PharmaSUG China 2025 - Paper 144

Automating SDTM Programming with Generative AI: A GPT-Based Framework for Clinical Data Transformation

Yanfen Zhu, Senior Manager, Statistical Programming, Pacira Biosciencals.

ABSTRACT

This project explores a novel approach to automating the development of programming logic for STDM (Standard Data Tabulation Model) using Large Language Models (LLMs) such as GPT. The methodology begins by preparing SDTM specifications for each domain. The study specifications are then translated into a structured description format that the LLM can interpret to understand the intended logic. Leveraging state-of-the-art prompting techniques and contextual modeling, Python code is generated to implement the STDM datasets. The generated code is executed in a predefined sequence, typically involving variables based on another derived variables—followed by quality control (QC) checks. If errors occur during execution, the system uses LLM-based analysis to interpret the error messages and iteratively resolve issues, enabling a feedback-driven development loop. This framework demonstrates the potential of generative artificial intelligence (AI) in accelerating and simplifying clinical data programming workflows, while maintaining accuracy and adaptability.

INTRODUCTION

In clinical trials, the transformation of raw data into standardized formats is a critical step for regulatory submission, data sharing, and analysis. The SDTM developed by the Clinical Data Interchange Standards Consortium (CDISC), is the industry standard for structuring clinical trial data, with its detailed implementation guidelines provided in the SDTM Implementation Guide (SDTMIG) (CDISC. 2023). However, converting raw clinical data into SDTM-compliant datasets is often a complex, labor-intensive process that involves detailed interpretation of specifications, manual coding, and rigorous quality control. Traditional methods of SDTM programming rely heavily on skilled programmers who translate study-specific metadata and protocol requirements into executable scripts—typically in SAS or Python. This process is not only time-consuming but also prone to inconsistencies and human error, especially across multiple studies and teams. As clinical development timelines shorten and data complexity increases, there is growing interest in leveraging AI to automate and streamline these workflows (AbbVie, 2025).

Recent advances LLMs, such as OpenAI's GPT, offer a novel opportunity to revolutionize clinical data programming. LLMs are capable of understanding and generating human-like text, making them well-suited to interpret structured metadata, study specifications, and programming logic (Brown et al., 2020). When guided with structured prompts and context-rich inputs, these models can generate reliable, reproducible code for SDTM dataset creation. This paper presents an end-to-end framework that uses LLMs to automate SDTM programming. The proposed approach begins with the preparation of study-specific metadata, translates it into a structured language model-friendly format, and uses prompt-based templates to generate executable Python code. The system includes an integrated quality control (QC) mechanism that leverages LLMs for iterative debugging and validation. Our framework demonstrates the feasibility of using generative AI to accelerate SDTM dataset development while maintaining accuracy, traceability, and compliance with regulatory standards.

OVERVIEW OF THE FRAMEWORK

This framework automates the creation of SDTM-compliant datasets using Large Language Models (LLMs) through a structured, six-step process. It begins with preparing machine-readable metadata from study specifications, followed by translating those specifications into structured prompts that LLMs can understand. Using these prompts, the model generates Python code to implement each domain (e.g., DM, EX, IC). The code is then executed in a logical sequence, producing SDTM datasets. Automated quality checks validate the structure, logic, and consistency of the outputs. If errors occur, the system uses LLM-

based feedback to analyze and fix issues, creating an efficient loop that reduces manual effort and improves reliability in clinical data programming.

STUDY SPECIFICATIONS PREPARATION

The process starts by building the foundational structure of the datasets needed for the study. The Study Data Tabulation Model (SDTM) includes a wide array of domains, but we'll initially concentrate on the most common and essential ones, like DM (Demographics), EX (Exposure), and IC (Inclusion Criteria), along with other study-specific relevant domains.

This step involves extracting and, if necessary, modifying the standard SDTM dataset specifications to derive the critical metadata. This metadata is then prepared in a machine-readable format (such as CSV). This includes Table Names: Unique identifiers for each domain (e.g., DM, EX, IC). Variable Definitions: Each table has a defined set of variables. The metadata for each variable capture:Variable Name (e.g., EXSTDTC), Data Type (e.g., character, numeric, datetime), Controlled Terminology (if applicable), Length and Format, Label/Description, Primary Keys and Sorting Order, Relationships between domains (e.g., linking DM with EX by USUBJID), Mapping Rules: Instructions on how raw source variables will populate the SDTM variables.

