PharmaSUG China 2024 - Paper AI-10052

Enhancing SAS Programming Efficiency with Large Language Models (LLMs)

Guangyong Li, Junkai Zhao, Everest Clinical Research

ABSTRACT

As statistical programmers, we often encounter repetitive tasks in programming, such as commenting code by sections and organizing code layout. We are also always challenged to reference code that already exists and has been written by others. Examples include understanding the functionality of the code correctly and in a time-saving manner, and the definition and use of macro parameters.

AI (Artificial Intelligence) as a highly popular topic in recent years, has a wide range of applications, including language understanding, image recognition, speech recognition, etc. Among these, the most widely used is language understanding, which is also commonly known as AI chatbot, and the essential part of language understanding is LLM. This article explores and describes how to incorporate the leading LLMs to help statistical programmers better analyze and refine the writing and usage of SAS® code.

Based on compliance considerations of arithmetic limitations and data privacy, the article firstly discusses the possibilities of deploying LLMs on PCs and on servers respectively and provide technical details of the deployment process and demonstrating results. And secondly the article provides practical guidance for statistical programmers to experience the power of LLMs, increasing the efficiency they work with SAS code.

By streamlining programming tasks and enhancing code understandability, LLMs have the potential to transform the field of SAS programming, enabling programmers to focus on higher-level tasks and drive more value from clinical data.

INTRODUCTION

A Large Language Model is a powerful type of AI designed to understand and generate human-like text. It is trained with enormous data sets containing information from papers, websites, etc. The training helps it learn structure and grammar of human languages. When giving it a prompt, it predicts the responses to generate within the given context. It can answer questions, translate languages, and even address programming challenges. The most advanced LLMs like GPT-4 from OpenAI and models from Meta AI can automate tasks, generate code, and provide valuable insights to significantly enhance programming efficiency.

Running LLMs on a local PC requires substantial computing power from CPU (Central Processing Unit) and GPU (Graphic Processing Unit). The requirements differ based on the size of the model, however, even the smallest models require very high hardware resources. Typically, it requires at least an Intel Core i7 CPU, and a GPU with 16 GB VRAM (Video Random Access Memory) such as RTX 3090 or equivalent. For users equipped with high-end computers, deploying LLMs locally allows them to utilize these models without relying on an internet connection. This setup not only ensures continuous access to LLM capabilities but also enhances data privacy by processing everything on the user's own machine. For users with less powerful computers (e.g., company-issued working laptop in most cases), server deployment offers a solution, they can leverage the computational strength of a dedicated server. This approach mainly provides protection for the company's regulatory and privacy policies. When in use, sensitive data will only be uploaded to the company's server, not to any third party. It protects intellectual property, data, and documentation security, while also being convenient and easy to maintain and update. Moreover, it ensures that even users with limited hardware resources can benefit from LLMs. A server can support multiple users simultaneously, making it an efficient use of resources which is particularly beneficial for enterprises, rather than for individual users. Our solution uses Featurize (a Virtual Machine provider) to simulate the enterprise server environment, exploring the deployment of LLMs to help improve SAS programming efficiency in the pharmaceutical industry.

LOCAL DEPLOYMENT

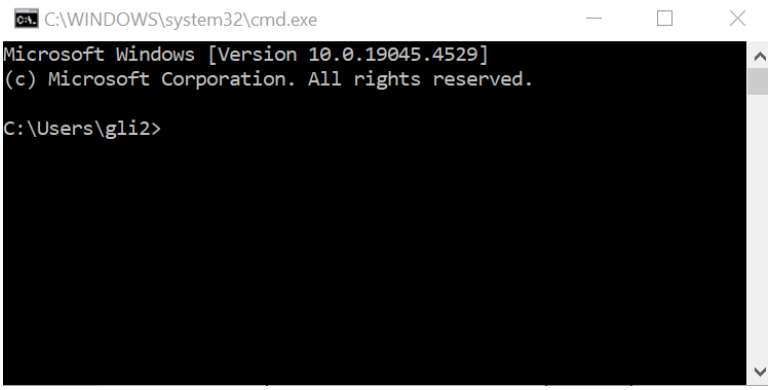
One effective solution to deploy LLM locally is to use Llamafile. Llamafile is a single executable file, which is combined with a model storage file (with a GGUF extension) and an executable program, it can be directly downloaded from the internet. Users simply download and execute the file to distribute and run the model without installation or setting up an additional environment. It supports various operating systems such as macOS, Windows, Linux, etc. Its advantage lies in its convenience of use, making it easier for more people to access LLMs locally. In this section, we demonstrate the possibility of local deployment by running the model <Meta-Llama-3-8b-Instruct> on a Windows system by using a downloaded Llamafile from ModelScope. The model is developed by Meta AI with size of 8B (i.e., 8 billion parameters), it is lightweight yet with stable predictions.

