Paper SM05

From Document to Derivation: Automating ADaM Variable and Endpoint Code Generation Using Study Specifications and Clinical Documents

Alisha D, Pfizer Healthcare India Pvt Ltd, Chennai, India
Sruthy Biju, Pfizer Healthcare India Pvt Ltd, Chennai, India

1. ABSTRACT

In clinical research, the process of deriving endpoints and validating ADaM datasets often demands intensive manual effort, deep domain expertise, and careful interpretation of clinical documents like protocols, SAPs, and dataset specifications. These documents are typically unstructured and highly variable across studies, making reproducibility and traceability a constant challenge. To address this gap, we introduce an interactive dashboard that automates endpoint derivation, ADaM variable code generation, and traceability mapping.

This leverages a hybrid intelligence approach that combines Natural Language Processing (NLP) and Large Language Models (LLMs) to interpret and extract endpoint definitions from clinical documents in PDF or DOCX formats. Once key elements such as the definition of endpoints such as time-to-event endpoints (e.g., PFS, OS, DoR) or baseline flag logic are identified, the tool uses a structured code templating engine to generate corresponding R code. This supports rapid derivation of endpoints while ensuring alignment with CDISC standards and the unique specifications of each study.

In parallel, it offers a metadata-driven code generator that enables users to select specific ADaM variables and instantly receive derivation code based on the uploaded ADaM specifications.

A key differentiator is its Traceability Engine, which visualizes and documents the full lineage of endpoint derivations—from SDTM source domains (e.g., AE, DS, RS) to intermediate derivation steps and final ADaM variables.

The dashboard is built on a flexible Python-based architecture that integrates document interpretation, metadata handling, and code generation workflows. It combines Natural Language Processing (NLP) tools such as spaCy and transformers and may incorporate Large Language Models (LLMs) such as GPT-based or BERT-based models—for extracting and interpreting endpoint definitions from unstructured clinical documents. By unifying document interpretation, metadata-driven automation, and traceability visualization, the tool empowers clinical programming teams to deliver high-quality, transparent, and reproducible outputs across diverse clinical studies.

Keywords *– Clinical Programming; CDISC ADaM; Automated R Code Generation; Large Language Models; Endpoint Derivation; Retrieval- Augmented Generation*

2. INTRODUCTION

The process of converting clinical dataset specifications into executable derivation code is a crucial yet predominantly manual task in clinical programming, often requiring significant interpretation of unstructured derivation text and ongoing reconciliation between documentation and its implementation. Specification documents often contain intricate logic, diverse source references, and changing rules, rendering this process lengthy and prone to inconsistency. To tackle this issue, we present an offline, automated framework that converts derivation descriptions from specification spreadsheets into validated R code, with intelligent management of direct mappings, derived variables, dataset interdependencies, and optimized mutation logic. By utilizing a locally deployed code generation model and rule-aware preprocessing, the system guarantees traceability and reproducibility of derivations in accordance with CDISC principles. Moreover, the framework offers minimal assistance for standardized endpoint derivations through regulated rule retrieval and alteration, acting as an auxiliary extension instead of being the main emphasis.

3. RELATED WORK AND BACKGROUND

Large language models (LLMs) have shown impressive abilities in producing executable code from natural language prompts by developing joint representations of human and programming languages. Research on code generation using LLMs indicates that transformer-based models can successfully tackle tasks like code synthesis, completion, and translation; however, it also points out ongoing issues regarding semantic accuracy, hallucinations, and the sensitivity to prompt wording, especially in applications where reliability is critical (Jiang et al., 2024).

The CodeGen model contributes to this field by presenting an open large language model tailored for code, featuring capabilities for multi-turn program synthesis and demonstrating enhanced logical consistency through iterative generation methods (Nijkamp et al., 2022). In conjunction with this, Tenneti et al. (2025) assess the practical dimensions of code generation using large language models, highlighting both productivity improvements and challenges in terms of correctness and interpretability. Together, these studies indicate that, although large language models are proficient in automating code generation, their application in regulated or specification-based workflows necessitates controlled generation processes, grounding in established logic, and traceable transformations.

