March 2026

ML12

Using AI to Convert ADaM™ Specifications to R Code

Alice Ehmann, GSK, Collegeville, PA, USA
Kjersten Offenbecker, GSK, Collegeville, PA, USA

gsk.com

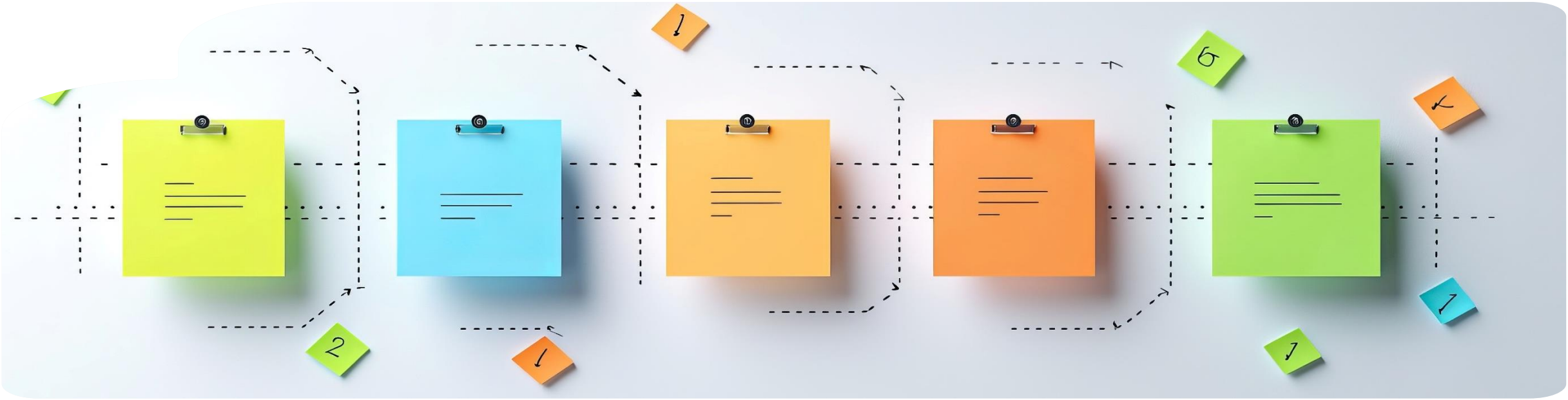
Introduction

- ADaM™ datasets are foundational to clinical trial analysis
- Industry transition from SAS to R is accelerating
- Many programmers face a steep R learning curve
- AI offers a way to accelerate development without lowering rigor

Methods

Our approach is built on four core components:

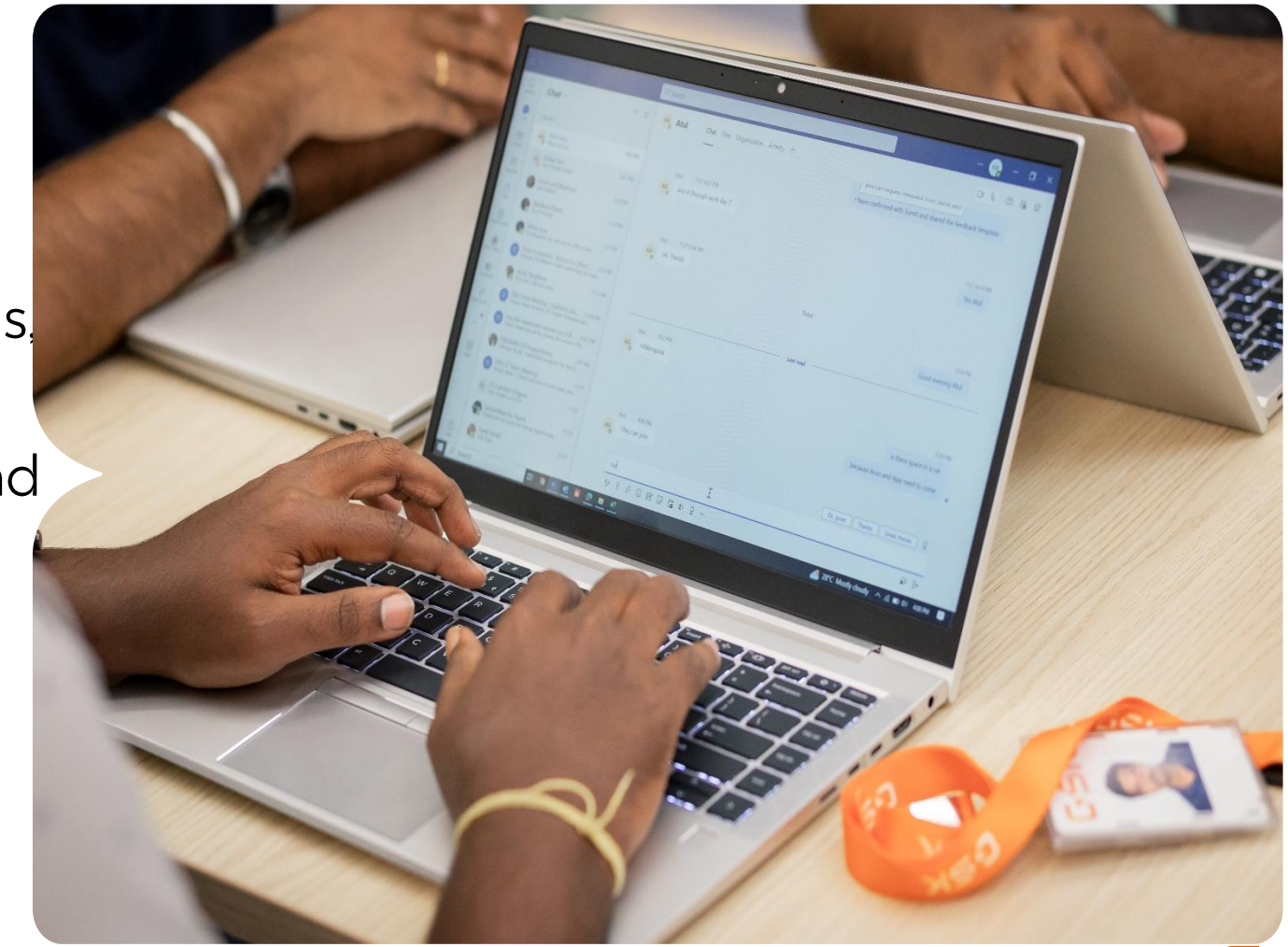
1. ADaM™ Specification Creation
2. Prompt Generation
3. The Generation Engine
4. Human-in-the-Loop Review



