PharmaSUG 2025 - Paper AI-246

Incorporating LLMs into SAS Workflows

Samiul Haque and Sundaresh Sankaran, SAS Institute Inc.

ABSTRACT

Python integration with SAS provides developers and convenient way to incorporate LLM frameworks into SAS Studio workflow. Through collaborating with Python developers LLMs can be easily incorporated into different SAS projects. We developed a Retrieval Augmented Generation (RAG) framework that can be easily incorporated into SAS Studio and leveraged by any SAS Viya user. We showcase how different component of RAG architecture can be implemented within SAS platform by leveraging proprietary and opensource technologies. We discuss how a generic RAG architecture can be used for different use cases such as Adverse Event Detection, Structured Data extraction, Data Mapping, Protocol Search, and Patient Matching. Finally, we provide a blueprint for data scientists to implement their own RAG architecture in SAS Studio for their custom use cases.

INTRODUCTION

This paper delves into the integration of Large Language Models (LLMs) with SAS Studio through Python interfaces. This integration helps us enhance SAS workflows with state-of-the-art Artificial Intelligent (AI) capabilities. Utilizing Python's rich libraries along with SAS's robust analytics, we present a Retrieval Augmented Generation (RAG) framework designed for straightforward adoption by SAS Viya users.

The RAG framework, which includes components for retrieval, augmentation, and generation, is implemented in SAS Viya using a blend of proprietary and open-source technologies. This adaptable architecture caters to a wide array of applications such as adverse event detection, structured data extraction, data mapping, protocol search, and patient matching. By merging the Natural Language Processing (NLP) and analytical capabilities offered by both Python and SAS, this approach enables data scientists to manage and analyze data more intelligently and efficiently.

Moreover, we offer a detailed blueprint for adapting RAG solutions within SAS Studio, tailored to meet specific needs of practitioners. This framework equips data scientists and developers with the tools to fully exploit the capabilities of LLMs in the SAS environment, promoting deeper analytical insights, scalable solutions, and improved decision-making processes, all within the familiar confines of SAS workflows.

By fostering this integration, SAS users are poised to tap into new avenues for innovation, effectively integrating latest LLM into existing workflows, without significantly disrupting existing operational frameworks.

RETRIEVAL AUGMENTED GENERATION (RAG)

Retrieval Augmented Generation (RAG) enhances Large Language Models (LLMs) results by addressing their intrinsic need for context. LLMs, by design, require substantial and relevant contextual data to generate meaningful and accurate outputs. RAG addresses this need through intelligent context retrieval from a vast corpus of data, thus facilitating generation of more accurate and contextually relevant responses.

Integrating RAG in a SAS environment hinges upon SAS's capability to manage and provide high-quality, relevant contextual data. SAS's Natural Language Processing (NLP) and data management capabilities ensure that the context used in RAG is not only pertinent but also of high integrity, enhancing overall effectiveness of LLMs. This integration allows seamlessly transfers data and context over to LLMs, ensuring that the output is both reliable and addresses specific user needs. By providing a structured framework for contextual data retrieval, SAS empowers RAG to significantly optimize the performance of LLMs in complex analytical tasks.

