Paper ML11

Leveraging AI for Seamless SAS to R Migration: Challenges and Opportunities

Dr. Milena Kraus, HMS Analytical Software, Heidelberg, Germany
Corinna Stöckinger, Boehringer Ingelheim, Biberach an der Riß, Germany
Dr. Robert Bauer, HMS Analytical Software, Heidelberg, Germany

ABSTRACT

While in the past SAS has been the de facto standard for the statistical analysis of clinical trials, in recent years a growing portion of the industry has had their eyes on open-source alternatives. Arguably the most popular is the programming language R. To support Boehringer Ingelheim's ClinPharm task force for the transition from SAS to R a team of experts from HMS was brought in.
Here, we present our joint effort for the migration of a collection of SAS macros and templates in the context of PK/PD analyses into R using GenAI to streamline the process. We discuss the different approaches we piloted including direct translation as well as utilizing existing packages of the pharmaverse and the pros and cons of each approach.
Finally, we discuss both the advantages and limitations of using GenAI in this context and the need for a guiding framework to successfully leverage its capabilities.

INTRODUCTION

For decades, SAS has been the cornerstone of statistical analysis in clinical trials, offering a robust and validated environment for regulatory-compliant workflows. Its widespread adoption across the pharmaceutical industry has led to the development of extensive legacy codebases, standardized templates, and macros tailored to clinical pharmacology (ClinPharm) and pharmacokinetics/pharmacodynamics (PK/PD) analyses.

However, the landscape is shifting. The rise of open-source programming languages—most notably R—has introduced new possibilities for data science in clinical research. R offers a rich ecosystem of packages, active community support, and modern paradigms for data manipulation and visualization. These advantages, coupled with increasing cost pressures and the desire for more flexible, scalable solutions, have prompted many organizations to explore alternatives to SAS.

In response to this industry trend, Boehringer Ingelheim initiated a transition from SAS to R in several departments. This strategic initiative is supported by HMS Analytical Software GmbH, leveraging their expertise in generative AI and migration strategies.

The primary objective of this collaboration is to migrate a large collection of SAS macros to R. The code base is currently used to prepare analysis datasets ready to use for noncompartmental analyses (NCA), Population PK (popPK) analyses and bioinformatic analyses, such as gene expression analysis. Here, R's extensive package ecosystem offers opportunities to reduce technology breaks and promises instant, but robust implementation of latest statistical models.
To accelerate and streamline the migration process, the team explored the use of Generative AI (GenAI) tools, including large language models, to assist in code translation, refactoring, and documentation.

The rest of the paper details the practical aspects of the migration. We begin by outlining the SAS codebase targeted for translation, including its structure and functional scope. Next, we describe the R-based environment where the translated code will be deployed, followed by a comparison of relevant R frameworks and the rationale behind our selection. We then present the structured, GenAI-assisted migration workflow, highlighting key roles, tools, and strategies used. We present the different approaches to GenAI-assisted translation and how they informed our tooling for automated batch migration of existing code bases to open-source frameworks like R or python.
Finally, we discuss the challenges faced during the project and highlight some of the future developments and opportunities for code migration.

WHY ARE WE MOVING?

Beyond the general advantages of open-source software, the current migration from SAS to R in the context of PK/PD analyses addresses several domain-specific challenges. Currently, the processing steps after the creation of analysis datasets involve multiple software solutions—for example, dataset preparation for NCA is done in SAS, while parameter calculation is performed in Phoenix® WinNonlin, and result presentation is again handled in SAS. This back-and-forth introduces inefficiencies and complexity. Transitioning to R offers the potential to streamline the entire workflow from SDTM datasets to the generation of Tables, Listings, and Figures (TLFs) and reduces technology breaks in general.

Additionally, pharmacometric analyses such as population PK modeling and bioinformatic workflows like gene expression or Quantitative Systems Pharmacology (QSP) modeling require advanced statistical methods where SAS reaches its limits. R's extensive package ecosystem, which evolves rapidly with methodological advancements, makes it a natural choice for these analyses. Aligning data preparation and analysis within R also facilitates smoother collaboration between neighboring departments.

Finally, the migration to R opens new possibilities for AI integration beyond translation. The translated templates can serve as structured input for AI tools designed to support daily work—so-called agentic workflows. In this context, R code is significantly more compatible with modern AI tooling than SAS code, further reinforcing the strategic value of the transition.

WHAT ARE WE MOVING?

The SAS codebase targeted for migration consists of approximately 20,000 lines of code, structured into around 50 macros and 30 templates. These components are used to transform SDTM datasets into analysis-ready datasets for a variety of pharmacological and bioinformatic workflows.