The size of a language model, indicated as 8B, 32B, 70B, etc., refers to the number of parameters the model has, which are the parts of the model learned from the training data and used to make predictions. With the increase in parameters, the model can capture more complex patterns and nuances in the data, potentially improving its performance on a wider range of tasks. Considering our limited hardware resources, and given our main goal is to demonstrate the possibility of local deployment, we chose only the 8B model for the demonstration.

In general, a Llamafile contains two sub-files. In this example, the Llamafile of <Meta-Llama-3-8b-Instruct> contains:

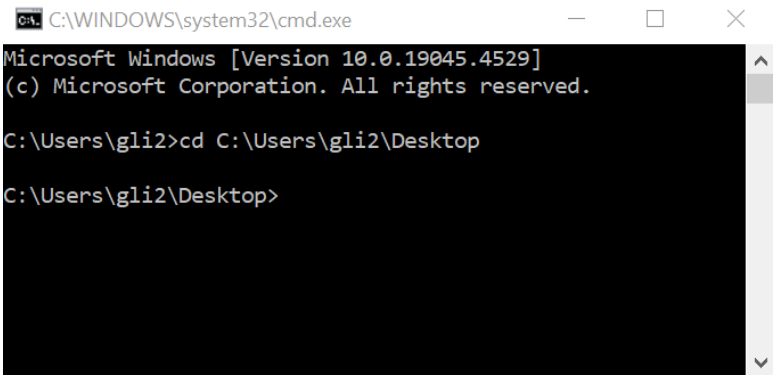
- Meta-Llama-3-8B-Instruct.Q4_0.gguf – stores the model
- llamafile-0.6.2.exe – to execute the .gguf file.

On Windows, press <Windows + R>, and then type <cmd> to open the Command Prompt (Display 1):



Display 1. Main Window of Command Prompt

Type <cd [File Location]> to navigate to the location of Llamafile (Display 2). In this example, they are located at Desktop:

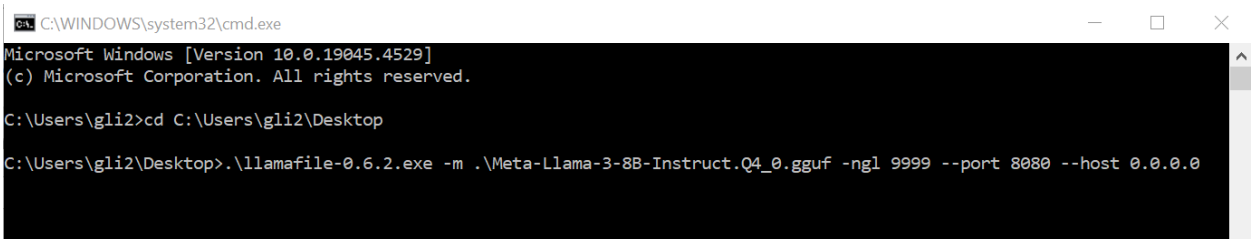


Display 2. Navigating to Desktop

Type the command below and click ENTER to execute the Llamafile (Display 3):

<.\llamafile-0.6.2.exe -m .\Meta-Llama-3-8B-Instruct.Q4_0.gguf -ngl 9999 --port 8080 --host 0.0.0.0>

- **.\llamafile-0.6.2.exe**: The executable being run.
- **-m .\Meta-Llama-3-8B-Instruct.Q4_0.gguf**: Specifies the file that the executable should use.
- **-ngl 9999**: Indicates the proportion of computational resources to be allocated to GPU. 0 indicates none, 9999 indicates full GPU allocation (by default).
- **--port 8080**: Specifies the port number '8080' for networking.
- **--host 0.0.0.0**: Specifies the host address '0.0.0.0' to listen on all interfaces.