Methods

ADaM™ Specification Creation

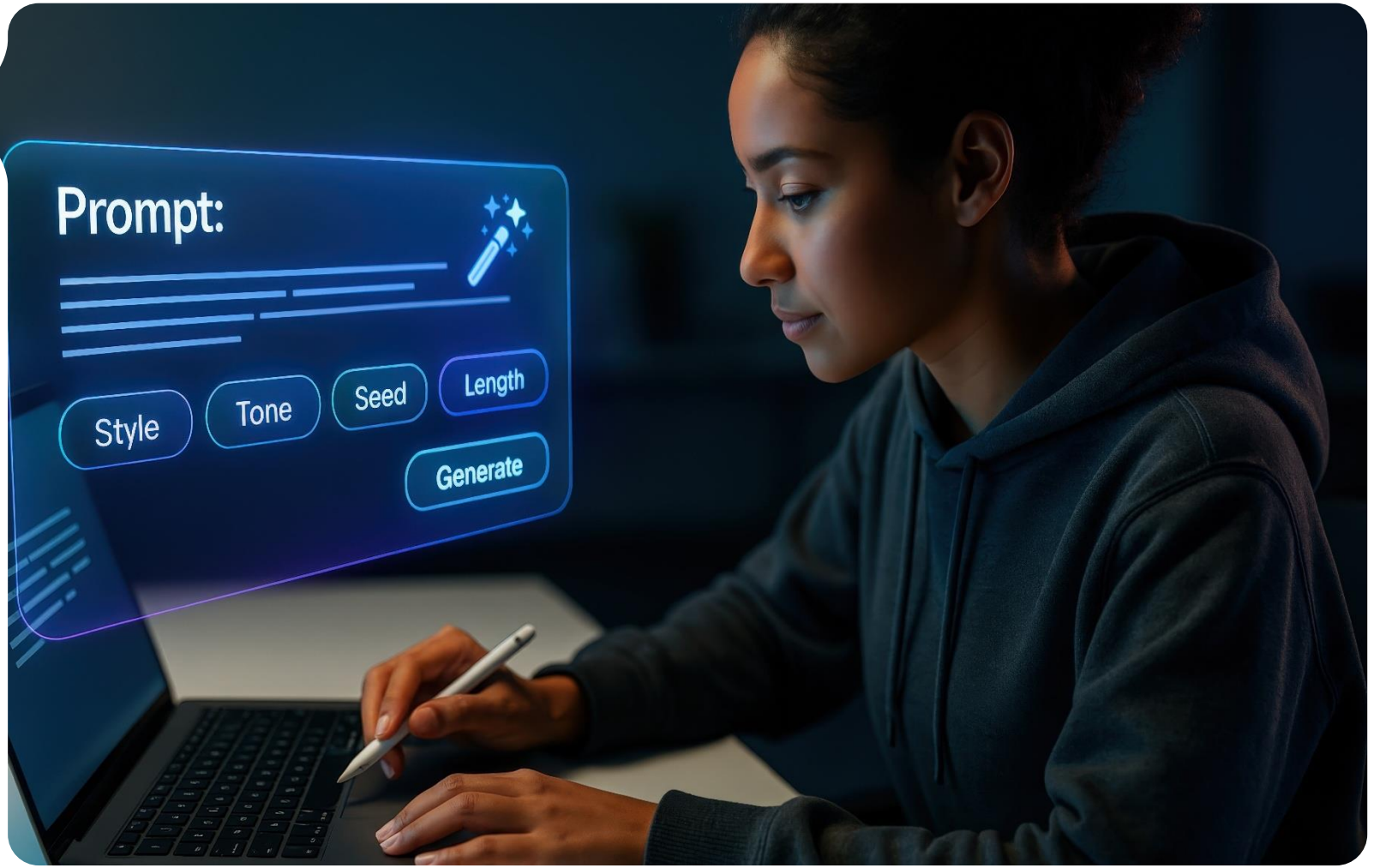
- Specifications are language-agnostic and P21™-based
- Explicit derivations, assumptions, and edge cases
- Clear source variables, joins, and keys
- Treated with the same rigor as production code



