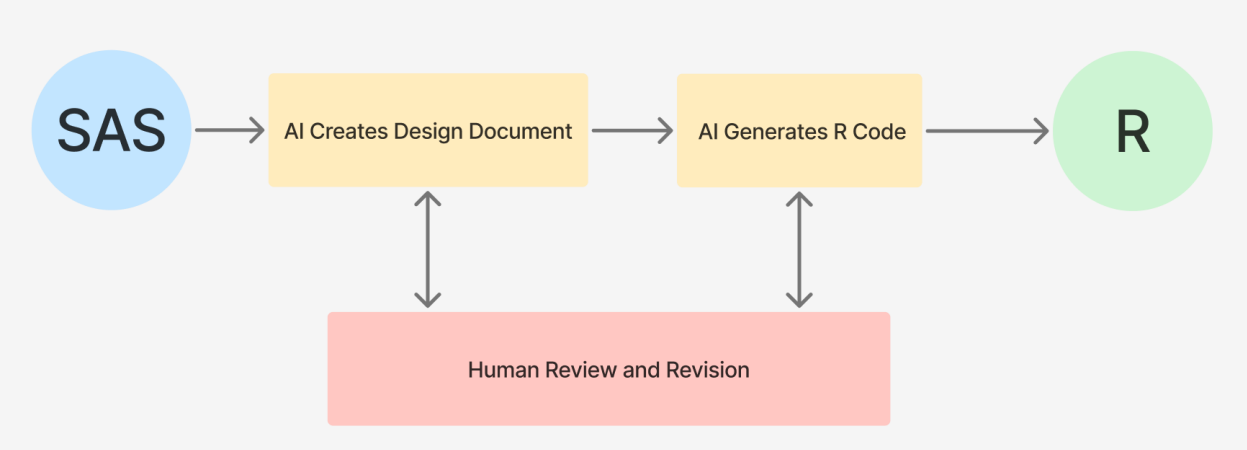

Figure 1. SAS-to-R Migration Workflow Using an Intermediate Design Document

DESIGN DOCUMENT

The design document is created directly from the structure and intent of the source SAS code. The LLM extracts and organizes what is present or implied to yield a pragmatic specification tailored to observed data flows, derivation steps, and assumptions. We propose the following criteria of what this document should cover. However, we believe this guide should serve as a recommended checklist, rather than enforced as a rigid template.

The document should state the program's purpose and scope, list input datasets, key variables, and formats, and report any preconditions such as sort order and expected uniqueness. It should define outputs with target dataset structures (e.g., one record per subject) and sorting keys. It should describe transformation steps as ordered operations reflecting SAS logic (e.g., sorts, BY-group processing, merges, retains, arrays, formats/informats, and first./last.), with dependencies and intermediate states noted. It should make ordering assumptions explicit, including sorting keys, sequences for baseline/last-value/LOCF selections, tie-breaking, and window boundaries. It should capture join keys and cardinality, document handling of duplicates and nonmatches, and specify missing-value behavior (numeric/character conventions, SAS missing ordering, imputation, defaults, partial dates).

```
Design prompt:
  Create a structured design document that captures my SAS program's purpose, inputs and
outputs, transformation sequence, and explicit assumptions. It should state the program's
purpose and scope, list input datasets, key variables, and formats, and report any
preconditions such as sort order and expected uniqueness. It should define outputs with
target dataset structures (e.g., one record per subject) and sorting keys. It should
describe transformation steps as ordered operations reflecting SAS logic (e.g., sorts,
BY-group processing, merges, retains, arrays, formats/informats, and first./last.), with
dependencies and intermediate states noted. It should make ordering assumptions explicit,
including sorting keys, sequences for baseline/last-value/LOCF selections, tie-breaking,
and window boundaries. It should capture join keys and cardinality, document handling of
duplicates and nonmatches, and specify missing-value behavior n(numeric/character
```

```
conventions, SAS missing ordering, imputation, defaults, partial dates). Use common human
language, not coding specific jargon. My SAS code is: [SAS code pasted]
```

Prompt 1. Example design prompt used to transform SAS code into a structured intermediate design document.

```
Code generation prompt:
Using only this template (this template is the only truth), generate R code that can
perform the functions described in the template
```

Prompt 2. Example code-generation prompt used to generate R code from the reviewed design document.

HUMAN REVIEW

Review activity is still needed but its' intent is to confirm requirement alignment, rather than to perform line-by-line code QC. Human review can assess whether the design document accurately captures the clinical and statistical intent of the original SAS program, with emphasis on logic that is prone to divergence between SAS and R. A structured checklist can guide this review, and include ordering assumptions, join keys and cardinality expectations, missing value handling and imputation, and deduplication rules. Where ambiguities exist, reviewers should annotate clarifications or decisions directly in the document. The outcome is a reviewed design document that serves as the "single source of truth", enabling traceable implementation and reducing the need for repeated deep dives into legacy SAS code.

