Paper ML23

Harnessing LLM-Augmented Automation for ADaM Dataset Generation with Admiral

Ashwin Kumar Nyalakonda, K3-Innovations Inc., Hyderabad, India
Anil Mogha, K3-Innovations Inc., New Jersey, USA
Gourav Garg, K3-Innovations Inc., Hyderabad, India

ABSTRACT

Generating Analysis Data Model (ADaM) datasets remains one of the most resource-intensive and error-prone steps in clinical trial programming. Although CDISC standards and open-source frameworks such as admiral provide structure and best practices, implementation still requires extensive manual coding and deep package expertise. General-purpose Large Language Models (LLMs) show promise for accelerating code development, but without domain context they often produce non-compliant syntax, misuse pharmaverse functions, and generate outputs that are difficult to reproduce or validate.

This paper presents a metadata-driven, LLM-augmented pipeline for ADaM dataset generation implemented within the SPARC platform. By embedding ADaM specifications, controlled terminology, and explicit library usage guidelines for packages such as admiral, dplyr, and haven, the pipeline enables in-context learning that guides LLMs toward regulator-ready R code. Prompts are organized by variable categories—common variables, copied source variables, date variables, and parameter-driven variables—before being synthesized into coherent, auditable programs.

Pilot evaluations using GPT-4o and a fine-tuned Qwen-2.5-7B model successfully generated ADSL, ADAE, and ADLB datasets with reduced programming effort and improved consistency across teams. Compared to conventional manual workflows, the hybrid approach accelerates dataset development, lowers error rates, and ensures alignment with pharmaverse best practices. This paper provides a practical walkthrough of the prompt pipeline, highlights successes and limitations, and offers guidance for integrating LLM-based code generation into existing ADaM workflows while maintaining compliance, efficiency, and reproducibility.

INTRODUCTION

ADaM dataset creation is a foundational step in clinical trial analysis and regulatory submission. Despite well-established standards and the increasing adoption of open-source frameworks such as admiral, the process remains labor-intensive, requiring experienced programmers to translate specifications into correct, consistent, and validated code.

Even in organizations with strong standards, ADaM programming typically involves:

- Manual interpretation of specifications
- Repetitive boilerplate coding
- Study-specific adaptations layered on top of standard logic
- Extensive review and rework during SAP amendments

Recent advances in LLMs have renewed interest in automating portions of this workflow. However, naïvely applying general-purpose LLMs to ADaM programming introduces new risks, particularly in regulated environments where reproducibility, traceability, and standard compliance are mandatory.

This paper explores how LLM-augmented automation, when grounded in metadata and pharmaverse best practices, can modernize ADaM dataset generation without compromising regulatory expectations.

EVOLUTION OF AUTOMATION IN CLINICAL TRIAL WORKFLOWS

Clinical trial automation has progressed through phases: script-centric reuse, metadata-driven standardization, and AI-assisted augmentation. Each phase improves speed, but introduces trade-offs in transparency, maintainability, and compliance. SPARC AI is designed to combine the strengths of deterministic automation (control, traceability,

reproducibility) with the strengths of LLMs (semantic understanding, adaptation, code scaffolding), while reducing the risks of either approach used in isolation.

CURRENT STATE OF ADAM DATASET GENERATION WITH ADMIRAL

Manual Programming Limitations

Manual ADaM programming is time-consuming and sensitive to individual expertise. Common challenges include:

- Inconsistent application of derivation patterns
- Copy-paste errors
- Difficulty scaling across multiple studies
- Increased rework during late-stage changes

Metadata-Only Automation Limitations

Metadata-driven automation can standardize dataset structure and simple derivations but struggles with:

- Complex, study-specific logic
- Conditional derivations
- Parameter-driven datasets
- Nuanced interpretation of specifications

Metadata alone lacks the semantic reasoning needed to interpret intent embedded in specifications and SAPs.

LIMITATIONS OF PURE LLM-BASED CODE GENERATION

While LLMs can generate syntactically valid R code, several limitations emerge when applied directly to ADaM programming:

- **Lack of CDISC context**: Misuse of variable roles and derivation order
- **Incorrect admiral usage**: Calling non-existent or inappropriate functions
- **Non-deterministic outputs**: Inconsistent code across runs
- **Poor auditability**: Difficulty tracing code back to specifications

These limitations make pure LLM approaches unsuitable as primary generators of submission-quality ADaM datasets.