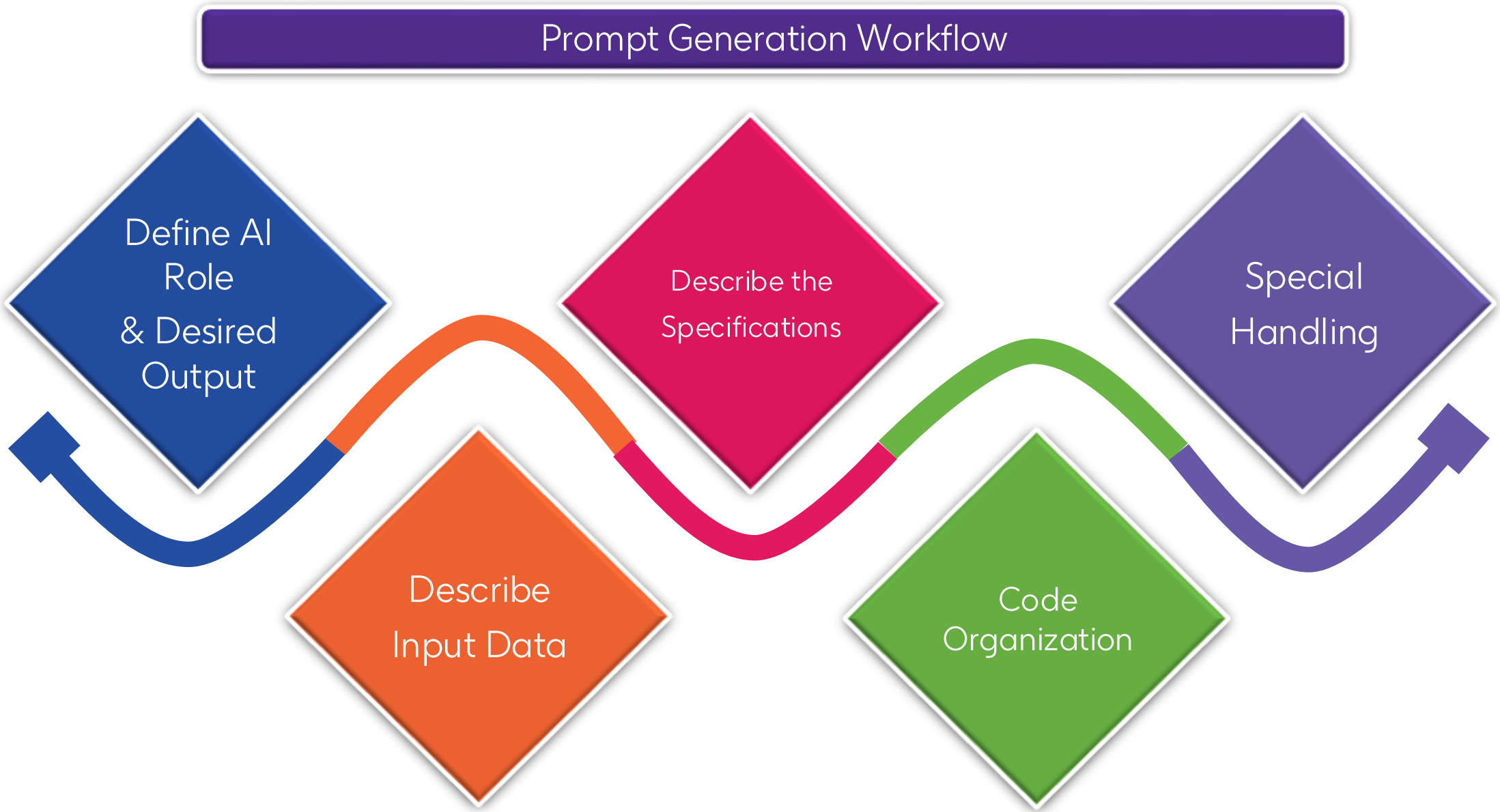
Methods

Prompt Generation

- Standardized prompt templates ensure consistency
- Specifications remain language-agnostic
- Metadata tables guide derivations
- Coding standards and package usage are enforced



Methods


Prompt Generation Workflow
Define AI Role & Desired Output
Describe the Specifications
Special Handling
Describe Input Data
Code Organization

Prompt Generation – ADSL Example

Define AI Role & Output

Start of the Prompt

Telling the AI what we plan to do, our input and our output

Only bolded text is custom to ADSL

In R, I would like you to generate code in the R language to create a dataframe called **ADSL** using the specifications that are provided in 4 .txt tab delimited file format along with additional instructions I am providing. The 4 .txt files are variables.txt, codelists.txt, comments.txt and methods.txt. The column labels are in the first row of each file. The variables.txt should be filtered to only rows where the Dataset equals **ADSL**.

```r
# --- 2. Read specification files ---
variables_spec <- read.delim("variables.txt", sep = "\t", stringsAsFactors = FALSE)
codelists_spec <- read.delim("codelists.txt", sep = "\t", stringsAsFactors = FALSE)
comments_spec <- read.delim("comments.txt", sep = "\t", stringsAsFactors = FALSE)
methods_spec <- read.delim("methods.txt", sep = "\t", stringsAsFactors = FALSE)

# --- 3. Filter variables_spec for ADSL ---
adsl_variables_spec <- variables_spec %>%
  filter(Dataset == "ADSL")
```