Display 3. Executing Llama File

Once done, the model <Meta-Llama-3-8b-Instruct> will be running on the local environment, visit http://127.0.0.1:8080 by Internet Browser to view the Chatbot (Display 4) from the Web UI (User Interface).

Display 4. Web UI of Meta-Llama-3-8b-Instruct model

In the Web UI, there are various settings and presets. The explanation of a few main configurations are as follows:

- **Prediction**: The ability to generate or predict the next word or sequence of words based on the input it receives. This involves understanding the context provided by the input and generating text that follows logically from that context.
- **Temperature**: Controls the randomness of predictions. A lower temperature makes the model more confident and conservative in its predictions. A higher temperature introduces more randomness, leading to more diverse and sometimes less predictable outputs.
- **Penalize repeat sequence**: Discourage the model from generating repeated sequences of words or phrases in its output.

Undoubtedly, adjusting them will partially increase the speed of computation and accuracy of responses. However, our goal is not to fully understand these settings. Under the default settings, we can try interacting with the Chatbot by starting with a simple prompt (Table 1) to see how it works:

User: Could you tell me something about SAS?
Llama: Ahah! Sure thing! SAS (Statistical Analysis System) is a software suite for data manipulation, analysis, and visualization. It's widely used in various fields like business intelligence, research, healthcare, finance, etc. Would you like to know more about its features or applications?

Table 1. Simple prompt

The response seems reliable. However, it takes 14 seconds for the full response to be generated which is not satisfying. Try another prompt with a simple SAS DATA step (Table 2).

User: Could you help me understand my SAS code below? data adsl; set adam.adsl; if saffl="Y"; run;
Llama: Hi there! I'd be happy to help you understand your SAS code. It looks like this is a simple data step that's reading in a dataset called "adam.adsl" and creating a new dataset called "adsl". The if statement is checking if the variable "saffl" has the value "Y", but it doesn't seem to be doing anything with those observations. Is there something specific you're trying to accomplish or are you just wanting me to explain what this code does? Let me know and I'll do my best to help!

Table 2. SAS related prompt

The model correctly explained the purpose of the SET statement, which is to read <adsl> data set from the library <adam>. However, its understanding of the <if> condition is apparently not sufficient and can be quite misleading. It takes 38 seconds this time for the full response to be generated under such computational power, which is not efficient enough to meet our needs.

Based on the command <-ngl 9999>, the computational power should be entirely provided by the GPU. However, due to the compatibility issues of the GPU we have, it is not available to participate in the computation, therefore, all computational power is currently provided by the CPU, which is an Intel Core i5 with 16GB of memory. This is the average level for most PCs, therefore, due to the hardware limitations, deploying LLMs locally cannot address the needs of most users, and is not highly feasible.

SERVER DEPLOYMENT

In fact, local deployment of LLMs requires very high demands on CPU and GPU. The size of the model significantly affects hardware requirements. In the examples above, we only demonstrated an 8B model, but both speed and accuracy were unsatisfying. To enhance accuracy, models with larger sizes like 70B

are undoubtedly a better choice. However, most users' computer hardware cannot meet these requirements. Therefore, deploying LLMs on servers with powerful CPU and GPU can effectively address this challenge, especially given the limited hardware resources commonly found in company-issued laptops.

To achieve server deployment, we use the following tools:

- **Featurize**: Featurize is a VM (Virtual Machine) provider that offers VM rental services for machine learning algorithm development environments. Among similar providers, it has better solutions and optimizations, which can be used to simulate company server solutions.
- **Xinference**: Xinference provides various open-source large models, such as language models, speech recognition models, and multimodal models. These models can run directly either on the cloud or locally on a computer.
- **Dify**: Dify is an open-source LLM application development platform. It is LLM-agnostic that can support multiple LLMs without being dependent on any single one, regardless of how the models to be used are changed in the future, the logic of the AI application itself can be reused. Once the model we are interested in is running on Xinference, we need to use Dify to embed it into AI applications, such as common AI chatbots.
- **Docker**: Docker is a platform which uses containerization technology to allow developers to package up an application with all its dependencies into a single unit. Docker is a prerequisite tool for Dify in our deployment procedure.