APPLICATION

To illustrate the proposed workflow, we applied it to a relatively straightforward SAS macro that profiles SDTM dataset metadata and generates an Excel summary file. In this setting, the intermediate design document was useful for clarifying the program objective, expected inputs and outputs, and the sequence of processing steps before R code generation. In this case, both migration approaches produced broadly similar final results. However, the design-document workflow captured the intended logic more quickly, requiring only one iteration in this example, whereas the direct translation approach needed three follow-up prompts to clarify the expected behavior. Human review remained necessary for implementation details such as folder paths, file locations, and other minor environment-specific adjustments. Overall, this case serves as a practical illustration of how the proposed design-first workflow can be applied in a real migration setting before evaluating its broader performance across examples.

```sas
proc datasets library=lptuser kill; run; quit;

data sdtmnames;
     length fref $8 fname $200;
     did = filename(fref, "&protpath/datasdtm");
     did = dopen(fref);
     do i=1 to dnum(did);
         fname = dread(did,i);
         if find(fname, ".sas7bdat") then output;
     end;
     did = dclose(did);
     keep fname;
 run;

proc sql noprint;
     select substr(fname, 1, find(fname,".")-1) into :sdtmlist separated by ' ' from sdtmnames;
 quit;
 *%put &sdtmlist.;

 %let sdtmnum = %eval(%sysfunc(countc(&sdtmlist., ' ')) +1);
 *%put &sdtmnum.;


 /************ INFO ON SDTM DATASETS RECEIVED ***************/
%macro getSDTMinfo;
     data allsdtmdom_info;
         format DOMAIN $8. DSLABEL $200.;
         call missing(of _ALL_);
         stop;
     run;

     data allsdtmvar_info;
         format DOMAIN $8. VARNUM 8. VARNAME $40. VARLABEL $200.;
         call missing(of _ALL_);
         stop;
```

Code 1. Original SAS Program

Design Document – SDTM Summary and Consistency Check Program

1. Purpose and Scope

Purpose

This program scans a folder of SDTM datasets and produces:

1. A metadata summary of all SDTM domains and variables.
2. A summary of key standard variables across domains (e.g., EPOCH, timing variables, test variables, treatment variables).
3. Simple quality/consistency checks for:
   - Missing coding (MedDRA, WHODrug) in AE, MH, CM.
   - Consistency of death-related dates across DM, DD, DS, and AE.
4. An Excel workbook that consolidates all summaries and checks into separate worksheets for review.

Scope

- Input: All SDTM `.sas7bdat` datasets located in the specified SDTM directory (`&protpath/datasdtm`) plus some specific domains from the SDTM library `lptss` (e.g., DM, EX, SV, AE, MH, CM, DD, DS).
- Output:
  - A set of summary datasets stored in the `lptuser` library.
  - A metadata dataset describing the summary datasets.
  - An Excel file (`SDTMsummary_&sysdate..xlsx`) with one worksheet per summary/check.

The program is exploratory/diagnostic and does not modify any SDTM source data. It is intended for data review, structure understanding, and basic consistency checks.

2. Environment and Libraries

2.1 Libraries

- `lptuser`
  - Location: "`&protpath/userexploration/zhangam3/outputs`"
  - Role: Output library. All intermediate and final summary datasets are written here. The library is cleared at the beginning using `proc datasets ... kill`.
- `lptss`
  - Not explicitly assigned in this scrip umed pre-assigned externally).

Type a message

Figure 2. AI-Generated Design Document

```r
############################################################
# 2. Discover SDTM datasets and load them
############################################################

# List all .sas7bdat files in sdtm_dir
sdtm_files <- list.files(sdtm_dir, pattern = "\\.sas7bdat$", full.names = TRUE)
sdtm_names <- basename(sdtm_files)
sdtm_base  <- file_path_sans_ext(sdtm_names)  # domain names without extension

# Read all SDTM domains into a named list
# Names: AE, DM, EX, etc.
sdtm_list <- map(sdtm_files, ~ as.data.frame(read_sas(.x)))
names(sdtm_list) <- toupper(sdtm_base)

# Helper: get domain data safely
get_domain <- function(dom) {
  dom <- toupper(dom)
  if (!dom %in% names(sdtm_list)) return(NULL)
  sdtm_list[[dom]]
}
```

```r
############################################################
# 3. Output container
############################################################

outputs <- list()

############################################################
# 4.  Domain-level and Variable-level summaries
############################################################

# 4.1 Domain-level summary: DOMAIN, DSLABEL, NOBS
allsdtmdom_info <- map2_df(
  sdtm_list,
  names(sdtm_list),
  ~ {
    df <- .x
    dom <- .y
    # No MEMLABEL in R; use a simple placeholder or NA
    data.frame(
      DOMAIN  = dom,
      DSLABEL = NA_character_,
      NOBS    = nrow(df),
      stringsAsFactors = FALSE
    )
  }
)
```