GSK

Prompt Generation – ADSL Example

Describe the Input Data

Explain the details including location and packages to utilize when reading data

You are given SDTM datasets **DM, SUPPDM, DS, SUPPDS, MH, SV, SC, EX, VS** which exist and can be found in the directory referred to as “sdtm”. These files are stored as SAS transport files and have the extension.xpt. For example, you read in SDTM datasets like this:dm <- haven::read_xpt(paste(sdtm, "dm.xpt", "/")). Both **DM and DS** have a SUPP domain that can be joined to the parent domain like dm <- metatools::combine_supp(dm, suppdm).You should also run the function admiral::convert_blanks_to_na() immediately after reading in a .xpt file. For example, dm <- haven::read_xpt(paste(sdtm, "dm.xpt", "/")) %>% admiral::convert_blanks_to_na().

```
# Helper function to read xpt files and convert blanks to NA
read_xpt_and_convert <- function(file_path) {
  haven::read_xpt(file_path) %>%
    admiral::convert_blanks_to_na()
}

# --- 1. Load SDTM Datasets ---
message("Loading SDTM datasets...")

dm <- read_xpt_and_convert(file.path(sdtm, "dm.xpt"))
suppdm <- read_xpt_and_convert(file.path(sdtm, "suppdm.xpt"))
ds <- read_xpt_and_convert(file.path(sdtm, "ds.xpt"))
suppds <- read_xpt_and_convert(file.path(sdtm, "suppds.xpt"))
mh <- read_xpt_and_convert(file.path(sdtm, "mh.xpt"))
sv <- read_xpt_and_convert(file.path(sdtm, "sv.xpt"))
sc <- read_xpt_and_convert(file.path(sdtm, "sc.xpt"))
ex <- read_xpt_and_convert(file.path(sdtm, "ex.xpt"))
vs <- read_xpt_and_convert(file.path(sdtm, "vs.xpt"))
qs <- read_xpt_and_convert(file.path(sdtm, "qs.xpt")) # Assuming QS domain exists for MMSETOT and EFFFL

# Combine SUPP domains with parent domains
dm <- metatools::combine_supp(dm, suppdm)
ds <- metatools::combine_supp(ds, suppds)
```

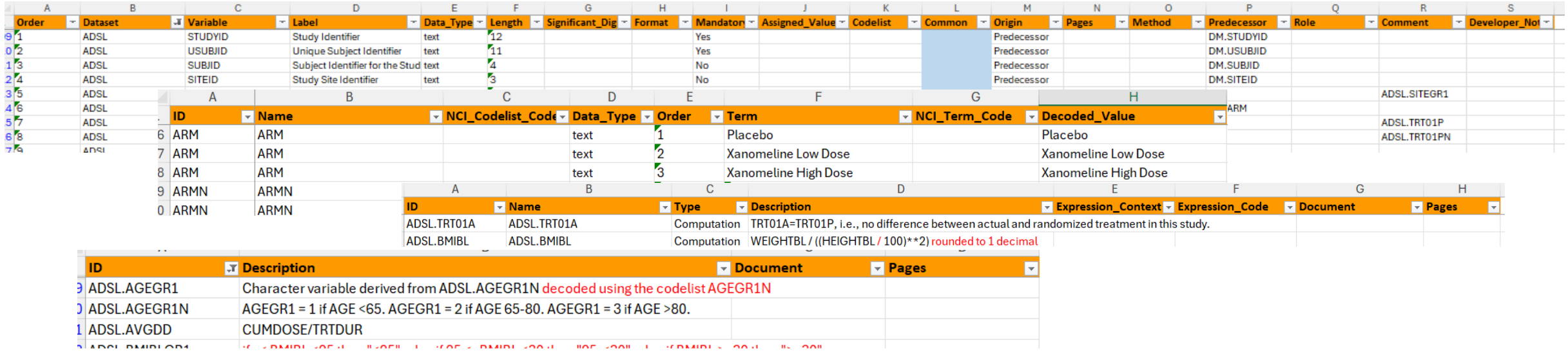
Prompt Generation – ADSL Example

Describe the Specifications

Detail what specification tables are provided and how each table is used.