The functional scope of the code base spans several analytical domains:

- PreADS: Prepares relevant SDTM domains, including date transformations and derivation of baseline covariates.
- PMx: Creates datasets for population pharmacokinetics (popPK) analyses, similar to CDISC ADPPK standards and NONMEM structures.
- PK ADS: Generates datasets for pharmacokinetic, pharmacodynamic, and biomarker analyses (similar to CDISC ADNCA), calculates NCA parameters (similar to CDISC PP), and includes immunogenicity datasets (ADIS).
- PBPK: Supports physiologically based pharmacokinetic modeling, building on PK ADS structures.
- CBSP: Enables clinical bioinformatics and systems pharmacology workflows, including gene and protein expression analyses.

These workflows involve a mix of data derivation, parameter calculation, and data preparation for downstream statistical modeling. While some transformations are relatively straightforward (e.g., date conversions), others require domain-specific logic and integration with modeling tools.

Further we can differentiate two types of code:

- SAS Macros: These are reusable, study-independent components. They perform standardized tasks such as converting SDTM character dates to numeric formats or transposing CDISC findings datasets to extract baseline values. These macros are considered stable and should remain consistent across studies.
- SAS Templates: These are study-specific scripts that call the macros and generate the actual analysis datasets. Templates are adapted for each study and reflect the unique structure and requirements of the data. They include rich commentary and solutions for alternative study settings.

Understanding this distinction is crucial for migration planning. Macros can often be translated once and reused. At the same time, they comprise large sets of parametrization options and implement complex logic. Templates are less complex and are more workflow alike. The captured domain logic and alternative code snippets shall not be lost.

WHERE ARE WE MOVING?

With a clear understanding of the SAS codebase and its functional scope, the next step is to explore the target environment for the migrated R code. This includes the infrastructure where the new workflows will be deployed, as well as the R frameworks selected to support clinical and pharmacological analyses.

TARGET ENVIRONMENT: RISE

The translated R code will be deployed within RISE (Report Integrated Statistical Environment) [1], a cloud-based, statistical computing platform developed custom developed for Boehringer Ingelheim as summarized in Figure 1. RISE is inherently programming language agnostic and supports both SAS and R workflows. RISE provides a

comprehensive suite of tools to support programming in both SAS and R, offering integrated development environments—SAS Studio on Viya and RStudio Pro on Posit Workbench. The platform enables formal validation of analysis macros, organizes results into structured report formats, and automates the generation of PDF documentation for regulatory submissions. This infrastructure ensures that migrated code can be executed, validated, and maintained within a consistent and scalable framework.

Figure 1 Overview on RISE features.

TARGET R FRAMEWORK SELECTION

While RISE provides the infrastructure, the migration effort also required selecting the appropriate R programming paradigm. R offers multiple frameworks for data handling and analysis, each with distinct characteristics:

R Base
- Procedural syntax similar to SAS
- Efficient handling of data frames
- Extensive statistical and graphical methods and resources
- Suitable for direct translation of macros and templates with minimal refactoring

Tidyverse
- Modern, composable grammar for data manipulation
- Emphasizes tidy data principles: functional, type-stable, verbose, tidy data, consistent, composable, expressions
- Includes packages like dplyr, tidyr, stringr, and readr
- Improves readability and modularity, but requires some upskilling

Pharmaverse (e.g., admiral)
- Ecosystem of R packages tailored for clinical trial reporting
- CDISC-compliant workflows for ADaM dataset creation
- Incorporates domain knowledge into reusable functions
- Templates and user guides (e.g. workflow for creating ADNCA, ADPPK)
- Still evolving, but strategically aligned with industry standards

The choice of framework significantly affects the structure and usability of the translated code. For example, admiral, part of the pharmaverse ecosystem, is the R package that most closely mirrors the purpose of the original SAS codebase, particularly in its handling of CDISC-compliant workflows. On the other hand, following the tidy design principles with admiral will not result in R code that is very similar to the original SAS code. These frameworks are interoperable and compatible, allowing selective integration based on the needs of each macro or template. For example, R Base may be used for close adherence to SAS logic where needed, tidyverse for structured data manipulation, and the admiral package for CDISC-compliant derivations.

A simple example—flagging duplicate records—illustrates the differences of the frameworks in Listing 1. In R Base, identifying duplicate records can be done with a simple one-liner using the `duplicated()` function, which closely

mirrors the logic used in SAS. The tidyverse approach, while requiring helper variables and multiple steps, offers greater flexibility and clarity in handling duplicates. Admiral, on the other hand, treats duplicates as data quality issues, issuing warnings and providing diagnostic tools to support validation and compliance.