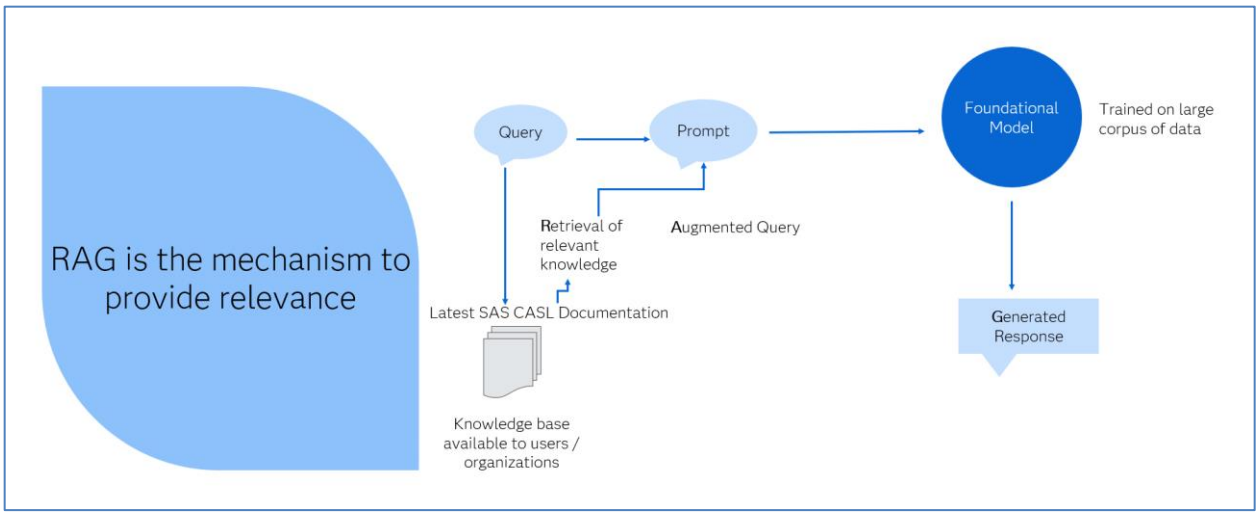

Figure 1 RAG provides relevant context to foundational model

We consider a hypothetical scenario where a user requires clarification about SAS Cloud Analytics Services Language (CASL) from SAS Documentation, which is considered a knowledge base. Figure 1 shows how RAG, upon a user query, retrieves relevant information from the knowledge base and provides this information as context, along with the user's question to the LLM.

Breaking down the process shown in Figure 1 into stages:

1. **Query**: The user initiates the process with a query, seeking information or an answer.
2. **Retrieval of Relevant Knowledge**: RAG accesses the SAS documentation, serving as a knowledge base, to retrieve information pertinent to the query. This ensures that the context provided is directly relevant to user needs.
3. **Augmented Query**: The relevant information retrieved from the SAS documentation is combined with the original query. This augmented query enriches the context available to the foundational model, enabling it to understand and address the query more effectively.
4. **Response Generation**: Using the augmented query, the foundational model, which is trained on a broad corpus of data, generates a response. This response is not only informed by the model's extensive training but is also specifically tailored to include the latest and most relevant information provided by the SAS documentation.

This workflow exemplifies how RAG leverages specific, real-time information from a structured knowledge base to enhance the accuracy and relevance of responses generated by AI models.

VECTOR DATABASE

RAG frameworks make use of a retrieval mechanism usually based on a vector store or a vector database (though other database models also lend themselves to this framework). A vector database efficiently indexes text data by converting text into high-dimensional vectors. These vectors retain semantic information, allowing quick similarity searches among text chunks. This capability is vital for applications requiring fast retrieval and comparison of text-based information. In this work we leveraged an open source vector database, ChromaDB.

IMPLEMENTING RAG IN SAS STUDIO

SAS provides versatile procedures like PROC HTTP and PROC Python that enable direct interaction with foundational models (LLMs). Proc Python facilitates execution of Python code within SAS workflows. PROC HTTP facilitates querying a foundational model making HTTP requests to the model's Application

Programming Interface (API). This capability is crucial for fetching real-time data or model responses that can be further processed within SAS. While direct querying through PROC HTTP is effective, using robust frameworks like LangChain or LlamaIndex make it easier to build LLM applications. These packages offer comprehensive tools and functions specifically designed for LLMs, making it easier for SAS users to develop complex applications tailored to their needs. These frameworks handle various aspects of LLM implementation.

TEMPLATED CUSTOM STEP FOR RAG

A Custom Step is a GUI interface used within SAS studio to allow easier interaction with SAS workflows. We introduce a custom step that indexes text documents into ChromaDB and connects to an Azure OpenAI instance. Users can modify the custom step to connect to any other LLM instance as well. A high level architecture diagram of the system is depicted in Figure 2.