Provide information on how the metadata tables relate to each other if not described in the specification

Here is a brief description of each of the 4 .txt files:variables.txt: You should only use rows in this sheet where the column Dataset equals ADSL. This file contains all variable definitions for the final ADSL dataset to be created. For example all rows in the variables table for the ADSL dataset should be included in the final ADSL dataset. The label, data type, length and origin of each variable is also provided. Ensure the final variable is the same data type defined in the specification.codelists.txt: This contains code (Term) and decode (Decoded Value) pairs identified with an ID. The codelists may be referenced in the variable definition or derivation.comments.txt: This contains a derivation to compute a variable in the Description column. Each comment is identified with an ID.methods.txt: Similar to the comments, each row contains a computation in the Description identified with an ID.



Order	Dataset	Variable	Label	Data_Type	Length	Significant_Dig	Format	Mandator	Assigned_Value	Codelist	Common	Origin	Pages	Method	Predecessor	Role	Comment	Developer_Not
1	ADSL	STUDYID	Study Identifier	text	12			Yes				Predecessor			DM.STUDYID			
2	ADSL	USUBJID	Unique Subject Identifier	text	11			Yes				Predecessor			DM.USUBJID			
3	ADSL	SUBJID	Subject Identifier for the Stud	text	4			No				Predecessor			DM.SUBJID			
4	ADSL	SITEID	Study Site Identifier	text	3			No				Predecessor			DM.SITEID			
5	ADSL																ADSL.SITEGR1	
6	ADSL														ARM			
7	ADSL																ADSL.TRT01P	
8	ADSL																ADSL.TRT01PN	

ID	Name	NCI_Codelist_Code	Data_Type	Order	Term	NCI_Term_Code	Decoded_Value
ARM	ARM		text	1	Placebo		Placebo
ARM	ARM		text	2	Xanomeline Low Dose		Xanomeline Low Dose
ARM	ARM		text	3	Xanomeline High Dose		Xanomeline High Dose
ARMN	ARMN						
ARMN	ARMN						

ID	Name	Type	Description	Expression_Context	Expression_Code	Document	Pages
ADSL.TRT01A	ADSL.TRT01A	Computation	TRT01A=TRT01P, i.e., no difference between actual and randomized treatment in this study.				
ADSL.BMIBL	ADSL.BMIBL	Computation	WEIGHTBL / ((HEIGHTBL / 100)**2) rounded to 1 decimal				

ID	Description	Document	Pages
ADSL.AGEGR1	Character variable derived from ADSL.AGEGR1N decoded using the codelist AGEGR1N		
ADSL.AGEGR1N	AGEGR1 = 1 if AGE <65. AGEGR1 = 2 if AGE 65-80. AGEGR1 = 3 if AGE >80.		
ADSL.AVGDD	CUMDOSE/TRTDUR		

Specification examples pulled from the ADaM™ Pilot Project: https://github.com/cdisc-org/sdtm-ADaM™-pilot-project

Prompt Generation – ADSL Example

Describe the Specification - Handling Processor

Tell the AI how the input files relate to each other

This should be standard for all ADaM™s if specs are written in a standard format consistently

For variables where Origin = Predecessor, there is a column called "Predecessor" which will tell you from which SDTM or ADaM™ dataset you can just pull the variable from. For example, if the Variable we are trying to get is STUDYID, the origin is Predecessor, and the Predecessor column states "DM.STUDYID", therefore STUDYID is equal to the STUDYID variable stored in the DM dataset.

```
# --- 5. Initialize ADSL dataframe with core identifier variables and direct predecessors ---
# Select mandatory and core predecessor variables directly from DM
adsl <- dm %>%
  select(STUDYID, USUBJID, SUBJID, SITEID, ARM, AGE, AGEU, RACE, SEX, ETHNIC, RFSTDTC, RFENDTC, DTHFL) %>%
  # Ensure one record per USUBJID
  distinct(STUDYID, USUBJID, .keep_all = TRUE)
```