A HYBRID METADATA-DRIVEN, LLM-AUGMENTED APPROACH

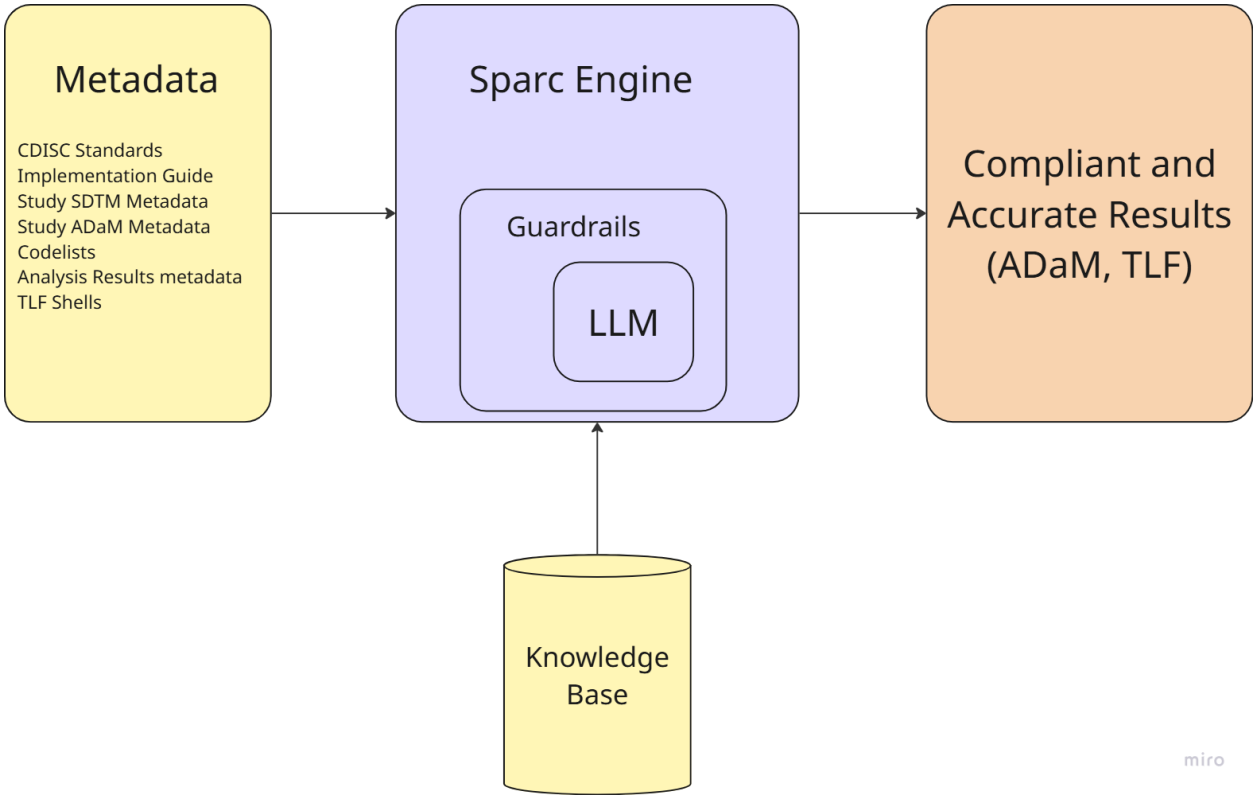
The approach presented in this paper combines the strengths of metadata-driven automation with constrained LLM augmentation.

Key Principles

1. **Metadata as the system of record**
ADaM specifications, controlled terminology, and variable classifications define what must be built.
2. **LLMs as guided assistants**
LLMs generate code only within well-defined boundaries, informed by metadata and library usage rules.
3. **Deterministic assembly and validation**
Generated code is assembled, versioned, and validated through deterministic processes.

SPARC AI ARCHITECTURE OVERVIEW

SPARC is designed as a metadata-native platform for regulatory workflows:



- **Metadata-native core**: operates on CDISC metadata and standardized representations (e.g., SDTM/ADaM structures, Define.xml concepts)
- **Deterministic control layer**: ensures reproducibility, versioning, and auditability
- **LLM augmentation layer**: supports enrichment, scaffolding, semantic validation, and study-specific flexibility, constrained by metadata context and validation gates

A CONCRETE EXAMPLE OF HYBRID CODE GENERATION FOR ADaM DATASETS (ADVS)

- A practical way to operationalize hybrid automation is to generate code in **bounded, labeled blocks**, where deterministic logic is produced from metadata and LLM-generated logic is restricted to clearly defined regions that benefit from semantic interpretation. The following example illustrates hybrid code generation for the **ADVS (Vital Signs Analysis Dataset)** using **pharmaverse/admiral** and supporting R packages.
- In this approach, the overall dataset structure, variable grouping, and execution order are determined by metadata. LLM assistance is applied only within pre-defined blocks corresponding to logically related variable groups, ensuring consistency, traceability, and reproducibility.
- In this pattern:
- **Metadata-generated structure** defines variable groupings, block boundaries, library usage, input/output conventions, and execution order.
- **LLM-assisted blocks** generate transformation logic within each variable group, guided by ADaM specifications, controlled terminology, and pharmaverse usage rules.
- The **LLM is constrained** by the pre-defined block structure, reducing ambiguity and limiting the surface area for errors or non-compliant logic.

