ADaM AI: PoC on how to interact with CDISC Library API using Natural language

Saikrishnareddy Yengannagari

Bristol Myers Squibb

PHUSE SDE 2025

Introduction: The Challenge

CDISC Library API: Authoritative source for standards metadata (ADaM, SDTM, etc.).

Access Methods: Website navigation or direct API calls.

Problem: Direct API interaction requires technical skill (formulating calls, parsing JSON), creating a barrier for many clinical programmers/managers seeking quick info.

Question: Can we use Natural Language Processing (NLP) for easier access?

Proposed Solution: ADaM AI

Goal: Enable natural language queries for ADaM variable info (metadata, codelists).

Core Idea: Use AI as an *interface layer*, not a data generator.

Workflow:

- User asks question in plain English.
- AI extracts key parameters (e.g., variable name).
- Backend script calls CDISC Library API for *actual* data.
- AI formats the *factual* API response into a natural language answer.

Method: Stage 1 - SAS Foundation

Initial Step: Created adamgenius.sas macro.

Functionality:

- Takes adamvar (required), adamigversion, adamctversion (optional).
- Fetches latest IG/CT versions if not provided (via /mdr/products).
- Retrieves ADaM IG structure (/mdr/adam/adamig-...).
- Finds variable's dataset context.
- Fetches variable metadata (/mdr/adam/adamig-.../variables/...).
- Extracts codelist links & fetches terms (via %GetCDISCCodelist, handles ADaM/SDTM CT).

Output: Prints details to SAS results window.

Purpose: Established baseline functionality using familiar SAS environment.

```
/*-------------------------------------------------------------------------------------------------------------------/
    Get details of the variable from the API
/-------------------------------------------------------------------------------------------------------------------*/
filename varcheck TEMP;
proc http
    url="https://library.cdisc.org/api/mdr/adam/adamig-&adamigversion/datastructures/&dataset/variables/&adamvar"
    method="GET"
    out=varcheck;
    headers
        "api-key"="&cdiscapikey"
        "Accept"="application/json";
run;

libname varcheck JSON fileref=varcheck;
```

Proc http call to fetch adam variable metadata using SAS

Method: Stage 2 - Python Conversion

Goal: Translate SAS logic to Python (adam_genius.py) for broader integration.

Process: "Vibe Coding"

- Used AI code assistant (e.g., GitHub Copilot) due to SAS expertise > Python expertise.
- Provided SAS code sections to AI.
- Stated intent: Replicate logic in Python using CDISC API.
- Iteratively refined & validated AI-generated Python code against SAS output.
- Adapted to Python idioms (requests, json, argparse, dotenv).

Outcome: Python script mirroring SAS macro functionality, serving as the core data engine.

```
class ADaMMetadataRetriever:
    def get_variable_details(self, adam_variable, adamig_version):

        print(f"Fetching details for {dataset}.{adam_variable} (ADaMIG {adamig_version_hyphen})...")
        url = f"{self.BASE_URL}/mdr/adam/adamig-{adamig_version_hyphen}/datastructures/{dataset}/variables/{adam_variable}
        data = self._make_request(url)

        if not data:
            print(f"ERROR: Could not fetch details for variable {adam_variable} in dataset {dataset}.")
            return None

        # Extract key details (similar to SAS macro output)
        details = {
            "Variable": data.get("name"),
            "Label": data.get("label"),
            "DataType": data.get("simpleDatatype"),
            "Core": data.get("core"),
            "CDISCNotes": data.get("description"), # Or use 'definition' if more appropriate
            "Dataset": dataset, # Add the dataset context
            "ADaMIGVersion": adamig_version_hyphen, # Store the version used
            "CodelistLinks": [],
            "Codelists": [] # To store fetched codelist details later
        }
```

Python version of fetching adam variable metadata

Method: Stage 3 - AI Integration

Goal: Enable natural language input/output using AI (adamai.py).

Tool: OpenAI GPT-4o-mini.

Two-Step AI Process:

- **Variable Extraction:**
  - AI Prompt: Extract *only* the ADaM variable name (uppercase) from user query.
  - Example: “Tell me about subject age” -> AGE
  - Purpose: Focused task to minimize errors.
- **Explanation Generation:**
  - Input: Factual metadata from adam_genius.py.
  - AI Prompt: Explain the provided metadata conversationally.
  - Purpose: Format reliable data for user-friendliness.

Method: Architecture & Interfaces

Architecture: Layered approach to mitigate AI hallucination.

- User Query -> AI (Extract Var) -> Python Script (CDISC API Call) -> Metadata -> AI (Explain) -> User Response
- AI handles NLP, Script handles factual data retrieval.

Interfaces:

- **CLI (adamai.py):** For command-line usage.
- **Web App (streamlit_app.py):** User-friendly GUI.
  - Input query -> Shows extracted var -> Shows raw metadata -> Shows AI explanation.