R Base

```
Data$IsDuplicate <- duplicated(data[sortkey])
```

- One line of code
- Assumes proper data/matching index

Tidyverse

```
data %>%
dplyr::distinct(sortkey, keep_all = TRUE)
```

- Tidyverse assumes duplicates should be removed and recommends to use distinct().
- Creating a duplicate flag requires helper structures.

```
data %>%
dplyr::arrange(dplyr::across(dplyr::all_of(sortkey))) %>%
    dplyr::group_by(dplyr::across(dplyr::all_of(sortkey))) %>%
dplyr::mutate(rn = dplyr::row_number()) %>%
dplyr::mutate(isDuplicate = dplyr::case_when(rn > 1 ~
1, .default = NA)) %>%
dplyr::select(-rn) %>%
dplyr::ungroup()
```

Admiral

```
admiral::signal_duplicate_records(adsl,
admiral::exprs(sortkey), cnd_type = "warning")
```

- Admiral expects duplicates to be erroneous data and throws an exception.
- Provides helpers to locate duplicates.

Listing 1 Comparison of duplicate handling in considered target frameworks R Base, tidyverse and admiral.

We summarized our thorough comparison of a R Base/tidyverse approach to one using Admiral across several dimensions in Table1. R Base/tidyverse offers high similarity to existing SAS code, making them easier to adopt with lower upskilling requirements and medium refactoring needs. They provide straightforward debugging and customization options, though they offer limited benefits from the open-source community and do not enforce CDISC standards. Admiral, in contrast, requires more substantial refactoring and upskilling but brings strong alignment with CDISC standards and greater support from the open-source community. It treats data quality more rigorously and offers diagnostic tools, making it suitable for standardized, compliant workflows. While Admiral has higher implementation effort, it holds promise for future cross-industry applications. In essence, the R Base/tidyverse approach aligns with a “migration” strategy—preserving existing logic with minimal changes—whereas the admiral-based approach represents a “refactoring” strategy, requiring much deeper restructuring. Given these trade-offs, the team opted for a migration approach centered on R Base and tidyverse, with selective use of Admiral functions where they add clear value.
This strategy ensures high recognition value for SAS-trained users, minimizes upskilling requirements, and maintains flexibility for future enhancements.

	R Base/tidyverse	Admiral
Similarity to existing SAS code base	High	Low
Inherent refactoring needs	Medium	High
Upskilling effort	Low	Medium
Open-source community benefits	Low	High

Debugging output/program insights	Easy	Possible
Templates/customization	Easy	Medium
Future perspectives/cross industry	-	Potential given
Adherence to CDISC standard	Not enforced	High
Benefit of AI for translation	High	Medium
Overall implementation effort	Medium	High
	= “Migration”	= “Refactoring”

Table 1 Comparative overview of R Base/tidyverse and Admiral approaches across key implementation dimensions.

HOW ARE WE MOVING?

Leveraging artificial intelligence for code migration promises speed, scalability, and reduced manual effort. Large Language Models (LLMs) offer a natural interface for this task: simply ask the model to translate SAS macros or templates into R. This approach works surprisingly well for small, self-contained snippets when the primary goal is replicating basic functionality.

However, specialized LLMs were trained to generate code from specifications or natural language descriptions, not specifically for translating code. Additionally, SAS code is seldomly shared giving little available training data for LLMs. As a result, when LLMs are used without proper guardrails, the generated code often has significant shortcomings. Below is a non-exhaustive list of issues we observed with translating code from SAS to R using state-of-the-art LLMs in the context of clinical data analysis:

- Use of functions that do not exist in R (hallucinations)
- Reliance on exotic packages that are not well maintained, not validated and should therefore not be used
- Incorrect handling of complex SAS idioms such as FIRST./LAST. or RETAIN
- Ignoring table orderings or other output-related settings
- Lack of uniform style, naming, error handling, and data type conventions
- Absence of domain knowledge (e.g., PK/PD and ADaM/CDISC nuances)
- Translation without chunking the code into smaller pieces will overload the context and e.g. token limits and lead to incomplete, less focused translations
- Similar inputs can produce significantly different outputs, undermining the goal of a unified, maintainable, and reviewable code base.

However, the gap is not that LLMs “can’t translate code,” but that ungoverned translation does not meet our constraints around code quality, validation, and reproducibility.

Our approach therefore couples GenAI with a controlled process and human-operated guardrails that are enforced by migration strategies, well-scoped code decomposition and a multi-layered approach for testing and verification. These concepts will now be explained in more detail in the following sections.

ROLES

The migration process can be described as a collaborative, multi-role workflow to ensure quality, reproducibility, and domain alignment. Three key roles were defined:

- SE: Data Science Engineer/Programmer for SAS; familiar with the original SAS code base (not necessarily familiar with R)
- ME: Migration Engineer familiar with SAS and R that oversees the migration process; not necessarily someone that is already familiar with the original SAS code base as this role will become familiar with the SAS code during the project
- RE: Data Science Engineer/Programmer for R; familiar with R and the newly built code base (not necessarily familiar with SAS)

It is possible that the same person takes multiple roles.