Enriched Metadata as the Foundation for Hybrid ADaM Generation

A key enabler of the hybrid code generation approach is the use of **enriched metadata** that goes beyond traditional ADaM specifications. In addition to defining *what* variables must be created, the metadata also captures *how* code should be structured, sequenced, and constrained during generation. This enriched metadata provides the contextual grounding required for reliable LLM augmentation while preserving deterministic control.

In this implementation, enriched metadata includes the following categories.

Library and Execution Metadata

The metadata explicitly specifies the R libraries required for ADaM dataset generation, such as *admiral*, *dplyr*, and *haven*. This information is used to deterministically generate library loading blocks and ensures consistent package usage across datasets and studies. By fixing library dependencies through metadata, the LLM is prevented from introducing unsupported or non-standard packages.

Dataset-Type Metadata

Each ADaM dataset is classified by dataset type (e.g., **ADSL**, **BDS**, **OCCDS**). Dataset-type metadata drives the overall structure of the generated program, including expected variable roles, typical derivation patterns, and ordering conventions. For example, BDS datasets require parameter-driven logic and change-from-baseline variables, whereas ADSL datasets focus on subject-level attributes and population flags. This classification allows the hybrid pipeline to apply dataset-specific scaffolding before invoking LLM-assisted logic.

Source Dataset Metadata

Metadata explicitly defines the SDTM source datasets required for each ADaM dataset (e.g., **VS**, **DM**, **EX**). This information determines which input datasets are loaded and which variables are available for derivation. By constraining the LLM to known source datasets, the risk of referencing unavailable or inappropriate inputs is reduced.

Merge and Join Metadata

For datasets requiring subject-level enrichment, metadata specifies merge relationships, such as joining **ADSL** to other ADaM datasets using **STUDYID** and **USUBJID**. These merge rules are applied deterministically using standard functions (e.g., *derive_vars_merged* in *admiral*), ensuring consistent join logic across datasets and preventing the LLM from inventing ad hoc merge conditions.

Variable Grouping Metadata

Optional metadata may define **logical groupings of variables** to be generated together, such as:

- Variables derived directly from SDTM sources
- Variables merged from ADSL
- Derived analysis flags or numeric mappings

These groupings are reflected in the bounded code blocks shown in the example. Group-level metadata enables the pipeline to generate code in manageable, reviewable segments and allows LLM assistance to be applied independently within each group.

Output and Export Metadata

Metadata also specifies output formats and conventions, such as exporting datasets to **XPT** format with defined versioning. This ensures that output generation is deterministic and submission-aligned, independent of any AI-assisted logic.

ADaM Specification Metadata

Finally, traditional ADaM specification elements—such as **predecessor variables**, **computation methods**, **labels**, and **data types**—are incorporated into the enriched metadata. These elements guide the LLM in selecting appropriate derivation patterns while ensuring that generated variables remain consistent with the formal specification.

Role of Enriched Metadata in Hybrid Generation

Together, these enriched metadata elements establish a **contract** between deterministic automation and LLM-assisted logic. Deterministic processes use metadata to define structure, sequencing, and constraints, while LLMs operate only within these boundaries to handle semantic interpretation and study-specific variability. This separation allows the hybrid approach to achieve flexibility without sacrificing traceability, reproducibility, or regulatory alignment.