This critical metadata is sourced from study-specific SDTM specification documents (e.g., Case Report Forms (CRFs), define.xml, or Excel spec sheets). By standardizing this information, we create a clear blueprint spec.txt (example as following) that the LLM can interpret to generate the necessary programming logic.

```
To generate the set of variables required, we need to follow a structured sequence of steps to ensure data integrity and
consistency. Below is the detailed plan for implementing each variable:

### Plan for Variable Generation

1. **STUDYID (Study Identifier)**
   - **Source Table:** DM
   - **Source Column:** SUBJNUM
   - **Implementation:** Extract `STUDYID` directly from the DM table for each participant.

2. **DOMAIN (Domain Abbreviation)**
   - **Source Table:** Assigned
   - **Implementation:** Set DOMAIN to a constant value "DM" for demographics, suggest expanding for other domains as re
quired.

3. **USUBJID (Unique Subject Identifier)**
   - **Source Table:** DM
   - **Source Columns:** STUDYID, SUBJNUM
   - **Implementation:** Concatenate `STUDYID` and `SUBJNUM` with a separator (e.g., "-") to form `USUBJID`.

4. **SUBJID (Subject Identifier for the Study)**
   - **Source Table:** DS
   - **Source Column:** SUBJNUM
   - **Implementation:** Copy `SUBJNUM` from DS table. Ensure exclusion of rescreening subjects, as indicated.

5. **RFXSTDTC (Date/Time of First Study Treatment)**
   - **Source Table:** EX
   - **Source Column:** EXDAT
   - **Implementation:** Find the earliest `EXDAT` for each subject to define the first study treatment date.

6. **RFXENDTC (Date/Time of Last Study Treatment)**
   - **Source Table:** EX
   - **Source Column:** EXDAT
   - **Implementation:** Find the latest `EXDAT` for each subject to define the last study treatment date.

7. **RFICDTC (Date/Time of Informed Consent)**
   - **Source Table:** DS
   - **Source Column:** DSSTDAT
   - **Implementation:** Use `DSSTDAT` to determine the informed consent date for participants. Ensure proper filtering
based on the event description for informed consent.
```

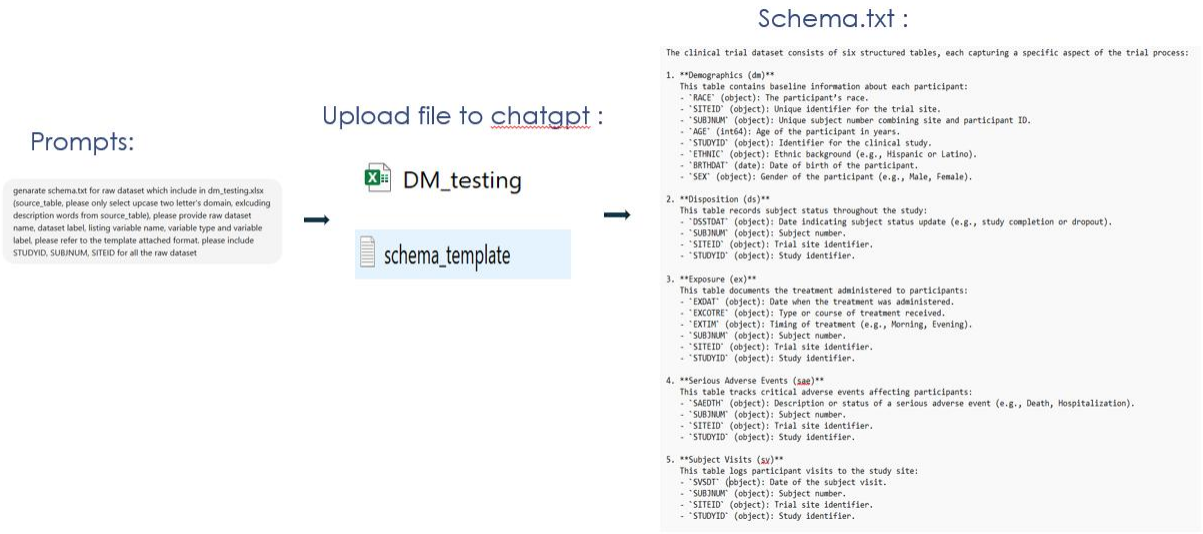
Display 1. Domain specific description (DM) from SDTM specification generated by AI

GENERATE A STRUCTURED DESCRIPTION SCHEMA FOR LLMS

Once raw data is prepared, the next step is to convert the narrative or tabular specifications into a structured, natural language or template-driven description schema that a language model can understand. The goal is to create domain-specific instructions and normalize terminology to ensure that terms used are standardized and consistent for LLM comprehension.