EXPLORATORY SETUP

In our initial approach, most of the process steps were performed manually in a minimal, exploratory setup:

- Posit Workbench Pro [2]
- Ellmer R package for LLM usage and structured output [3]
- Custom R utilities to support iterative tasks, authentication and data/output comparison
- Boehringer Ingelheim's LLM endpoints including models claude sonnet 4 and gpt4.1
- Copy/paste of code and prompts

The setup ensured low maintenance and easy usage of the restricted BI environment.

WORKFLOW

The general process applied for migration is summarized in Figure 2 and includes:

1. Code decomposition
2. Prompt engineering and strategy selection
3. GenAI-assisted translation
4. Reassembly, integration and testing
5. Translation iteration and strategy update
6. Second level testing and finalization

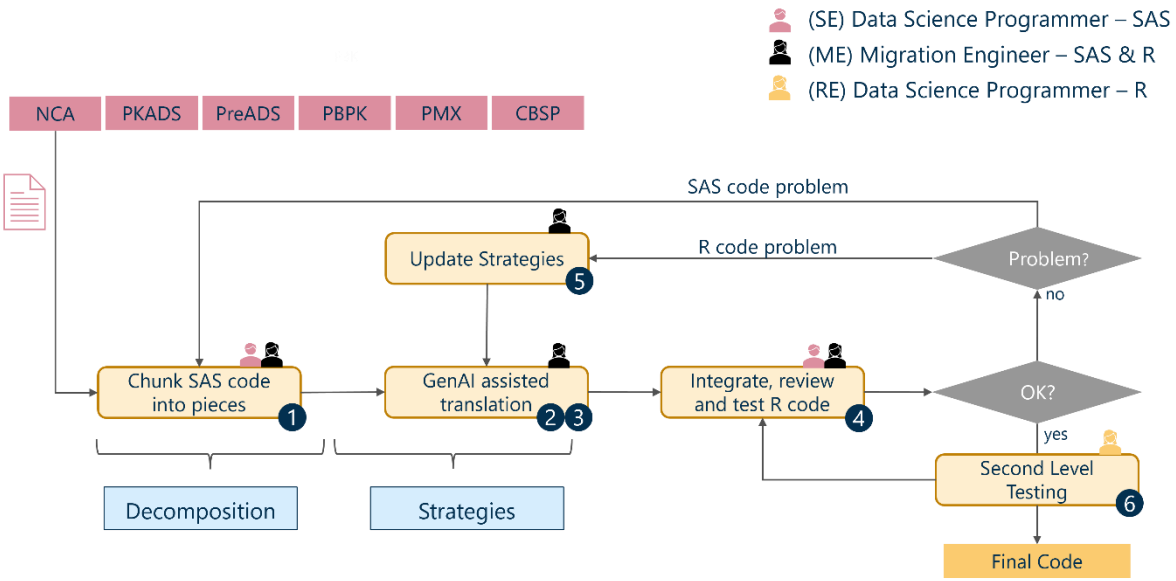

Figure 2 Migration workflow. 1 – Code decomposition; 2 – Prompt engineering and strategy selection; 3 – GenAI-assisted translation; 4 – Reassembly, integration and testing; 5 – Translation iteration and strategy update; 6 – Second level testing and finalization

CODE DECOMPOSITION

There are several reasons why breaking down complex code into smaller, well-defined chunks leads to higher-quality translations:

- Reduced Prompt Noise and Ambiguity: Small chunks simplify the context for the LLM, reducing cognitive load and improving determinism if similar code is translated.
- Enhanced Testability and Validation: Fine-grained units enable unit-level tests, comparisons to gold standard data, and step-by-step validation, which are essential for compliance and reproducibility.
- Incremental Execution and Caching: Supports incremental re-runs when only one chunk changes, improving efficiency and reducing processing time.

However, code decomposition introduces context breaks and the need for proper cross-chunk re-integration.

In a manual translation, ME decides whether a code snippet needs to be chunked. SE1 is consulted if deeper domain knowledge is necessary to make this decision.

PROMPT ENGINEERING AND STRATEGY SELECTION

Next to an appropriate chunk of code, the prompt i.e. the instructions for translation need to be concise and should not overload the context. Throughout the migration process the ME writes theses instructions, collects and groups them into “strategies” that can be reused and assembled dynamically based on the given SAS code chuck. For example, there are general instructions that apply to all translation tasks like “Use package::function() notation instead of `library()` calls” and very specific instructions only applying for specific code chunks. In Figure 3, we demonstrate the strategy on how we would like NAs to be handled in R. In that specific case, we instruct on using the admiral function to convert blanks to NA on load. Despite R’s capability to type NAs and the AI’s very strong urge to use typing when translating the SAS code, we instruct to not type NAs as it does not provide any advantages in our context. Most strategies include examples of SAS/R code to make the translation more reliable and concise. However, these examples bloat the prompt and should only be applied when needed.

```
TEMPLATE = """
<strategy_format_missing_values>
This strategy defines how to handle missing values (NAs).

After loading, convert blank strings to `NA` using `admiral::convert_blanks_to_na()`.
After that, assume NA's are not typed and ignore explicit formatting from SAS.
Typed NA's are not needed in R and will complicate further processing.

Example of NA conversion as it's done in admiral:
<r>
ds <- haven::read_xpt(rawdata_path) |> admiral::convert_blanks_to_na()
</r>

Example of not typing R NA'S despite SAS NA's always being typed:

<r>
data <- dplyr::mutate(data, VAR1 = NA) # Note how this is NOT NA_real_ or NA_character_
data %>% dplyr::filter(!is.na(VAR1)) # Note how this can be filtered with is.na() instead
</r>

<sas>
if compress ( lowcase ( studyid ) ) in ( "" " " 'n.a." "missing" "-" "." ) ....
</sas>
<r>
dplyr::if_else(!is.na(STUDYID), ... , NA)
</r>
```