4. METHODOLOGY

4.1. VARIABLE METADATA- DRIVEN R CODE GENERATION FRAMEWORK

In this section, a combination of an open-source large language model (LLM) and rule-based algorithms is employed to automatically generate R code from the given specifications. This process is structured into multiple phases to ensure accuracy and usability. First, the raw metadata undergo a comprehensive data cleaning pipeline, which involves tasks such as segregating content based on its source, applying text normalization techniques, and systematically identifying and flagging date-related variables. These preprocessing steps are critical for transforming heterogeneous specification inputs into standardized derivation prompts. Once cleaned and structured, these derivation prompts are then provided to the LLM, which interprets them and produces the corresponding R code implementations.

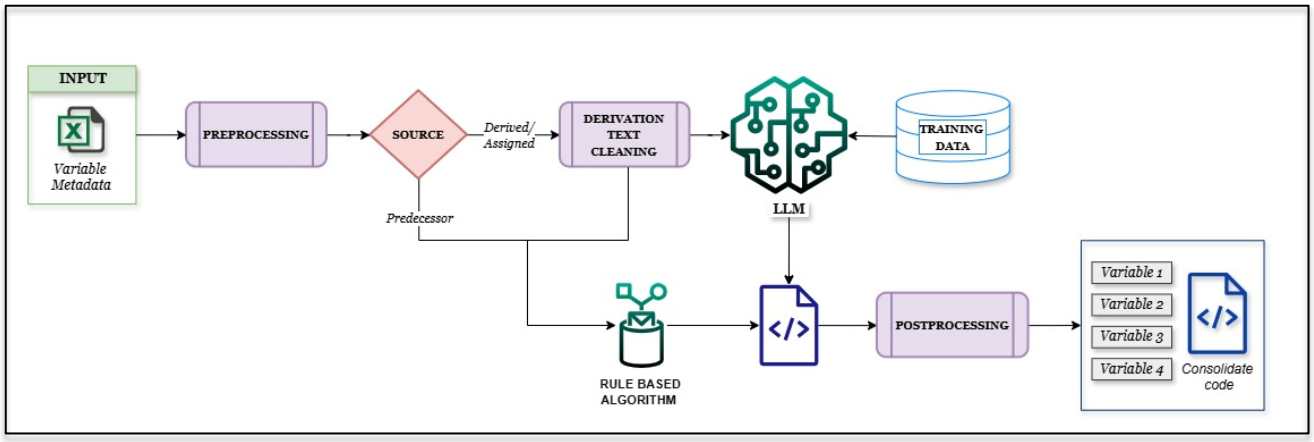

Figure 1: Flowchart of Variable Metadata to R Code

4.1.1 INPUT METADATA STRUCTURE AND PREPROCESSING

The variable metadata was developed in accordance with CDISC endpoint examples, which organize each variable into: Variable Name and Source/Derivation/Comment. For greater clarity, the Source/Derivation/Comment field can be subdivided into Source and Derivation. Using Source information, variables are categorized into two groups: Predecessor and Assigned/Derived.

Predecessor variables represent direct mappings from the source data, whereas Assigned/Derived variables require programmed derivation based on predefined logic. Within the Assigned/Derived group, certain date variables may still be direct mappings; however, these require conversion from ISO date formats to numeric representations. Such variables are flagged as date variables and are processed together with the Predecessor data using the date conversion routines.

4.1.2. R CODE GENERATION STRATEGY

Direct Mapping of Variables (Rule- Based Logic):

For variables that require direct assignment from the source dataset, a rule-based algorithm was employed to automatically generate the corresponding R code. These rules map source variables to target variables using predefined logic, ensuring consistency and reproducibility across datasets.

Derived Variables (LLM- Based Logic):

To generate R code for derived variables from specification-based programming rules, an open-source large language model (LLM) pretrained on multiple programming languages was leveraged. The model was fine-tuned specifically on CDISC ADaM derivation patterns so that it could interpret programming rules and produce valid R code for variable derivations.