Code 2. Migrated R code

RESULTS AND COMPARISON

We compared two AI-assisted SAS-to-R migration workflows: direct SAS-to-R translation and a two-step design-first workflow using an intermediate design document. The evaluation focused on three criteria: X, the number of conversational iteration rounds per migration file; Y, the proportion of AI-generated code requiring manual revision; and Z, the level of agreement between the final R output and the original SAS reference results. Across the evaluated cases, the direct translation workflow required an average of 3 conversational rounds per file, whereas the design-document workflow required 2 rounds on average. The design-document workflow also substantially reduced post-generation editing: 55.2% of AI-generated lines required manual revision under direct translation, compared with only 38.4% under the design-first approach. In addition, the design-document workflow achieved higher output fidelity, with the final R implementation reaching 80.3% agreement with the SAS reference results, compared with 64.5% for direct translation.

Metric X was calculated by counting assistant responses in each conversation log that contained fenced R code blocks, using these code-producing responses as a proxy for iterative code-generation rounds. Workflow-level values were then summarized as the average number of such rounds across the test cases.

Metric Y was calculated by extracting the R code from the last AI-generated response containing code and comparing it with the final human-edited R file. After removing blank lines and pure comment lines, the percentage of AI-generated lines not retained verbatim in the final code was recorded as the manual revision rate. This metric reflects how much cleanup or correction was needed after the AI-generated draft.

Metric Z was calculated as output fidelity, defined here as the percentage of cell-level agreement between the final R output and the SAS reference output. These values were obtained from the smart fidelity comparison step, which summarized agreement for each case and workflow in a separate fidelity report. Averaged across cases, the higher Z value for the design-document workflow indicates that making program assumptions explicit before code generation improved alignment with the original SAS logic. Taken together, these findings suggest that the intermediate design document improved migration efficiency, reduced manual rework, and produced R outputs that more closely matched the SAS reference results.

CONCLUSION

LESSONS LEARNED

The main benefit of the design-document workflow was improved fidelity to SAS reference outputs, along with a substantial reduction in manual code revision. The workflow was especially valuable when the original SAS logic contained hidden assumptions that needed to be made explicit before implementation, such as ordering rules, join behavior, and missing-value handling. Improvements were also observed even without substantive manual revision of the design document, suggesting that the intermediate design layer itself contributes value by making program intent more explicit before R code generation.

FUTURE IMPROVEMENTS

Future work should evaluate the workflow on a broader range of SAS programs. It will also be important to assess the approach on larger and more complex macro libraries, especially where macro dependencies span multiple helper libraries and shared utility components. This will help determine how the same design-first idea can be extended from single-program conversion to higher-level system migration. In addition, reusable Agent Skills or similar scalable mechanisms could be developed to reproduce this workflow more efficiently, with the long-term goal of enabling AI to iteratively refine migrated code until it reaches a version that matches the SAS reference output.

REFERENCES

Ling, C., & Wang, Y. (2025). Writing SAS MACROs in R? R functions can help! In Proceedings of the PharmaSUG 2025 Conference (Paper AP-078), San Diego, CA. PharmaSUG. https://pharmasug.org/proceedings/2025/AP/PharmaSUG-2025-AP-078.pdf

Danner, B. J., & Sarkar, I. (2023). Repetitive analyses in SAS: Use of macros versus data inflation and BY group processing. In Proceedings of the PharmaSUG 2023 Conference (Paper QT-046), San Francisco, CA. PharmaSUG. https://pharmasug.org/proceedings/2023/QT/PharmaSUG-2023-QT-046.pdf

Cheng, J., Malipeddi, S., Veeravel, G., Yang, J., & Sanjee, S. R. (2025). GenAI assisted code conversion: From SAS to R standard ADaM templates. In Proceedings of the PharmaSUG 2025 Conference (Paper AI-239), San Diego, CA. PharmaSUG. https://pharmasug.org/proceedings/2025/AI/PharmaSUG-2025-AI-239.pdf

Bosak, D. J., & Varney, B. (2025). Translation from SAS to R using ChatGPT. In Proceedings of the PharmaSUG 2025 Conference (Paper AI-242), San Diego, CA. PharmaSUG. https://pharmasug.org/proceedings/2025/AI/PharmaSUG-2025-AI-242.pdf

ACKNOWLEDGMENTS

The authors would like to thank the management teams at Merck & Co., Inc., Rahway, NJ, USA, as well as Huei-Ling Chen and Jeff Cheng, for their advice on this paper/presentation.

CONTACT INFORMATION

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

Junze Zhang
Merck & Co., Inc., Rahway, NJ, USA
E-mail: junze.zhang@merck.com

Amy Zhang
Merck & Co., Inc., Rahway, NJ, USA
E-mail: amy.zhang2@merck.com

Any brand and product names are trademarks of their respective companies.