

SASBuddy: Enhancing SAS® Programming with Large Language Model Integration

Karma Tarap, Analysis Standards & Innovation

Derek Morgan, Global Macros and Technical Innovation

To be Covered

- What Is SASBuddy?
- Challenges of LLM's In Clinical SAS Programming
- SASBuddy Solutions
- SASBuddy: The Process
- Next Step
- Where Are We Headed?
- Summary

What Is SASBuddy?

- Friendly SAS programming assistant
- Facilitates SAS programming through Large Language Models (LLMs)
- Empowers users, particularly those with limited SAS coding expertise, to generate precise SAS code efficiently
- Interprets natural language input
- Can provide contextually accurate SAS code tailored to specific dataset

What's a Large Language Model?

- A subset of machine learning models
 - Understands and generates human language
 - Trained on vast amounts of text
 - Leverage statistical methods to predict subsequent words in a sequence based on the preceding ones
- Many advanced language models: BERT, GPT-2 and GPT-3
 - Development continues with the advent of better, faster technology

Challenges of LLM's In Clinical SAS Programming

- SAS is not open source
 - More publicly available code across many solutions for open source
 - Smaller representation in LLM training databases
 - Without equal context, LLM will have trouble generating relevant and precise code
 - Example (not representative numbers):
 - Get a more precise answer w/1M datapoints than 1K
- Difficult to write a SAS program without knowledge of data structures and available macros

SASBuddy Solutions

Data-Relevant Code Generation

- Create rich metadata for query from datasets
- Ensures generated code is syntactically correct and contextually aligned with specific data structure(s) it will analyze or manipulate
- Arms LLM with necessary context to understand where and how to look for specific data points
- Identify appropriate variables or columns for a given analysis task

Standard Macro Utilization

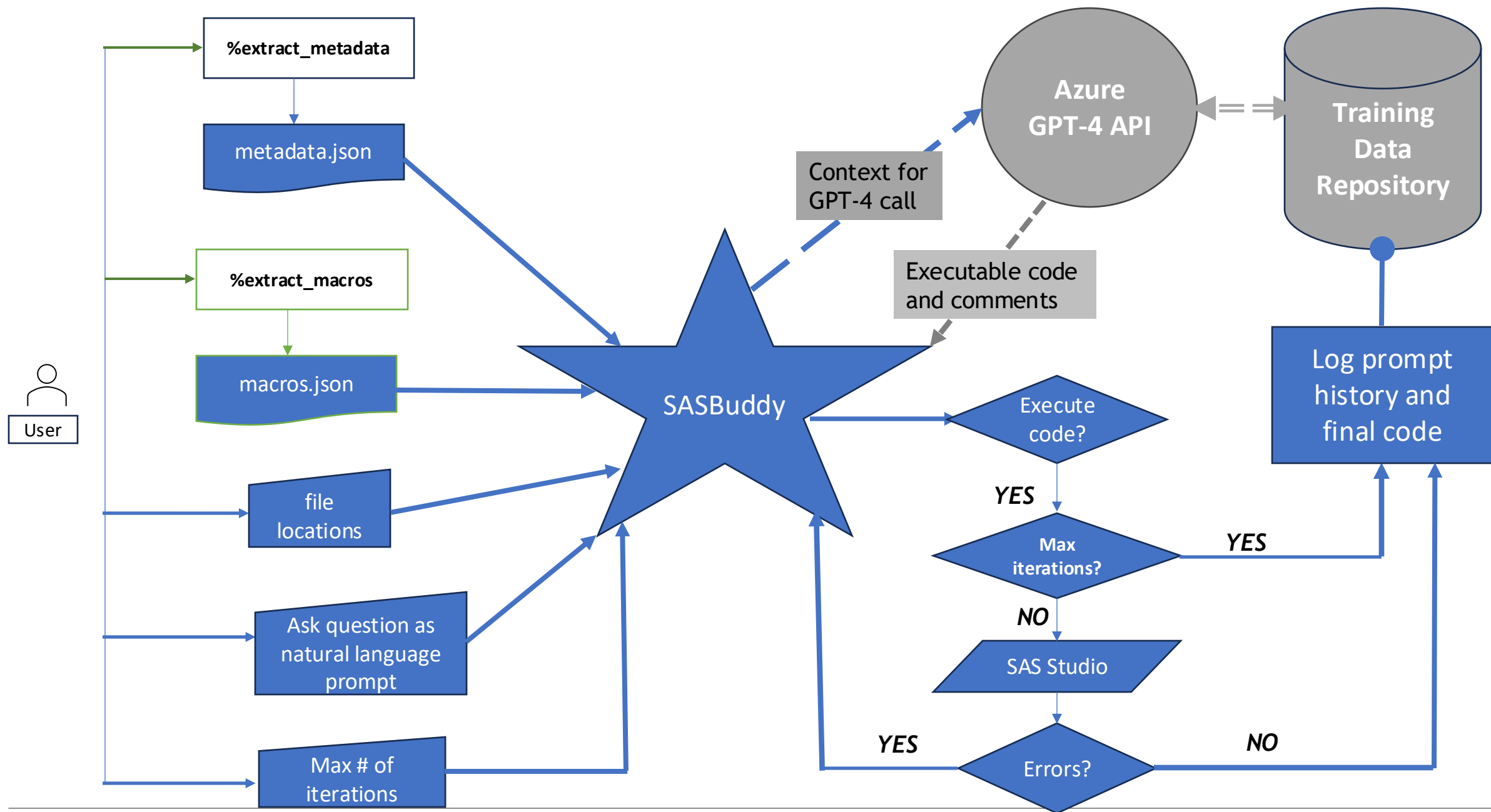
- Using a JSON function schema, make macros available to the process
 - Parsed from Doxygen macro headers
- JSON schema provides LLM with detailed blueprint of available macros
 - Including functions, parameters, and usage conventions
- Allows more reliable macro call generation
 - Ensures code produced adheres to established coding practices
 - Leverages efficiencies macros are designed to provide