Figure 3 Example of a prompt to handle missing data when translating SAS to R code

GENAI-ASSISTED TRANSLATION

The ME sends the code chunk and selected translation strategies to the LLM. In the manual process, the step is facilitated by the Ellmer R package [3] using the structured output function. Quick iterations are possible through the same chat objects of fresh setups, while keeping track of different versions. Together with a small custom utility framework coded in R the ME is enabled to work on the restricted BI environments for the initial exploration and build phase.

REASSEMBLY, INTEGRATION AND TESTING

Code decomposition requires reassembly into a complete R script, integration into the target code base and thorough testing.

SE prepares test studies and test data that are processed with the original code base to produce a gold standard data set for comparison. The ME performs a detailed step-by-step analysis between results created by the updated target code base and the original results using customized compare functions. See Figure 4 for an example of how the `pc2use` dataset is produced in SAS and the corresponding R code. The custom `test_against_gold()` function is based on the testthat R package and includes utilities to preprocess data produced by SAS for proper comparison. For example, NAs, labeling and order needs to be removed to compare intermediate datasteps and are controlled in a different procedure only for the final output of the script.

› SAS

```
DATA pc2use(DROP=param paramcd);
  SET pcncaex;
  IF INDEX(acexco,'ALL CALC')=1 AND acsamtyp='SP' THEN DO;
    aval=.;
    acexco='';
  END;
  IF INDEX(acexco,'HALF LIFE')=1 THEN acexco='';
RUN;
```

› R

```
pc2use <- pcncaex %>%
  # Remove PARAM and PARAMCD as they are not needed
  dplyr::select(-PARAM, -PARAMCD) %>%
  # 1)
  dplyr::mutate(
    AVAL = dplyr::if_else(stringr::str_starts(dplyr::coalesce(ACEXCO, ''), 'ALL CALC') & ACSAMTYP == 'SP', NA_real_, AVAL),
    ACEXCO = dplyr::if_else(stringr::str_starts(dplyr::coalesce(ACEXCO, ''), 'ALL CALC') & ACSAMTYP == 'SP', '', ACEXCO)
  ) %>%
  # 2)
  dplyr::mutate(
    ACEXCO = dplyr::if_else(stringr::str_starts(dplyr::coalesce(ACEXCO, ''), 'HALF LIFE'), '', ACEXCO)
  )
```

› Test for intermediate results in R

```
> test_against_gold(actual = pc2use, gold_path = pkadsin)
Column names do not match:

Error: `actual_df` (`actual`) not equal to `gold_df` (`expected`).

`actual` is length 72
`expected` is length 71

`names(actual)[1:3]`:            "STUDYID" "USUBJID" "SUBJID"
`names(expected)[1:4]`: "ACEXCO" "STUDYID" "USUBJID" "SUBJID"

`names(actual)[9:15]`:    "ACEVLINT" "ACTPTSP" "ACTPTSPN" "PARAMCD" "ACCAT" "ACTEST" "ACSPEC"
`names(expected)[10:15]`: "ACEVLINT" "ACTPTSP" "ACTPTSPN"           "ACCAT" "ACTEST" "ACSPEC"

`names(actual)[55:61]`:   "SRCSEQ" "ACTSTDTL" "ACEX" "ACEXCO" "SEX" "RACE" "AGE"
`names(expected)[55:60]`: "SRCSEQ" "ACTSTDTL" "ACEX"          "SEX" "RACE" "AGE"

`actual$PARAMCD` is a character vector
```

Figure 4 Example of a SAS code chunk and its R equivalent and how our custom test function highlights differences in SAS-generated data vs. R generated data.

Comparing SAS output still needs cumbersome manual comparison to console output in R as shown in Figure 5.

› SAS

When sorted by studyid acspec acsamtyp actest acgrpid actptn subjid, there are duplicates in concentrationfile (concds).
--> This needs to be resolved!