This involves:

- Parsing the spec document: translate one variable's spec row into a structured schema format
- Select useful variables from raw data and remove unnecessary information
- Apply standard structure: Makes logic machine-readable and generalizable



Display 2. Generate a Structured Description Schema for LLMs Steps by AI

LLM-BASED CODE GENERATION USING TEMPLATE-DRIVEN PROMPTS

After preparing the metadata and translating study specifications into a structured description, the system generates Python code by prompting a Large Language Model (LLM) using a template-based prompt framework. This prompt template is dynamically populated with content extracted from the metadata (e.g., variable names, types, domains) and the specification logic (e.g., derivations, filters, merges). The goal is to consistently provide the LLM with rich, standardized, and context-specific instructions.

The prompt template follows a fixed structure, typically including the following sections:

- Target domain name (e.g., EX, DM)
- Input source datasets (e.g., raw.DM, EX, SV…)
- Variable mapping instructions, formatted in plain language (e.g., " Variables like USUBJID, AGE, RFSTDTC, etc. must be computed using the specified rules ")
- Conditional logic and filters (e.g., " Date fields are already in string format ")
- Join conditions and keys (e.g., " Only use columns that exist in the defined schema")
- Output structure constraints, aligned with SDTM standards (e.g., Ensure variable names in the output match SDTM DM naming exactly)

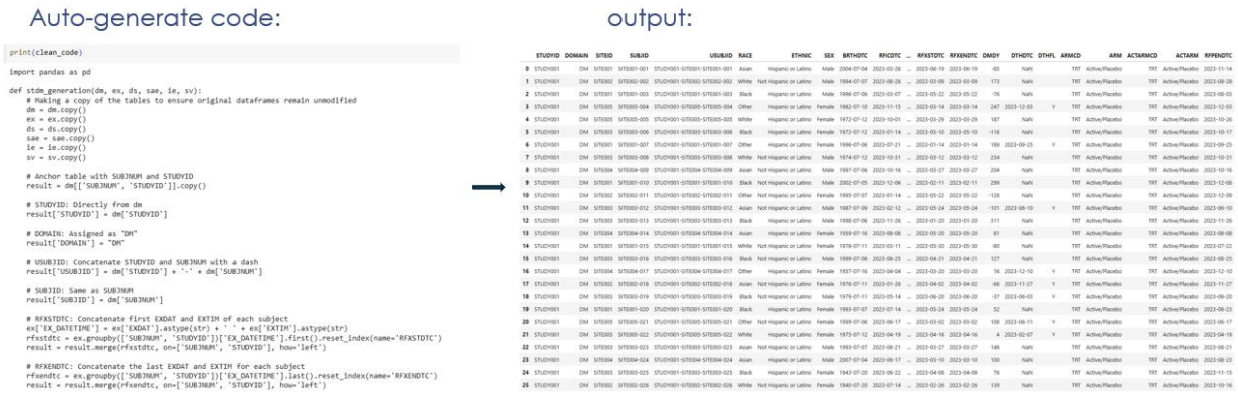
The template is as follows:

```
system_prompt = f'''
You are a clinical data processing assistant specialized in SDTM dataset generation using pandas in Python.

Your task is to generate a Python function named stdm() that constructs the DM (Demographics) domain of an SDTM-compliant dataset.

The user will provide the following:

    The business and derivation rules for the DM variables

    Follow these strict assumptions:

All input tables are pandas DataFrames and named in lowercase (e.g., dm, ds, ex, sv, sae)

All column names inside the DataFrames are in UPPERCASE

Date fields are already in string format, all other columns are also in string format except AGE in int formart

Variables like USUBJID, AGE, RFSTDTC, etc. must be computed using the specified rules.

The data table schema is {schema}

The function stdm() should:

    Take the required DataFrames as arguments

    Return a single DataFrame with the generated DM domain (dm_out)

    Handle missing data gracefully (e.g., no SAE record = no DTHDTC)

    Use ISO 8601 formatting for datetime fields (e.g., 'YYYY-MM-DD')

    Ensure variable names in the output match SDTM DM naming exactly, just provide the code do not provide comments

Only use columns that exist in the defined schema. Do not assume missing columns unless defined.

'''
```

```
user_prompt = f'''
The output should be a DataFrame (dm_out) containing the following DM variables, with logic {specs}.
'''
```

Display 3. Prompt for SDTM generation

The structured prompt ensures that the LLM receives all necessary context in a predictable, interpretable form, improving both the accuracy and reproducibility of the generated code. The resulting Python code, typically leveraging pandas, can transform raw input into SDTM-compliant output according to the specified logic. This template-driven prompting approach bridges the gap between clinical specifications and executable code, enabling automation while maintaining clarity and control.