Automated Feedback Loops

- Critical component of iterative improvement process
- After generating and executing SAS code, capture logs including errors during execution
 - Logs are sent back to LLM, which analyzes errors and adjusts the generated code trying to fix encountered issues
 - Enhances the accuracy of code over time and contributes to the LLM's learning
 - Improves generation of error-free SAS code in future queries

Versioning and Fine-Tuning

- Record prompts and corresponding responses from successful executions
- Historical record logged into a file, creating repository of effective queries and SAS code snippets
- Invaluable resource for fine-tuning future responses
- Draws on past successes to improve relevance and accuracy of generated code



SASBuddy: The Process (1/2)

1. Run macro to scan datasets of interest
 - Creates file named metadata.json detailing the metadata
 - Includes information about data structure and attributes data, but removes confidential or patient-identifiable information
2. Run macro to analyze the Doxygen headers of standard macros used within organization
 - Creates macros.json file to serve as catalog of available macros, detailing usage, parameters, and functionality.
3. User poses question in natural language
 - Like asking a colleague for help with a SAS programming task
 - Specifies location of metadata and macro JSON files
4. SASBuddy formulates query to the Azure GPT-4 API.
 - Query crafted using reflexion and few-shot learning techniques
5. API prompted to generate executable SAS code
 - Includes explanations and comments to add natural language guidance on its use and intention

SASBuddy: The Process (2/2)

6. If user sets execute parameter to "Y"
 - SASBuddy runs generated SAS code
 - SASBuddy captures any error messages and sends them back to the Azure API for further refinement of code
 - This loop of execution, error capture, and refinement continues until code runs without errors or a predefined maximum number of iterations is reached
7. After code executes successfully without errors:
 - If feedback option selected, log successful prompt and response sequence into training data repository
 - Repository contains invaluable data for fine-tuning system

Leads to improved ability to generate accurate and useful SAS code in future queries

Next Step

- Originally designed as macro call in a SAS program running in SAS Studio:

```
%SASBuddy(question=how do you calculate the mean,  
output_file=&srcdir/basic.sas);
```

SAS Buddy as a SAS Studio Task

The screenshot displays the SAS Buddy application window, which is integrated into the SAS Studio environment. The window is titled "SASBuddy" and features a tabbed interface with "Settings", "Code/Results", and "Split" options. The "GENERATE" tab is active, showing a form for generating SAS code. The form is divided into several sections:

- DATA SOURCE**: A section for choosing the data source. It includes a checkbox for "Use existing metadata.json file" and a text input field for "Metadata File Path".
- OR EXTRACT NEW METADATA**: A section for extracting new metadata. It includes a "Select Datasets to Analyze" button, a "Column" input field, and text input fields for "Domain Path*", "Code Path*", and "Codelist Path*".
- QUESTION**: A section for entering a question. It includes a text input field labeled "*Enter your question*:".
- OUTPUT OPTIONS (OPTIONAL)**: A section for output options. It includes a text input field for "Output File Path:", checkboxes for "Append to existing file" and "Execute generated code", and a text input field for "Macros JSON Path:".

The "Code/Results" tab is also visible, showing a code editor with the following placeholder text:

```
1 /*  
2 *  
3 * Code cannot be generated beca  
4 * options are not set:  
5 *  
6 * Enter your question*:  
7 *  
8 *  
9 */  
10  
11
```

Ongoing Work

- Evaluate code generation accuracy
 - Does it address clinical SAS programming needs well?
 - Can it address needs across varied scenarios?
 - Benchmarks include code correctness, efficiency, and adherence to programming standards
 - Compare SASBuddy output against established benchmarks and real-world task outcomes
- Investigate potential to adopt more autonomous operational capability
 - Dynamic query refinement
 - Strategic analysis suggestions
 - Proactive error detection
 - Fine tuning the model
 - Incorporate programming nuances and clinical data analysis requirements
 - Systematic collection of high-quality, real-world usage data

Where Are We Headed?

**Turn SASBuddy from a simple
command-executing tool into a
proactive coding partner**

Summary

- SASBuddy:
 - Uses LLM's to generate error-free SAS code from natural language queries
 - Macros provide task-specific information on dataset structure and attributes, and organization-standard macros as additional input
 - Iteratively generates code, executes it, and captures errors until success or user-defined stop point.
 - Captures error-free SAS code it generates and adds it to a learning database

Thank You!