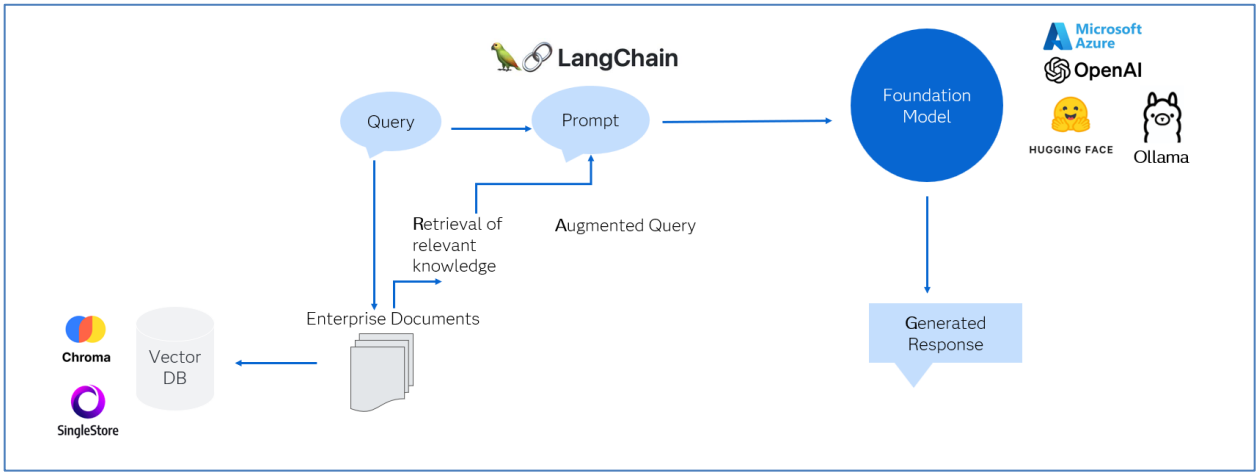

Figure 2 RAG Implementation

Figure 3 shows the front end of the custom step. Users have the option to select a single pdf file, an entire folder containing input files, a Comma Separated Value (csv) file, pandas data frame, or a SAS dataset. Given information about source files, the program driving the custom step will indexes input documents into ChromaDB, making them readily searchable based on their content vectors.

The same custom step also connects to an LLM. Users edit system prompts based on their use cases. Users also provide a user prompt (reflecting the question asked of the LLM) in this interface and can leverage various prompt engineering techniques to improve the quality of their query.