Training Data Construction: The training dataset was constructed using paired prompts and targets that reflect the structure of CDISC endpoint derivations. Each prompt consisted of a concatenation of the variable name and its derivation description, while the corresponding target contained the appropriate R code implementing that derivation. The training data emphasized combining structured data manipulation techniques with core R operations to reflect real-world ADaM programming practices.

Model Architecture and Fine- Tuning: The model used for R code generation was CodeGen, an open-source family of large language models developed by Salesforce for program synthesis from natural-language or example-based specifications. The model is optimized to generate syntactically correct and semantically meaningful code across multiple programming languages. Within this family, CodeGen-350M-Multi model is a lightweight 350-million-parameter variant trained on a multilingual mixture of code and text, making it efficient while still capable of handling multiple programming languages. Although smaller than other CodeGen models, it retains strong performance in code generation and understanding tasks, and its cross-language training allows it to generalize patterns learned in one language to others, which is useful for tasks such as code translation, refactoring, and developer assistance. This model was further fine-tuned on the curated CDISC-aligned training dataset to learn the mapping between derivation rules and their corresponding R implementations.

4.1.3. POSTPROCESSING AND UNIFIED R CODE ASSEMBLY

In the postprocessing phase, the R code produced by the model is methodically combined with R code that has been directly mapped to create a unified transformation workflow. This combination takes place only after all necessary preprocessing steps have been completed to guarantee the data's internal consistency and accuracy. The generation of code is closely linked to the source datasets—regardless of whether they are mapped, assigned, or derived—thus maintaining clear traceability for each variable and the logic behind its derivation.

The development of variables related to dates, times, or datetimes follows standardized transformation methods that comply with industry norms for clinical data modeling. These methods facilitate uniform conversion from source representations to formats suitable for analysis and enable the controlled implementation of study-specific imputation rules, all while ensuring traceability, consistency, and adherence to regulatory requirements for time-based endpoints.

The result is established at two tiers: (1) R code for each variable that clearly outlines the reasoning behind the derivation of every single variable, and (2) a unified R code that combines all the variable-specific logic into a singular executable program. This two-tiered output format facilitates both clear validation and effective generation of datasets from start to finish.

4.2. ENDPOINT REPOSITORY–DRIVEN RETRIEVAL AND CODE GENERATION FRAMEWORK

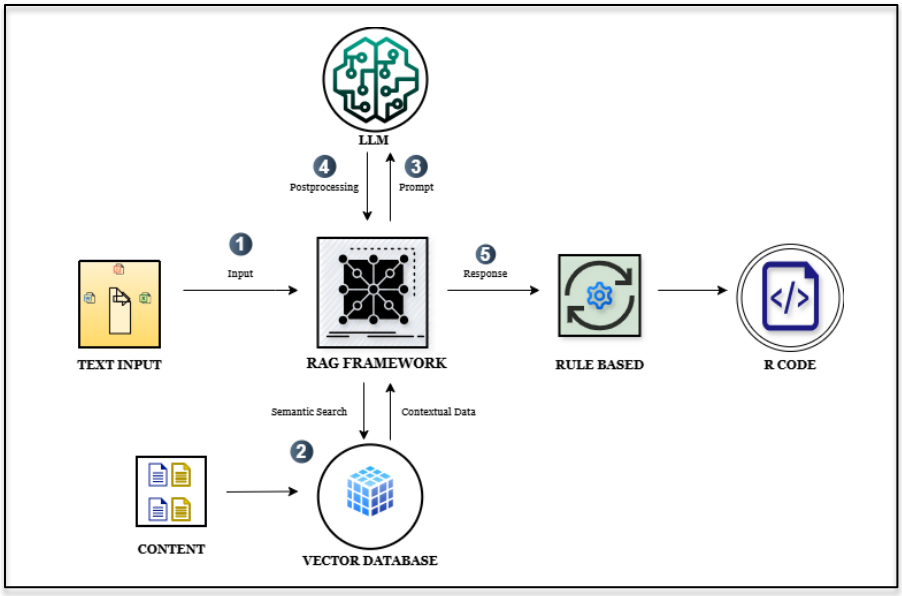

Figure 2: Flowchart of Endpoint Repository

4.2.1. ENDPOINT REPOSITORY CONSTRUCTION

A centralized repository for endpoints was created using a structured Excel file to document standardized endpoint definitions along with their associated controlled derivation rules. Each entry in the repository corresponds to a pre-approved clinical endpoint that adheres to CDISC ADaM standards and widely accepted regulatory definitions (such as OS, PFS, DOR, TTR). The repository consists of the following critical elements:

- Endpoint: Standard name for the endpoint (e.g., PFS, OS).
- Definition: Descriptive explanation of the endpoint as generally outlined in clinical protocols.
- Rules: Controlled, user-friendly logic for deriving definitions of events, criteria for censoring, and analysis timeframes.
- Derivation Template: A standardized R code framework that implements the endpoint logic using ADaM variables.

This repository acts as a verified knowledge base, guaranteeing that all subsequent derivations are based on pre-reviewed and reusable endpoint logic, which minimizes subjective interpretation during study-specific execution.

4.2.2. RETRIEVAL-AUGMENTED ENDPOINT IDENTIFICATION FROM TEXT

To facilitate protocol-driven retrieval of standardized endpoint definitions, a Retrieval-Augmented Generation (RAG) methodology has been applied. In contrast to generating free-form text, RAG limits outputs to retrieved and validated information, which is essential for regulatory and quality control scenarios.

Text Embedding and Vector Representation: Each endpoint definition was encoded using a pre-trained Sentence Transformer model to generate dense semantic embeddings. Embeddings were L2-normalized to enable cosine similarity computation. Similarly, user-provided endpoint descriptions were encoded into normalized embedding vectors using the same model.

Similarity Computation: When a user inputs a protocol-level endpoint description (for instance, “Progression or death within 90 days after treatment initiation”), the text input is embedded. A FAISS inner-product index (IndexFlatIP) was constructed over the normalized endpoint embeddings. Since vectors were L2-normalized, inner-product similarity was equivalent to cosine similarity.

$$Similarity(A, B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

The result produces a score ranging from -1 (completely opposite) to 1 (exactly the same), while a score of 0 signifies that the concepts are unrelated. An effective vector index (FAISS) is utilized to conduct a top-k nearest-neighbor search, which pulls the endpoint definitions that are most semantically similar.

4.2.3. CONTROLLED RULE EDITING AND QC-ORIENTED R CODE GENERATION

Once an endpoint is selected, the associated base rule is displayed in read-only format to maintain standardization. User modifications are strictly constrained to:

- Variable substitution (CAPS-only variables, e.g., DTHDT, TRT_STDT)
- Pre-approved rule clauses (e.g., censoring windows, event restrictions)
- Numeric parameters (e.g., time windows in days or months)

Free-text modification of core logic is not permitted, ensuring governed flexibility while preventing logic drift.

Automatic Logic Propagation: Edits to the rule text are programmatically detected by comparing variable usage before and after editing.

These changes are automatically propagated into the corresponding R code template via deterministic string substitution, guaranteeing rule–code consistency.

4.2.4. APPLICATION FOR QC AND REGULATORY VALIDATION

This framework is especially useful for QC validation processes, where separate programmers need to recreate endpoint derivations without consulting the production code. By enabling QC users to input solely the description of the protocol endpoint, the system autonomously generates standardized endpoint rules, relevant derivation logic, and clear R code implementations. This feature allows for independent confirmation of endpoint derivations, helps identify discrepancies between protocol specifications, TFL definitions, and ADaM programs, and minimizes ambiguity in the interpretation of endpoints during regulatory reviews.

5. COMPARATIVE EVALUATION WITH ALTERNATIVE CODE GENERATION MODELS

The framework was tested on a separate set of endpoint specifications aligned with CDISC, focusing on assessing syntactic validity, semantic accuracy against expert reference code, and compliance with ADaM standards. The

CodeGen-350M-Multi model exhibited effective code generation with reduced computational demands and quicker inference, thanks to its relatively smaller parameter size, producing high-quality executable code compared to Salesforce/codet5-base (which, while faster, tends to be less accurate in generation) and larger models like CodeLlama-7B (which generally produce superior quality but necessitate more computing resources), signifying an advantageous balance between speed and correctness for the intended R code derivation application.