Beob.	Study Identifier	Specimen Material Type	Data Source Identifier	Pharmacokinetic Test Name - Analyte	PK Period	Analysis Timepoint (N) - Plan. Time	Subject Identifier for the Study
1	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10101
2	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10101
3	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10102
4	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10102
5	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10103
6	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10103
7	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10104
8	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10104
9	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10105
10	1234-5678	PLASMA	SP	BI 654321	3	-25.000	10105

› R

```
When sorted by STUDYID, ACSPEC, ACSAMTYP, ACTEST, ACGRPID, ACTPTN, SUBJIDN, there are duplicates in concentrationfile (gold_adpcattr).
--> This needs to be resolved!
# A tibble: 30 × 7
   STUDYID   ACSPEC ACSAMTYP ACTEST    ACGRPID ACTPTN SUBJIDN
   <chr>     <chr>  <chr>    <chr>       <dbl>  <dbl>   <dbl>
 1 1234-5678 PLASMA SP       BI 654321       3    -25   10101
 2 1234-5678 PLASMA SP       BI 654321       3    -25   10101
 3 1234-5678 PLASMA SP       BI 654321       3    -25   10102
 4 1234-5678 PLASMA SP       BI 654321       3    -25   10102
 5 1234-5678 PLASMA SP       BI 654321       3    -25   10103
 6 1234-5678 PLASMA SP       BI 654321       3    -25   10103
 7 1234-5678 PLASMA SP       BI 654321       3    -25   10104
 8 1234-5678 PLASMA SP       BI 654321       3    -25   10104
 9 1234-5678 PLASMA SP       BI 654321       3    -25   10105
10 1234-5678 PLASMA SP       BI 654321       3    -25   10105
```

Figure 5 Manual comparison of SAS and R outputs.

TRANSLATION ITERATION AND STRATEGY UPDATE

Especially in the initial phase, the translated code will not run or produce proper results. The ME iterates with the model until the output meets quality standards. In these cases, the ME can adapt the code directly but should always consider an update or new creation of a strategy. The strategy should then be tested and added to the strategy library for reuse, improving scalability and consistency.

FINALIZATION AND REVIEW

After rigorous testing, the ME creates a finalized version of the new R code. It is then handed off to the RE for second-level testing on previous, but for the ME unseen studies. This ensures robustness and generalizability. Feedback of the RE can also inform chunking, strategy update and testing.

PATH TO AUTOMATION

The exploratory setup outlined in the previous section forms an effective translation process with high quality and reproducible results. However, it still requires substantial manual work from the ME. To enhance scalability and minimize manual effort, moving toward automated migration is critical. For this purpose, we leveraged a support tool referred to as “Code2X” that is internally developed by HMS to accelerate development activities of HMS migration engineers in the context of AI-assisted code migration. The tool can be seen as a set of support scripts and concepts interconnected via a database and a specialized graphical user interface. The basic principles used under the hood,

however, can be easily reproduced individually for similar automation setups. In this project, it was applied to assist with certain aspects of the migration workflow, including decomposition, strategy management, and batch processing.