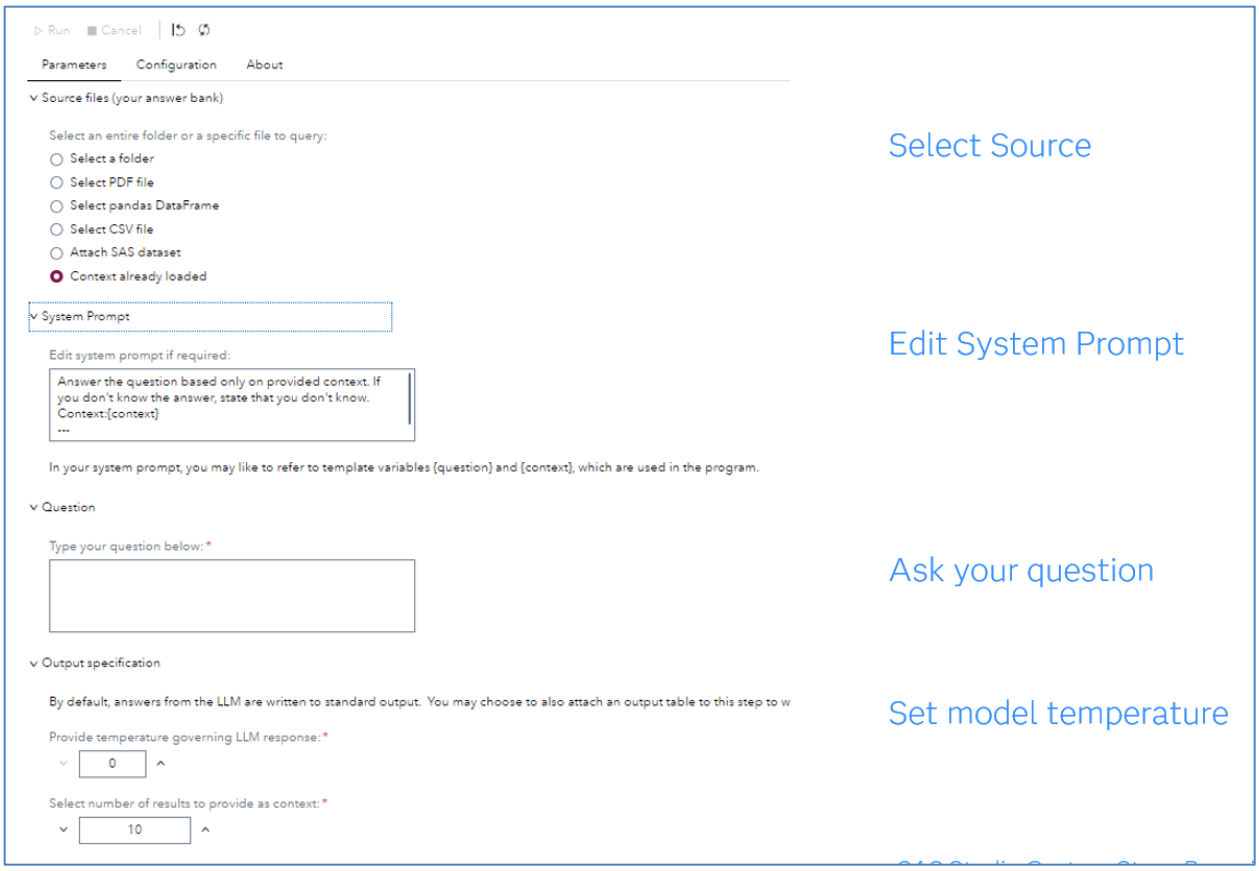

Figure 3 Custom Step Front End

Additionally, the interface provides flexibility in tailoring system interactions, offering a field where users can input specific questions for the Large Language Model (LLM). This customization enables the LLM to generate responses or perform tasks aligned with the user's particular needs, enhancing the applicability and functionality of the system across various use cases. Detailed instructions for using the custom step can be found in https://github.com/samiulhq/LLM-RAG-SAS

Results are presented to the user in two ways. One is through the results window using the Output Delivery System (ODS) in Question – Answer format as shown in in Figure 4. The other is through an output dataset designated for receiving the LLM response, along with the question. The output dataset also contains the context that was retrieved in response to the query, along with a relevance score indicating relevance to the question. This is beneficial for evaluation of the LLM's response downstream.

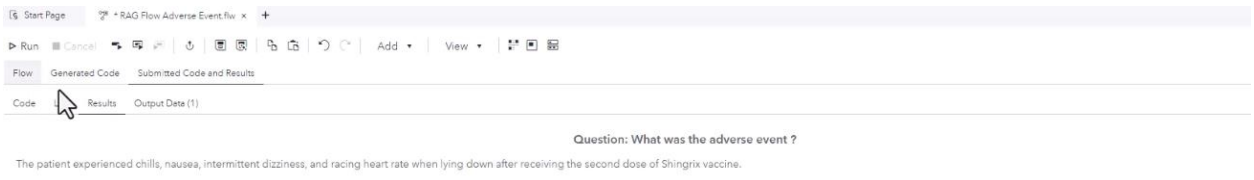

Figure 4. Example output of RAG step

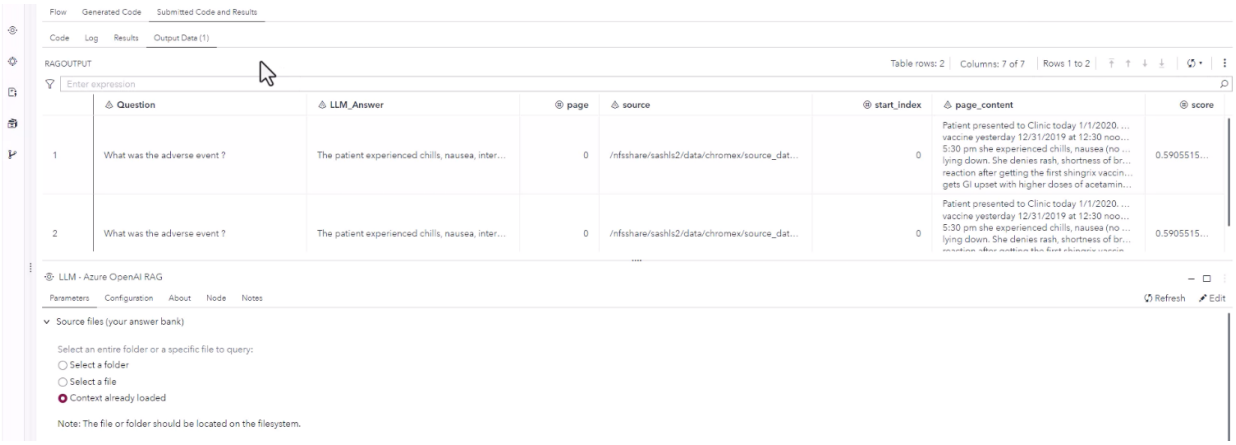

Figure 5. Example output table from RAG

EVALUATION OF LLMS

LLM evaluation is important for ensuring trust in AI results. LLMs are inherently black-box models and not very transparent in terms of providing details on their computational logic. This raise concerns around whether their results can be trusted, are reliable or can be controlled.

Trust in LLMs through RAG depends on the sufficiency of the context provided. This is under the control of the organization. Further to this, trust also depends on whether the LLM used relevant context in framing its answer and whether the answer was correct or not (raw accuracy).