Evaluation Setup

Models evaluated

- GPT-4o
- Fine-tuned Qwen-2.5-7B model

Datasets generated

- ADSL
- ADAE
- ADLB

Hybrid-generated R code example for ADVS

```
# Generated on 2026-02-06 11:22:53 UTC
# ============================================================================
# Group: VS Origin Variables (Part 1 of 3)
# Variables: ABLFL, ADT, ADY, ATPT, AVAL, AVISIT, BASE, BASETYPE, PARAM, PARAMCD,
VISIT, VISITNUM, VSSEQ
# ============================================================================
# Load required libraries and initialize data
library(admiral)
library(haven)
library(dplyr)
vs <- read_sas("SDTMDatasets/Outputs/vs.sas7bdat")
adsl <- read_sas("AdamDatasets/Outputs/adsl.sas7bdat")
tryCatch(
  advs <- vs %>%
    mutate(
      PARAMCD = VSTESTCD,
      PARAM = VSTEST,
      ADT = as.Date(VSDTC),
      ADY = VSDY,
      ATPT = VSTPT,
      AVAL = VSSTRESN,
      BASE = ifelse(VSBLFL == "Y", VSSTRESN, NA),
      BASETYPE = VSTPT,
      VISITNUM = VISITNUM,
      VISIT = VISIT,
      VSSEQ = VSSEQ,
      ABLFL = VSBLFL
    ),
  error = function(e) {
    message("Error in processing VS Origin Variables: ", e)
  }
)

# ============================================================================
# Group: ADSL Origin Variables (Part 2 of 3)
# Variables: AGE, AGEGR1, AGEGR1N, RACE, RACEN, SAFFL, SEX, SITEID, STUDYID, TRTA,
TRTAN, TRTEDT, TRTP, TRTPN, TRTSDT, USUBJID
# ============================================================================
tryCatch(
  advs <- advs %>%
    derive_vars_merged(
      dataset_add = adsl,
      by_vars = exprs(STUDYID, USUBJID),
      new_vars = exprs(
```

```
        AGE = AGE,
        AGEGR1 = AGEGR1,
        AGEGR1N = AGEGR1N,
        RACE = RACE,
        RACEN = RACEN,
        SAFFL = SAFFL,
        SEX = SEX,
        SITEID = SITEID,
        TRTA = TRT01A,
        TRTAN = TRT01AN,
        TRTEDT = TRTEDT,
        TRTP = TRT01P,
        TRTPN = TRT01PN,
        TRTSDT = TRTSDT
      )
    ),
  error = function(e) {
    message("Error in processing ADSL Origin Variables: ", e)
  }
)

# ============================================================================
# Group: Other Variables (Part 3 of 3)
# Variables: ANL01FL, ATPTN, AVISITN, CHG, PARAMN, PCHG
# ============================================================================
tryCatch(
  advs <- advs %>%
    mutate(
      PARAMN = case_when(
        PARAMCD == "SYSBP" ~ 1,
        PARAMCD == "DIABP" ~ 2,
        PARAMCD == "PULSE" ~ 3,
        TRUE ~ NA_real_
      ),
      ATPTN = case_when(
        ATPT == "PRE-DOSE" ~ 1,
        ATPT == "POST-DOSE" ~ 2,
        TRUE ~ NA_real_
      ),
      AVISITN = case_when(
        AVISIT == "SCREENING" ~ 0,
        AVISIT == "BASELINE" ~ 1,
        AVISIT == "WEEK 1" ~ 2,
        AVISIT == "WEEK 2" ~ 3,
        TRUE ~ NA_real_
      ),
      CHG = AVAL - BASE,
      PCHG = ifelse(!is.na(BASE) & BASE != 0, 100 * (CHG / BASE), NA_real_),
      ANL01FL = ifelse(
        !grepl("SCREENING|UNSCHEDULED|AMBUL ECG REMOVAL", AVISIT),
        "Y",
        NA_character_
      )
    ),
  error = function(e) {
    message("Error in processing Other Variables: ", e)
  }
)
# Export the dataset
write_xpt(
  advs,
  "AdamDatasets/Outputs/advs.xpt",
  version = 5
)
```

```
print("Dataset advs has been saved to: AdamDatasets/Outputs/advs.xpt")
```

Dependencies

- tidyverse – v2.0.0 (CRAN)
- haven – v2.5.5 (CRAN)
- admiral – v1.4.1 (CRAN)
- GPT-4o – March 26, 2025

Why this reduces LLM error surface area

This example demonstrates a core principle of the hybrid approach: **deterministic automation establishes the structural contract, while LLM assistance is confined to clearly bounded semantic regions.**

In this example, deterministic metadata-driven automation:

- Defines variable groupings and execution order (VS-origin, ADSL-origin, derived variables)
- Fixes library usage (*admiral*, *dplyr*, *haven*)
- Fixes input/output locations and dataset naming conventions
- Enforces consistent block-level structure across ADaM datasets

LLM-assisted logic is restricted to:

- Mapping SDTM variables to ADaM roles within a predefined group
- Selecting derivation patterns aligned with the ADaM specification
- Controlled creation of numeric mappings, flags, and change variables

Because the structure, variable groups, and data flow are fixed, the LLM operates within a constrained context. This significantly reduces the likelihood of hallucinated variables, incorrect joins, or non-compliant derivations. The explicit separation of metadata-driven structure and AI-assisted logic also improves auditability, as reviewers can clearly distinguish deterministic logic from AI-supported transformations.