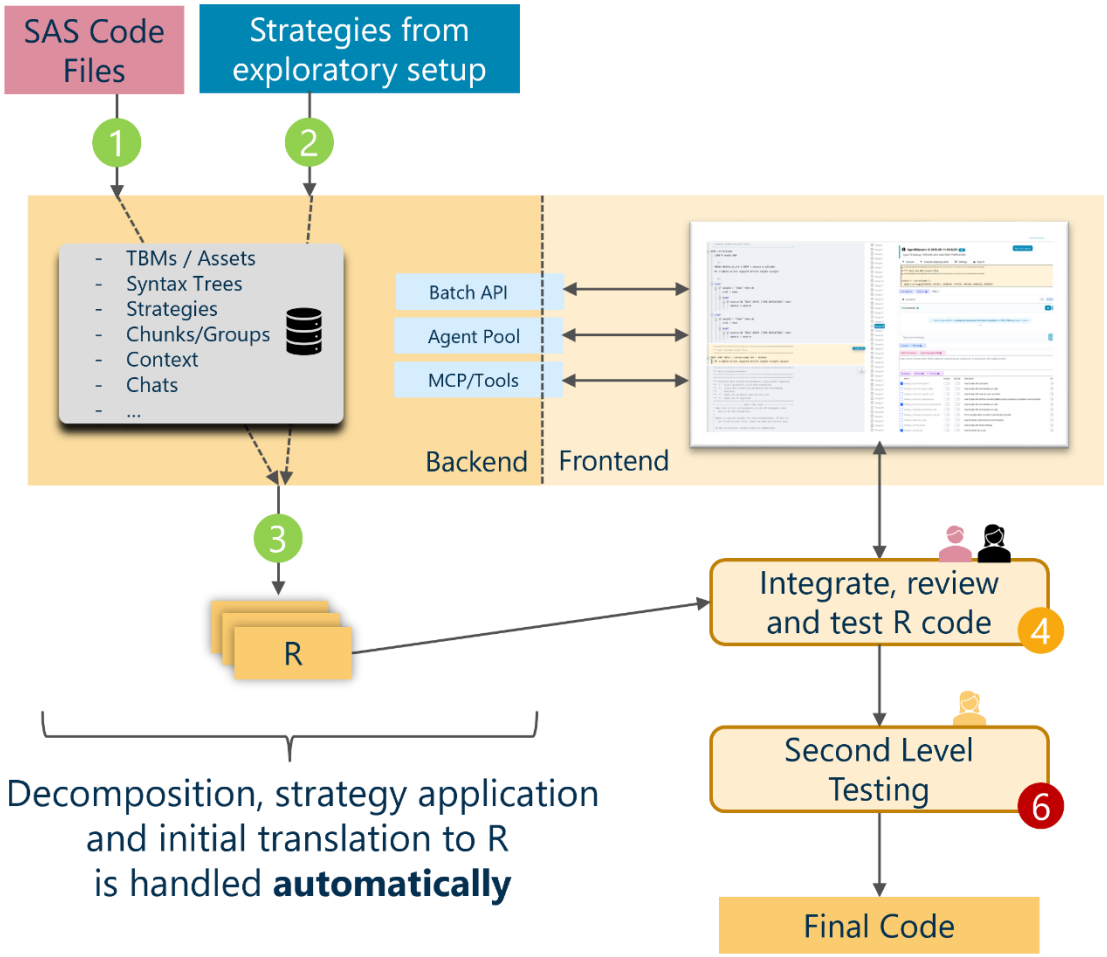

Figure 6 Code2X-assisted automated migration process

The Code2X assisted setup is shown in Figure 6. In step (1), the SAS source files are first organized in TBMs. TBM stands for “To-Be-Migrated” and represents a group of programs that need to be translated together. In our case, each analytical domain (PreADS, PMx, etc.) is represented as two separate TBMs: One for the macros and another one for the templates. This structure does not only help with organizing the tasks but also allows it to specify strategies and system prompts on a per-TBM basis. Next, all source files inside a TBM are analyzed with a special parser to build Abstract Syntax Trees (ASTs) for each SAS program. TBMs and ASTs are stored in a database alongside all relevant metadata.

In step (2), the strategies found in the exploratory phase are added to the mix. Within Code2X, migration strategies are wrapped in a well-defined class structure with the core purpose of the strategy (what to do) and a selection function implemented as python code (when to apply). The latter can utilize code inspection based on the AST and is crucial because it allows the system to select or un-select strategies automatically. An example of a selection function is shown in Listing 2. This example implements a strategy in the context of PROC SQL and will only return true (=strategy is applied) if at least one of the code elements in the selected chunk is a CREATE TABLE statement.

```
def selection_function(ast_model: AST_MODELS.ASTModelGeneric, future_state: dict[str, Any]) -> bool:
    return len(ast_model.get_all_nodes_of_type(node_type=AST_MODELS.ASTModelSqlCreateTable)) > 0
```

Listing 2 Code example of a selector function.

Based on the ASTs and the pre-built strategies, the system creates a proposal for the code decomposition and an initial translation of the SAS code in step (3). Unlike before, this happens fully automatically and covers a large portion of the manual effort from the first three steps from the exploratory phase and therefore substantially reduces effort and time required for migration.

To support the interactive steps that follow for integration and review, we use a graphical user interface provided by the Code2X toolbox that is shown in Figure 7.