6. PERFORMANCE EVALUATION AND METRICS

The combined rule-based and LLM-driven approach was evaluated for its effectiveness in both code generation and endpoint retrieval. For code generation, performance was assessed by measuring the proportion of model-generated R code snippets that were syntactically executable and logically consistent with reference implementations. Approximately 70% of the generated codes were fully functional and accurately reflected the intended derivation logic, while the remaining discrepancies were largely attributable to minor syntactic issues or logical errors due to limited training data, rather than systematic modelling deficiencies.

Endpoint retrieval performance was assessed using Mean Reciprocal Rank (MRR), defined as the average inverse rank of the first correct endpoint retrieved per query. Using sentence-transformer embeddings and a FAISS cosine similarity index, the framework achieved an MRR of approximately 0.80, indicating reliable top-ranked retrieval. This performance is sufficient for guided endpoint selection in a governed, ADaM-compliant workflow.

7. FUTURE WORK AND EXTENSIONS

While the current system manages specific post-processing situations like date-based derivations, further enhancements are necessary to tackle more intricate challenges such as conditional logic, dependencies between datasets, handling missing data, and dynamic censoring rules.

Future developments will focus on improving the model's capacity to understand and produce complex derivation logic essential for efficacy datasets and time-to-event endpoints, incorporating nested conditions, prioritizing multiple events, and implementing endpoint-specific censoring strategies. Expanding the framework to facilitate adaptive learning through programmer feedback could lead to ongoing enhancements in the accuracy and durability of derivations.

8. CONCLUSION

This work presents a governed, offline framework that significantly reduces the manual burden and interpretive risk associated with translating clinical specification documents into executable R code. By integrating a controlled endpoint repository, retrieval-augmented endpoint selection, and rule-constrained editing, the framework ensures that endpoint derivations remain standardized, traceable, and aligned with CDISC ADaM principles. The hybrid use of deterministic rule-based logic for direct mappings and a fine-tuned, lightweight code generation model for derived variables enables reliable automation without sacrificing auditability or regulatory confidence. Importantly, the separation of rule governance from code generation prevents logic drift while still allowing study-specific customization within clearly defined boundaries.

From a quality control and regulatory validation perspective, the framework provides a practical mechanism for independently reproducing endpoint derivations using only protocol-level descriptions and approved rules. The strong endpoint retrieval performance, high agreement with expert reference code, and consistent generation of executable, ADaM-compliant R scripts demonstrate that the approach achieves a favorable balance between accuracy, efficiency, and computational cost. Overall, this system represents a scalable and reproducible solution for modern clinical programming workflows, supporting both primary dataset production and independent QC validation while maintaining transparency, consistency, and regulatory readiness.

REFERENCES

- Jiang, J., Wang, F., et al. (2024). A Survey on Large Language Models for Code Generation. arXiv preprint.
- Nijkamp, E., Pang, B., Hayashi, H., et al. (2022). CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv preprint.
- Tenneti, L. S. D., Gayatri, M. K., Prabha, M. D., & Shravani, D. (2025). Code Generation Using LLMs. Future Engineering Journal.
- E. Dehaerne, B. Dey, S. Halder, S. De Gendt and W. Meert, "Code Generation Using Machine Learning: A Systematic Review," in IEEE Access, vol. 10, pp. 82434-82455, 2022, doi: 10.1109/ACCESS.2022.3196347.

- Nijkamp, Erik, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. "A Conversational Paradigm for Program Synthesis." arXiv preprint (2022).
- CDISC. (2026, Jan 20). CDISC standards. Clinical Data Interchange Standards Consortium. https://www.cdisc.org/
- OpenAI. (2026). ChatGPT (20 Jan version) [Large language model]. https://chat.openai.com

CONTACT INFORMATION

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

Authors Name: Alisha D, Sruthy Biju
Company: Pfizer Healthcare India Pvt.Ltd.
Address: New No.237, Anna Salai, Thousand Lights, Chennai, Tamil Nadu 600014, India
Email: alisha.d@pfizer.com, sruthy.biju@pfizer.com