The overall procedure is shown as Figure 1:

Figure 1. Overall Procedure

We chose to rent VMs just to explore the feasibility of server deployment. In most companies, it is normal to have their own servers, which are often more secure, more powerful, and more efficient at running LLMs.

Featurize offers VMs with different levels of GPUs for use, such as RTX3060/3090/4090/A6000, etc. Through testing, we found that running LLMs smoothly requires at least 20 GB of VRAM, thus, we opted to rent a machine with a RTX 4090 GPU, which is already very high-end in the market. The A6000 has higher VRAM and better performance but comes with a higher price. The price and other configurations parameters of this VM are as follows:

- Price: 2.78 CNY per hour
- VRAM: 24 GB
- CPU: 14 Cores
- Memory: 64.4 GB
- Disk Space: 401 GB

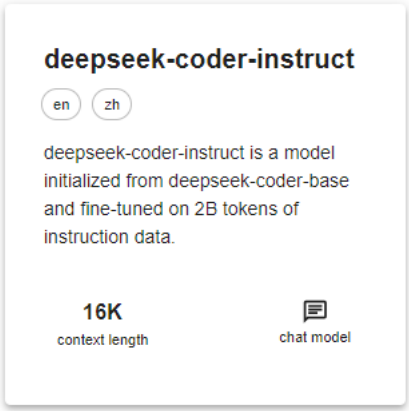
Next, enter a few commands directly in the Command Prompt (Terminal for MacOS) to use SSH to connect to the VM. In subsequent steps, such as installing inference tools, deploying models, and

composing Dify, all operations will need to be performed in the Command Prompt. The commands to use in each step are summarized in the table (Table 3) below:

Step	Purpose	Command	Explanation of Command
1	Configure environment for Docker	sudo apt update	Update server software dependencies to ensure subsequent steps won't encounter environmental issues.
		sudo apt install docker-compose-plugin	Install the Docker Compose plug-in using APT package manager.
		docker compose version	Display the Docker Compose version information to confirm a successful installation.
2	Install Xinference to run model	pip install "xinference[all]"	Install Xinference python package along with all its dependencies.
		XINFERENCE_MODEL_SRC=modelscope xinference-local --host 0.0.0.0 --port 9997	Start the 'xinference-local' service.
		sudo featurize port export 9997	Allow access to XINFERENCE via Internet Brower.
3	Compose Dify and start Docker container to allow Web UI creation	cd /home/featurize/work sudo docker load -i cuda.tar sudo docker load -i dify-api.tar sudo docker load -i dify-sandbox.tar sudo docker load -i dify-web.tar sudo docker load -i nginx.tar sudo docker load -i postgres.tar sudo docker load -i redis.tar sudo docker load -i squid.tar sudo docker load -i weaviate.tar	Navigate to WORK directory, unzipped pre-uploaded Dify image files (.tar). The WORK directory is a storage feature provided by Featurize, which is equivalent to a cloud drive, used for long-term file storage. Note: We pre-uploaded the image files needed for Dify Compose in Featurize's cloud drive (under WOK repository) to simplify the deployment process and avoid potential issues caused by network connection that might prevent downloading files from the Dify server. If you need to try this step, please contact us via email, and we can provide the files.
		git clone https://github.com/langgenius/dify.git	Extract the Dify source code from Github for subsequent Docker container initialization and composition. This action is necessary for the first-time deployment. Once it is done, the source code stores in the Featurize's cloud drive, and no longer needed for subsequent deployments.
		cd dify/docker docker compose up -d	Navigate to Docker files repository, start Docker container.
		featurize port export 80	Allow access to Dify via Internet Browser.