Enterprises can use internal mechanisms such as NLP-based distance measures and external mechanisms known as ‘evaluator LLMs’ or ‘judge LLMs’ to measure accuracy for LLMs. An example of an LLM RAG evaluation is provided below in Figure 6. Three scaled measures – that of accuracy, relevancy, and sufficiency – are represented on the dashboard against a cross section of labelled interactions, which help assess the trust that an enterprise could put in the LLM under question.

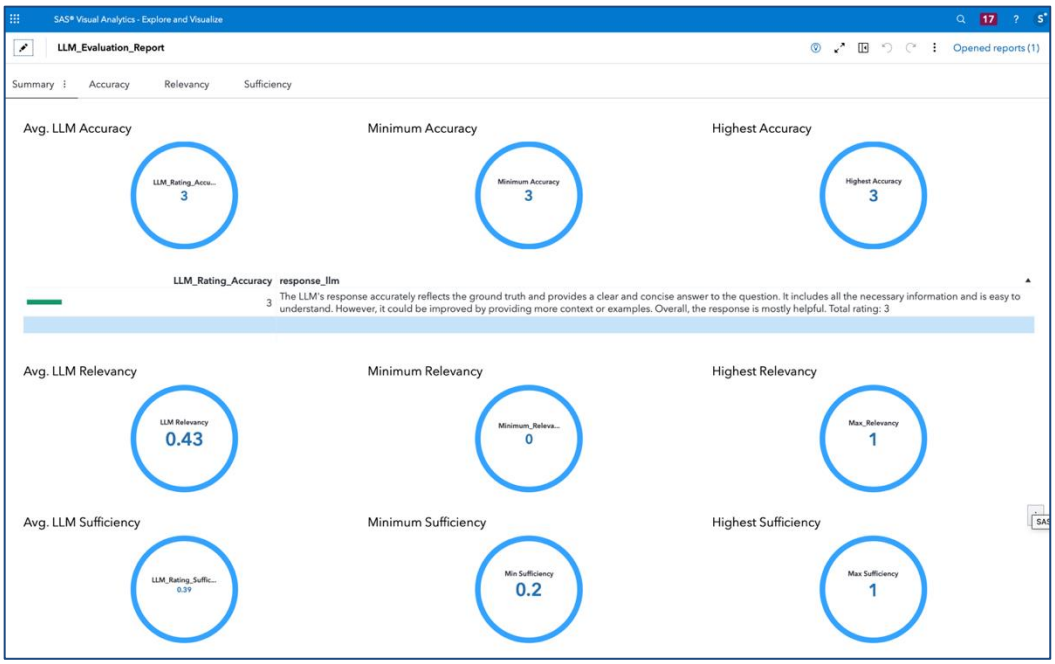

Figure 6. Evaluation of RAG results from an LLM across Context Sufficiency, Relevancy and Accuracy.

APPLICATIONS

The RAG approach can be applied to a wide range of use cases across the life sciences domain. Example applications include **adverse event detection**, **patient matching**, **structured data extraction**, **protocol search**, and **data mapping**—all of which benefit from combining domain-specific retrieval with generative language capabilities.

CONCLUSION

In this paper, we explored how Retrieval Augmented Generation Frameworks can help in providing accurate and contextual information in concert with Large Language Models (LLMs) and aid many life sciences applications in the field of Adverse Event Detection, Structured Data extraction, Data Mapping, Protocol Search, and Patient Matching, among others. At the same time, issues of trust and accuracy are important and require analytical frameworks for evaluation and visualization of LLM performance at a broader level. Enterprises are encouraged to use RAG in addition to other frameworks to maximize their ROI from LLMs.

REFERENCES

1. Sankaran, Sundaresh & Eid, Sherrine, “Three Pointers for Effective and Accurate LLM Integration”, Phuse US Connect 2025 Paper ML-22, https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/US/Orlando/PAP_ML22.pdf
2. Haque, Samiul & Sankaran, Sundaresh, “LLM – Azure OpenAI Retrieval Augmented Generation”, SAS Studio Custom Step, GitHub repository, https://github.com/sassoftware/sas-studio-custom-steps/tree/main/LLM%20-%20Azure%20OpenAI%20RAG ; https://github.com/samiulhq/LLM-RAG-SAS

CONTACT INFORMATION

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

Samiul Haque
SAS Institute
919 531 2797
Samiul.haque@sas.com
www.linkedin.com/in/samiulhaque

Sundaresh Sankaran
SAS Institute
919 531 4921
Sundaresh.sankaran@sas.com
www.linkedin.com/in/sundareshsankaran