Prompt Generation – ADSL Example

Describe the Specification - Handling Assigned

Tell the AI how the input files relate to each other

This should be standard for all ADaM™s if specs are written in a standard format consistently

Similar for origin = Derived

For variables where Origin = Assigned, you would look at the "comments" table to find the description of how to set the variable. For example, if the Variable is AAGEU, and you look at the origin and it says "Assigned", then you would look at the comments column to find the name of the comment which is “COM.AAGEU” then you head over to comments table and there will be a variable called COM.AAGEU in the ID column with a description such as "Set to Years" meaning that variable will just be the unit of Age which will be a column of "Years".

```r
# Demographic and Baseline Variables
# AGEGR1 (Assigned: Character from AGEGR1N codelist)
# AGEGR1N (Assigned: 1 if AGE <65, 2 if AGE 65-80, 3 if AGE >80)
adsl <- adsl %>%
  mutate(
    AGEGR1N = case_when(
      AGE < 65 ~ 1,
      AGE >= 65 & AGE <= 80 ~ 2,
      AGE > 80 ~ 3,
      TRUE ~ NA_integer_
    )
  )

agegr1_map <- codelists_spec %>%
  filter(Name == "AGEGR1N") %>%
  select(`NCI.Term.Code`, `Decoded.Value`) %>%
  rename(AGEGR1N = `NCI.Term.Code`, AGEGR1 = `Decoded.Value`)

adsl <- adsl %>%
  left_join(agegr1_map, by = "AGEGR1N")
```

Prompt Generation – ADSL Example

Code Organization

Describe how code provided by AI should be structured and what packages to use.

The code generated should be structured as follows. First, please source the script rxsetup.R. This will define references for the data. Next, the following packages can be used and should be included in library statements: tidyverse, admiral, metacore, metatools, haven and xportr. After the library statements, please read in all data using haven::read_xpt.

Tell the AI any additional information that is helpful to ensure quality code

Force derivations to use particular functions

The final dataset **ADSL** will have **one record per USUBJID**. The final dataframe can be written out to the ADaM™ directory as a .xpt file using the xportr package. The variable labels, lengths can be found in the variables.txt data. The final dataset should contain all variables in the variables.txt where Dataset equals **ADSL**.

The final dataset **ADSL** will have **one record per USUBJID**. The final dataframe can be written out to the ADaM™ directory as a .xpt file using the xportr package. The variable labels, lengths can be found in the variables.txt data. The final dataset should contain all variables in the variables.txt where Dataset equals **ADSL**.

Prompt Generation – ADSL Example

Special Handling

Describe any additional joins/derivations needing special handling.

Variables ending in "DT" are numeric date variables. Variables ending in "DTM" are numeric datetime variables. For derivations that require a character date to be converted to a numeric date the function admiral::convert_dtc_to_dt should be used. For derivations that require a character datetime to be converted to a numeric datetime the function admiral::convert_dtc_to_dtm should be used.

```r
# RFENDT (Assigned: RFENDTC converted to numeric date)
adsl <- adsl %>%
  mutate(RFENDT = admiral::convert_dtc_to_dt(RFENDTC))
```

Original Spec:

ID	Description	Document	Pages
ADSL.RFENDT	RFENDTC converted to numeric date		

Methods

The Generation Engine

- Combines controlled prompts, templates, and LLMs
- Produces conservative, readable R scaffolds
- Highlights assumptions and unresolved ambiguity
- Designed for traceability and auditability

Methods

Human-in-the-Loop Review

- AI output is a starting point, not final code
- Programmers run, inspect, and refine iteratively
- Domain expertise resolves ambiguity
- Regulatory compliance is preserved