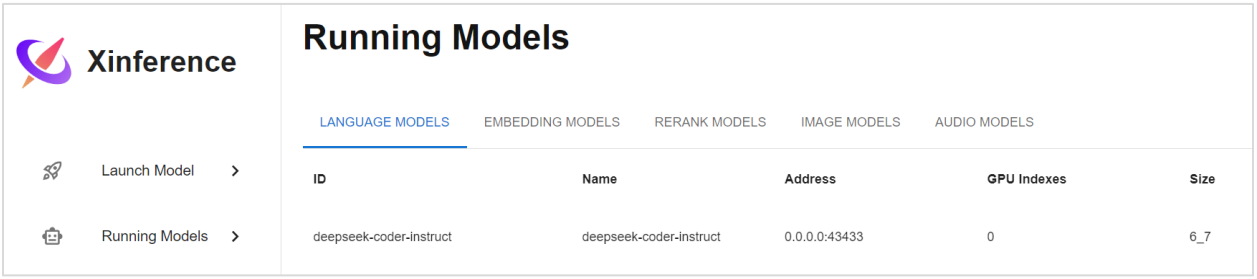
Table 3. Shell/Terminal Commands

Once step 1 and 2 are completed, access the exported URL to visit XINFERENCE. In XINFERENCE, there are many open-source large models available to run. We can directly search for models of interests, or view models by categories. We chose to deploy a large model called DeepSeek Coder Instruct (Display 5). This model is specialized in various programming languages, including SAS, R, Java, Python, etc. It also provides different sizes, ranging from 1B to 33B, and supports both Chinese and English.



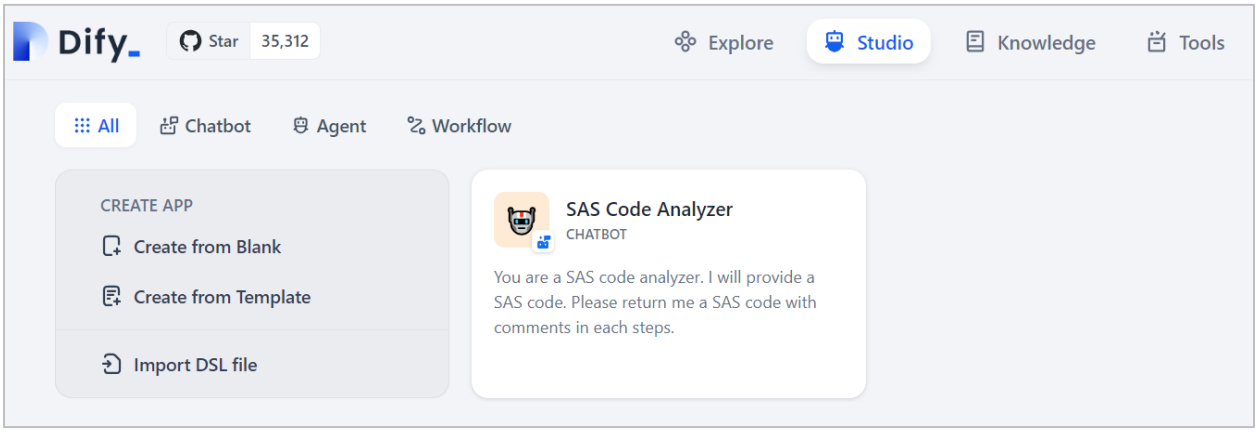
Display 5. DeepSeek Coder Instruct

Select the model, configure a few settings, and click on 'Launch'. During the model deployment process, Command Prompt automatically downloads needed files for the model in the background. Once the model is successfully deployed, it will appear under the 'Running Models' tab (Display 6).



Display 6. XINFERENCE

The final step is to compose Dify and start the Docker container. After entering the needed commands in the Command Prompt, access Dify through the exported URL and register for a Dify account (Display 7).