Display 4. LLM-generated code is executed to produce the SDTM dataset DM

AUTOMATED QUALITY CONTROL (QC) AND ERROR RESOLUTION

After the LLM-generated code is executed to produce the SDTM datasets. The next critical step is automated Quality Control (QC). This step ensures that the output data conforms to expected structures, derivation rules, and SDTM standards. QC is performed using a combination of predefined validation checks, metadata-driven assertions, and runtime monitoring.

Key components of the QC process include:

- Structural Validation: Verifying that all required variables are present, properly named, and follow expected data types and formats (e.g., ISO 8601 for dates).
- Row-Level Validation: Checking logic correctness by comparing derived values against known rules or reference datasets.
- Consistency Across Domains: Validating key relationships and subject-level consistency across datasets (e.g., matching USUBJID across DM and EX).
- Controlled Terminology Checks: Ensuring values conform to allowed code lists, where applicable.

The QC engine runs after each domain is generated (e.g., EX, DM, IC) and logs in with any discrepancies or failures. If an error is detected—such as missing values, invalid data formats, or join mismatches—the error message is fed back into the LLM along with contextual metadata and the original prompt. The LLM analyzes the error, interprets the likely cause, and proposes a corrected version of the code. This enables a feedback-driven loop where the system can iteratively fix issues with minimal manual intervention.

For example, if the LLM-generated code fails due to a missing TRTFLAG column, the system will recognize the issue, regenerate a prompt specifying that TRTFLAG must be read from a certain file, and re-attempt code generation. This intelligent QC feedback loop not only improves accuracy but also increases efficiency by reducing debugging time for programmers.

Ultimately, this QC mechanism acts as a safeguard, ensuring that the AI-generated output adheres to regulatory and data quality standards before being finalized for downstream use.

CHALLENGES AND LIMITATIONS

Despite its potential, the proposed framework faces several challenges. Large Language Models (LLMs) can sometimes produce incorrect or inconsistent code, particularly when study specifications are ambiguous or poorly structured. Their general-purpose nature means they may lack deep understanding of clinical and regulatory nuances, which can impact accuracy. Additionally, variability in how different organizations format SDTM specifications makes standardization difficult. While the feedback loop improves automation, complex errors may still require human intervention. Lastly, integrating AI-generated outputs into validated, regulatory-compliant workflows demands rigorous quality control, documentation, and traceability to meet industry standards.

SUMMARY

This paper presents a framework that leverages Large Language Models (LLMs) to automate the generation of SDTM-compliant datasets for clinical trials. By transforming study specifications into structured prompts, the system enables LLMs to generate Python code that accurately implements the required logic for each domain. The process includes automated execution, validation, and an intelligent feedback loop for resolving errors, significantly reducing manual effort and programming time. While challenges such as model reliability, specification variability, and regulatory compliance remain, this approach demonstrates strong potential to modernize and streamline clinical data programming workflows. With continued refinement, such AI-driven tools could become integral to efficient, scalable, and compliant clinical trial data management.

REFERENCES

CDISC. 2021. "*Study Data Tabulation Model Implementation Guide (SDTMIG) v3.4*." Available at https://www.cdisc.org/standards/foundational/sdtmig.

AbbVie. (2025). Clinical Data Transformation: AbbVie's AI Journey. PharmaSUG 2025 Proceedings, SI-180.

PhUSE. 2022. "Automation in Clinical Data Standards Programming." Proceedings of the PhUSE US Connect 2022. Available at https://www.phuse.global.

Brown, Tom B., et al. 2020. "Language Models Are Few-Shot Learners." Advances in Neural Information Processing Systems, 33:1877–1901.

CONTACT INFORMATION

Do not change the heading style or the text "CONTACT INFORMATION" of the preceding heading. Do not change the text of the following paragraph. Replace all fields that are shown in angle brackets.

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

<Yanfen Zhu>
<Pacira Pharmaceuticals)>
<Yanfen.zhu@pacira.com>