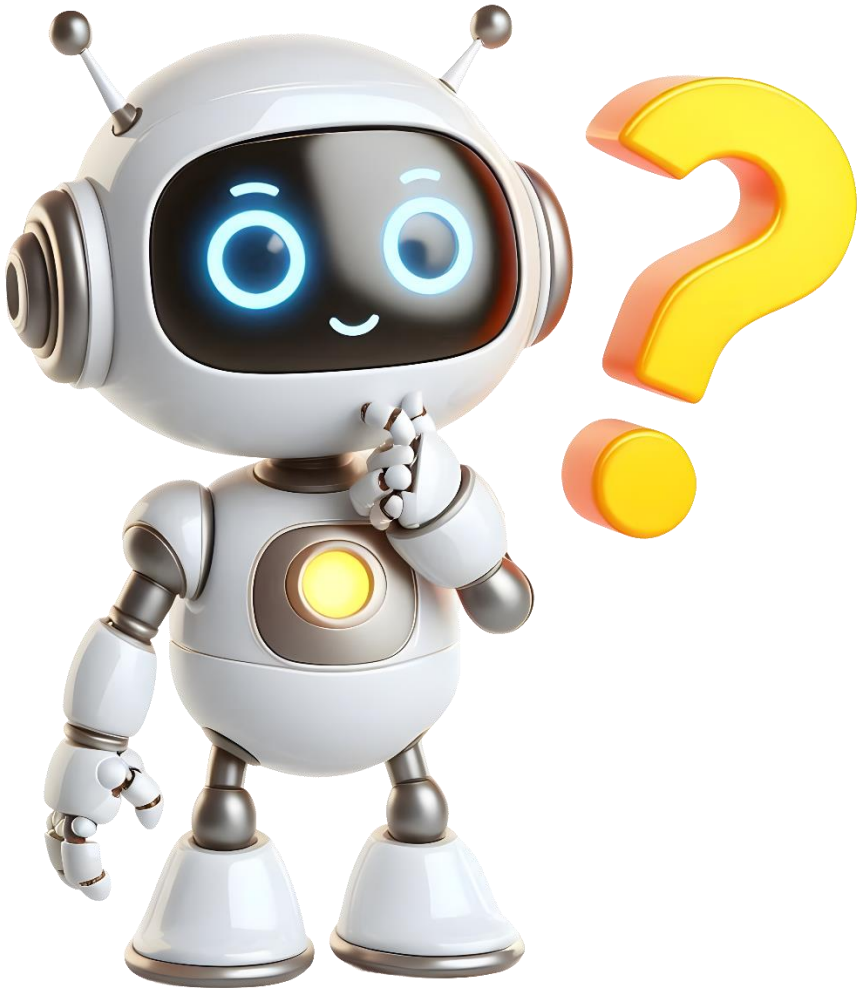
LLM failure modes

- Hallucinations
- Over Generalization
- Incorrect Assumptions

In Closing

- AI can materially accelerate ADaM™ development
- High-quality specifications are critical
- Human oversight is non-negotiable
- This model balances automation with accountability

Questions

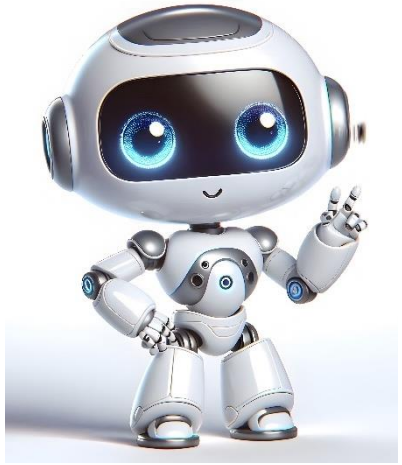


GSK

Contact Information

Alice Ehmann

Associate Director
Infectious Disease - Clinical Programming (US)
RD Projects Clinical Platforms & Sciences
Alice.x.ehmann@gsk.com
gsk.com



Kjersten Offenbecker

Programming Leader
Infectious Disease - Clinical Programming (US)
RD Projects Clinical Platforms & Sciences
Kjersten.x.offenbecker@gsk.com
gsk.com