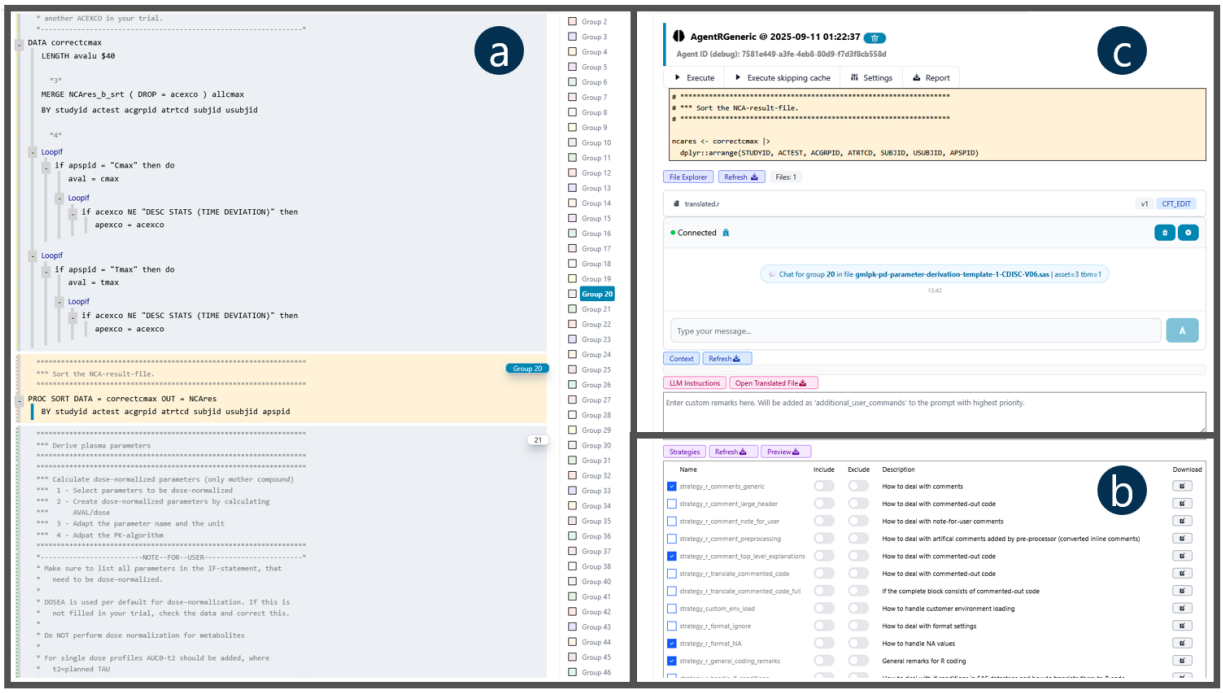

Figure 7 Screenshot of the Code2X migration tool UI

This GUI streamlines some of the steps that we identified as cumbersome in the exploratory phase:

a. The chunk view keeps track of the chunks / groups in the current decomposition. The ME can adjust the decomposition on the fly and select individual chunks for closer inspection. Details for the selected chunk are shown in (b) and (c).
b. The strategy view shows available strategies and whether they are applied to the selected chuck. The ME can edit the strategies and overwrite decisions made by the system, i.e., force or deny application of specific strategies.
c. The output view shows a preview of the translated code and gives the ME access to chat and agent-based interactions with the selected code chunk.

There are several other views not visible here, i.e., for managing additional context that is used in the translation process. GUI-based interaction helps with fast iteration cycles and makes the life of the ME easier because mapping between files, chunks, strategies and prompts is handled automatically and it is no longer necessary to manually copy/paste snippets into full scripts.

CONCLUSION

The migration initiative has not only demonstrated the feasibility of GenAI-assisted code translation but also highlighted how GenAI can reshape and enhance clinical data workflows in general. Our phased approach, starting with exploration and advancing to structured automation, has played a key role in optimizing the process. Future enhancements will focus on enabling direct code execution, automated testing, and intelligent strategy generation – ultimately supporting large-scale, reproducible batch migrations.

However, several challenges remain. Ensuring semantic equivalence between SAS and R outputs requires rigorous testing, especially in domains like PK/PD where subtle differences in logic can have regulatory implications. Maintaining consistency across translations, avoiding hallucinations from GenAI, and managing domain-specific nuances such as CDISC standards or SAS idioms (e.g., RETAIN, FIRST./LAST.) demand a robust framework of guardrails and human oversight. Additionally, integrating AI-generated code into validated environments like RISE requires careful attention to compliance, traceability, and documentation standards.

Despite these hurdles, the opportunities are substantial. The migration has fostered cross-functional collaboration between ClinPharm, TMCP, and IT teams, encouraged the adoption of tidy data principles and functional code design, and accelerated knowledge transfer across roles. GenAI opens the door to new use cases, such as adapting templates to study protocols using structured inputs like study plans—an area where R's flexibility offers clear advantages over SAS.

Looking ahead, the combination of human expertise, structured migration strategies, and evolving GenAI capabilities positions the team to not only complete the SAS-to-R transition but also to lead innovation in clinical data programming. The lessons learned here will inform future initiatives, including the development of AI agents for study-specific customization and broader adoption of open-source standards across the industry.

DISCLAIMER

This paper was developed with the assistance of AI-based tools, specifically ChatGPT by OpenAI (GPT-4 and GPT-5), which supported the authors in refining language and enhancing clarity. AI was not used to generate original content or ideas. All AI-generated content was reviewed and verified by the authors, who remain fully responsible for the final output.
Code2X is not a commercially available product but is developed for and used in-house at HMS.

REFERENCES

[1] "Development of a Customised Statistical Computing Environment for Clinical Trial Data Analysis," [Online]. Available: https://www.analytical-software.de/ad05-phuse2024-presentation_24oct2024/.

[2] "Posit Workbench Pro," [Online]. Available: https://posit.co/products/enterprise/workbench/.

[3] "Ellmer R package," [Online]. Available: https://ellmer.tidyverse.org/.

CONTACT INFORMATION

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

Author Name: Milena Kraus
Company: HMS Analytical Software GmbH
Address: Grüne Meile 29, DE-69115 Heidelberg
Work Phone: +49 6221 6051-247
Email: milena.kraus@analytical-software.de
Website: https://www.analytical-software.de/