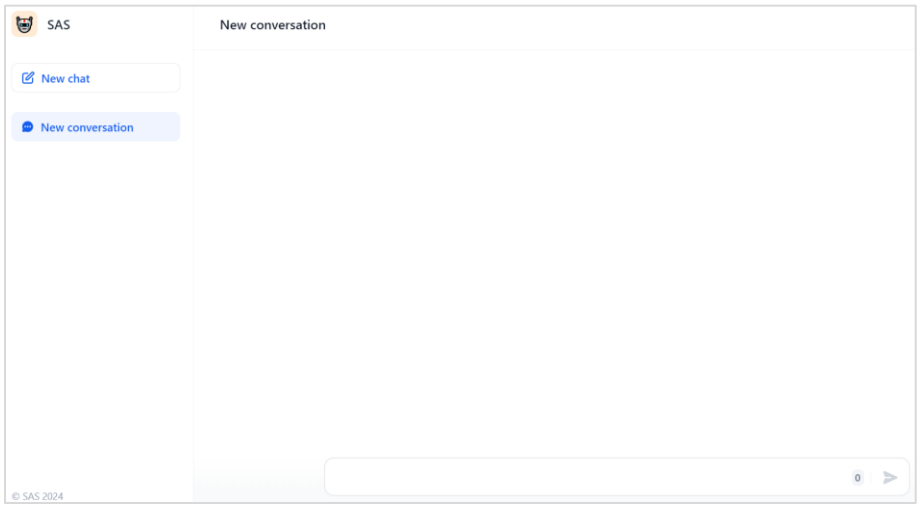
Display 7. Main interface for Dify

Next, navigate to Settings, under Model Provider tab, click on 'Add model' to add the running model within Xinference to Dify, which is the <deepseek-coder-instruct> model in this example. Select 'LLM' as the

model type, make sure Model Name matches the exact name in Xinference, provide Xinference Server URL and then save the settings.

Go back to the main interface, now create a simple Chatbot by clicking ‘Create from Blank’ and selecting ‘Chatbot’. For the Chatbot Orchestrate Method, select ‘Basic’. Enter a name for the application, such as ‘SAS Code Analyzer’, and provide a description as a preset. For example, in Display 7, with the description “You are a SAS code analyzer. I will provide a SAS code. Please return me a SAS code with comments in each step.” the Chatbot is informed in advance with questions like ‘Who am I?’, ‘Who are you?’, ‘What do I need to do?’, etc., which can be answered without the needing of providing additional prompts.

On the subsequent page, select ‘Publish’ – ‘Run App’. At this stage, Dify automatically generates a web-based Chatbot (Display 8) that any users can access via its URL.



Display 8. Main interface for the Chatbot generated by Dify

Try interacting with it using the prompts in the previous section (Table 4):

User: Could you tell me something about SAS?
SAS Code Analyzer: Sure, SAS (Statistical Analysis System) is a popular business analytics platform that is used for various statistical analyses, data management, and visualization tasks. It's widely used in academia, research institutions, and industries like banking, insurance, and healthcare. It supports various types of data such as structured and unstructured data, and provides various analytics tools like predictive analytics, data mining, and advanced analytics.
User: Could you help me understand my SAS code below? data adsl; set adam.adsl; if saffl=”Y”; run;
SAS Code Analyzer: Sure, I'd be happy to help you understand your SAS code. Here's a breakdown of your code: Sas data adsl; set adam.adsl;

if saffl=”Y”; run; data adsl;: This line is creating a new SAS data set named adsl. set adam.adsl;: This line is telling SAS to read data from the adam.adsl data set in the adam library. The set statement is used to specify the input data set. if saffl=”Y”;: This line is a conditional statement that filters the data. It only includes observations where the saffl variable equals "Y". The if statement is used to filter observations based on a condition. run;: This line tells SAS to execute the code. In this case, it's telling SAS to create a new data set adsl from the adam.adsl data set, but only including observations where saffl equals "Y". So, in summary, your code is creating a new data set adsl that includes only the observations from adam.adsl where the saffl variable is "Y".

Table 4. Interaction with DeepSeek Coder Instruct

Apparently, DeepSeek Coder Instruct provides more accurate responses to SAS-related questions. It precisely describes the function of the <if> statement in the code. More importantly, due to deployment on a server with powerful hardware resources, its response time is significantly reduced. Analyzing the SAS statement and providing a full response takes less than 3 seconds. With such accuracy and efficiency, it can be used to offer more assistance in SAS code. At this point, we have successfully deployed the LLM on the server and made it available online for multiple users to use simultaneously.