PILOT STUDY OUTCOMES

In pilot studies, SPARC demonstrated:

- 85–95% reduction in ADaM specification and TLF generation timelines
- Improved consistency and reduced manual rework during SAP amendments
- Generation of audit-ready outputs with end-to-end traceability

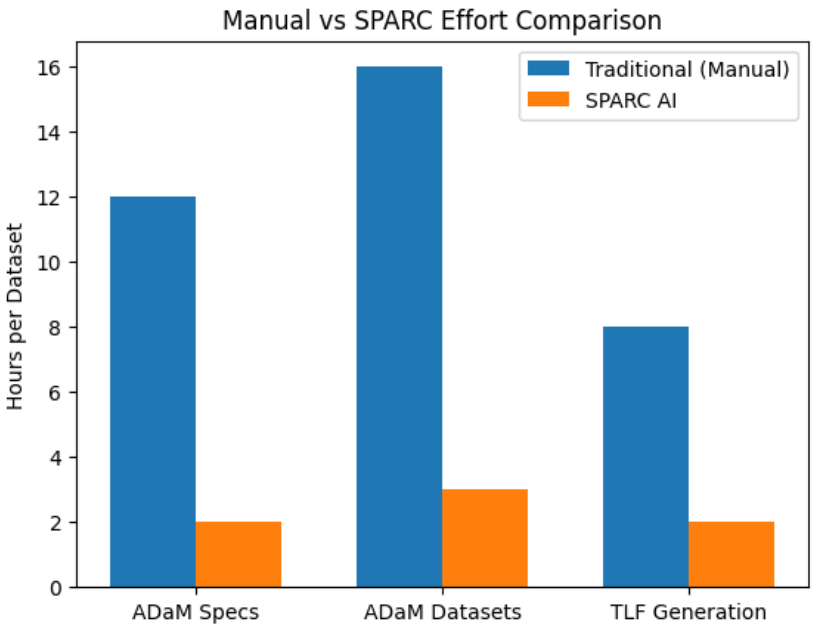


Figure 1. Reduction in Programming Effort per Dataset Using Hybrid AI Native Metadata-Driven Automation

These outcomes reflect both time savings and reduced operational risk in late-stage trial activities.

BENEFITS AND LIMITATIONS

Benefits

- Accelerated ADaM development
- Reduced dependency on individual expertise
- Improved standardization
- Better alignment with open-source best practices

Limitations

- LLM outputs still require review
- Complex edge cases may need manual intervention
- Prompt design and metadata quality are critical success factors

---

CONCLUSION

LLMs alone are insufficient for generating compliant ADaM datasets, and metadata-only automation lacks the flexibility needed for real-world studies. A hybrid, metadata-driven and LLM-augmented approach offers a practical path forward.

By embedding ADaM specifications and pharmaverse usage guidelines directly into a structured prompt pipeline, LLMs can be guided toward consistent, regulator-ready outputs. Pilot evaluations demonstrate meaningful efficiency gains while preserving reproducibility and compliance.

This work illustrates how LLM-augmented automation can be responsibly integrated into ADaM workflows, modernizing dataset creation while respecting the rigor required for regulatory submission.

REFERENCES

1. K3-Innovations. (n.d.). *SPARC AI platform*. https://sparcgen.com/
2. OpenAI. (2024). *GPT-4o model documentation*. https://platform.openai.com/docs/models/gpt-4o
3. Clinical Data Interchange Standards Consortium. (2019). *SDTM–ADaM pilot project*. https://github.com/cdisc-org/sdtm-adam-pilot-project
4. Pharmaverse. (2026). *admiral: ADaM in R asset library*. https://pharmaverse.github.io/admiral/

ACKNOWLEDGMENTS

The authors thank **Kajal Anandani**, CEO of K3-Innovations, for her support and guidance. We also thank **Daniela Popa** and the **K3-Innovations team** for their collaboration and feedback during the development and pilot implementation of SPARC AI.

CONTACT INFORMATION

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

Author Name: Anil Mogha
Company: K3-Innovations, Inc.
Email: anil.mogha@k3-innovations.com
Website: https://k3-innovations.com

Author Name: Ashwin Kumar
Company: K3-Innovations, Inc.
Email: ashwin.kumar@k3-innovations.com
Website: https://k3-innovations.com

Author Name: Gourav Garg
Company: K3-Innovations, Inc.
Email: gourav.garg@k3-innovations.com
Website: https://k3-innovations.com