CLI Examples

```
S C:\Users\yenganns\Downloads\cdisc\ADaM-Genius> python adamai.py "Is DTYPE codelist extensible?


🤖 AI Explanation:
Yes, the codelist for DTYPE is extensible, which means new terms can be added as needed. This flexib
ility allows for a variety of derivation techniques to be represented.
```

```
PS C:\Users\yenganns\Downloads\cdisc\ADaM-Genius> python adamai.py "Is DECADE a valid submission val
ue for AGEU variable, is codelist extensible for this variable?"

🤖 AI Explanation:
Yes, "DECADE" is a valid submission value for the AGEU variable because the codelist for this variab
le is extensible.

Here's the relevant metadata for your reference:

```
ADaM Variable Details: ADSL.AGEU
ADaMIG Version: 1-3
Label: Age Units
Data Type: Char
Core Status: Req
CDISC Notes: DM.AGEU
Codelist HREFs: /mdr/root/ct/sdtmct/codelists/C66781

Associated Codelist(s)
Codelist: Age Unit (AGEU) [C66781]
Extensible: Yes
Terms:
  TERM   Decoded Value

  DAYS   Day
  HOURS   Hour
  MONTHS   Month
  WEEKS   Week
  YEARS   Year
```
```

Streamlit webapp example

ADaM Genius

What would you like to know about an ADaM variable?

tell me about eotstt

Extracted Variable: EOTSTT

AI-Generated Explanation

The EOTSTT variable indicates a subject's End of Treatment Status, with possible values like COMPLETED, DISCONTINUED, or ONGOING. It is defined in the ADSL dataset and is a permanent character variable.

Full Metadata for Reference:

```
ADaM Variable Details: ADSL.EOTSTT
ADaMIG Version: 1-3
Label:        End of Treatment Status
Data Type:    Char
Core Status:  Perm
CDISC Notes:  The subject's status as of the end of treatment or data cutoff. Exam
Codelist HREFs: /mdr/root/ct/adamct/codelists/C124296

----------------------------------------------------------------------

Associated Codelist(s)

Codelist:     Subject Trial Status (SBJTSTAT) [C124296]
Extensible:   Yes
Terms:
  TERM                Decoded Value
  ------------------------------------------------------------
  COMPLETED           Complete
  DISCONTINUED        Discontinue
  ONGOING             Continue
```

Results

Functionality: Both CLI and Streamlit app successfully implemented.

- Accurate variable extraction from natural language.
- Reliable metadata/codelist retrieval via adam_genius.py (using CDISC API).
- Clear, conversational explanations generated by AI based on factual data.

Usability: Streamlit app provides easy-to-use GUI.

Hallucination Mitigation: Architecture successfully prevented AI from fabricating metadata; AI only formatted API-sourced data.

Performance: Acceptable for PoC; API calls are main latency source.

Enhancement: ADaMIG Q&A with RAG (`adamrag.py`)

- **Problem:** The API approach is great for specific metadata/codelists, but can't answer broader questions about ADaM principles or implementation details found *within* the ADaMIG document.
- **Solution:** Developed `adamrag.py` using Retrieval-Augmented Generation (RAG).
- **How it Works:**
  - Uses `LlamaIndex` library.
  - Loads and indexes the ADaMIG PDF document (e.g., v1.3) from a local directory.
  - Creates a vector store of the document content.
  - User asks a natural language question via command line.
  - The script queries the index, retrieves relevant text chunks from the PDF, and uses an LLM to synthesize an answer based *only* on the retrieved document context.
- **Benefit:** Complements the API tool by enabling queries about the textual content, examples, and guidance within the ADaMIG itself.
- **Example Query:** "What are fundamental principles of adam standard?"

Example RAG query

```
PS C:\Users\yenganns\Downloads\cdisc\ADaM-Genius> python adamrag.py "What are fundamental principles of adam stadnard"
ADaM datasets must adhere to certain fundamental principles as described in the ADaM Model document:
- ADaM datasets and associated metadata must clearly and unambiguously communicate the content and source of the datasets supporting
the statistical analyses performed in a clinical study.
- ADaM datasets and associated metadata must provide traceability to show the source or derivation of a value or a variable (i.e., th
e data's lineage or relationship between a value and its predecessor(s)).
- ADaM datasets must be readily usable with commonly available software tools.
- ADaM datasets must be associated with metadata to facilitate clear and unambiguous communication. Ideally the metadata are machine-
readable.
- ADaM datasets should have a structure and content that allow statistical analyses to be performed with minimal programming. Such da
tasets are described as "analysis-ready." ADaM datasets contain the data needed for the review and re-creation of specific statistica
l analyses. It is not necessary to collate data into analysis-ready datasets solely to support data listings or other non-analytical
displays.
```

Discussion & Key Takeaways

AI as Interface: Successfully enhances usability of technical APIs (CDISC Library).

Hallucination Mitigation: Layered architecture (AI for NLP, Script for API data) is crucial for trust in regulated environments.

"Vibe Coding": Pragmatic use of AI assistants for code translation (SAS -> Python), but requires rigorous validation.

Target Audience Benefit: Lowers barrier for SAS programmers/managers to access ADaM info.

Public GPT: "Adam Genius" Custom GPT demonstrates practical application and accessibility (for Plus users).

Limitations: API dependency, PoC performance, simple query handling.

Conclusion & Future Work

- **Conclusion:** AI can be a powerful, trustworthy interface for technical APIs like CDISC Library when architected correctly (separating NLP from data retrieval).
- **Value:** Improves accessibility and usability for clinical teams.
- **Future Work:**
  - Expand to other CDISC standards (SDTM, SEND).
  - Handle more complex queries.
  - Optimize performance (caching).
  - Integrate into development/review platforms.

Thank You!