MORE INTERACTIONS

- Explaining functions (Appendix 1)
- Adding comments (Appendix 2)
- Checking syntax errors (Appendix 3)
- Understanding/Writing macros (Appendix 4)

CONCLUSION

The rapid development of AI technology has brought huge changes to various industries. How to effectively use AI to improve the efficiency of SAS programming becomes a more important topic. We are already very familiar with some advanced AI technologies, such as ChatGPT, which demonstrate excellent accuracy and intelligence. In this article, we do not show the interaction results with these top-tier AIs. Instead, we demonstrate how to deploy Large Language Models on servers so that companies can utilize them for internal employees. Of course, all AI tools are not always correct. The DeepSeek Coder Instruct model shown in this article also has misunderstandings when responding to certain prompts. Nevertheless, the performance of such open-source models is already good enough to provide help to SAS programmers. With the ongoing development of AI technology, the benefits it can bring to us are undeniable.

REFERENCES

ModelScope. “bingal/llamafile-models” Accessed June 28, 2024. https://modelscope.cn/models/bingal/llamafile-models/summary.

Deepseek. “deepseek-ai/DeepSeek-Coder” Accessed June 28, 2024. https://github.com/deepseek-ai/DeepSeek-Coder.

Mozilla's Innovation Group. "Introducing llamafile" Accessed June 28, 2024. https://hacks.mozilla.org/2023/11/introducing-llamafile/.

Featurize. "About Featurize" Accessed June 28, 2024. https://docs.featurize.cn/docs/about.

Dify. "Create Application" Accessed June 28, 2024. https://docs.dify.ai/guides/application-orchestrate/creating-an-application.

Xinference. "Getting Started" Accessed June 28, 2024. https://github.com/xorbitsai/inference.

ACKNOWLEDGMENTS

I would like to thank my manager Junkai Zhao for his technical support, patient guidance, and contributions to this paper.

CONTACT INFORMATION

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

Guangyong Li
Everest Clinical Research
guangyong.li@ecrscorp.com

APPENDIX 1

User: Can you help me understand the function index() and its usage?

SAS Code Analyzer: Sure, the index() function in SAS is used to find the position of the first occurrence of a substring within a string. The function takes two parameters:

string: This is the string in which you want to search for the substring.

substr: This is the substring you want to find.

The function returns the position of the first occurrence of the substring within the string. The position is counted from 1. If the substring is not found, the function returns 0.

Here's an example:

```
Sas
data test;
    string = 'Hello, SAS Programming';
    substr = 'SAS';
    pos = index(string, substr);
    put string= substr= pos=;
run;
```

In this example, the string 'Hello, SAS Programming' contains the substring 'SAS' at position 8. Therefore, the variable pos will be 8.

APPENDIX 2

User: Please help me add comments to my SAS code:

```
data adsl;
length AGEGR1 $20;
set adam.adsl;

if age ^=. then do;
        if age < 20 then do;
                agegr1='<20';
                agegr1n=1;
        end;
        else if 20<= age < 40 then do;
                agegr1='>=20 to < 40';
                agegr1n=2;
        end;
        else if 40 <=age then do;
                agegr1=">=40";
                agegr1n=3;
        end;
end;
run;
```

SAS Code Analyzer: Sure, here's your code with added comments:

Sas

```
/* Begin the data step to create a new dataset named 'adsl' */
data adsl;

/* Define the length of variable 'AGEGR1' to be 20 characters */
length AGEGR1 $20;

/* Set the input dataset as 'adam.adsl' */
set adam.adsl;

/* If the age variable is not missing */
if age ^=. then do;

   /* If age is less than 20 */
   if age < 20 then do;
      /* Set 'agegr1' to '<20' and 'agegr1n' to 1 */
      agegr1='<20';
      agegr1n=1;
   end;

   /* If age is between 20 and 39 */
   else if 20<= age < 40 then do;
      /* Set 'agegr1' to '>=20 to < 40' and 'agegr1n' to 2 */
```

```
        agegr1='>=20 to < 40';
        agegr1n=2;
    end;

    /* If age is 40 or above */
    else if 40 <=age then do;
        /* Set 'agegr1' to '>=40' and 'agegr1n' to 3 */
        agegr1=">=40";
        agegr1n=3;
    end;
end;

/* End the data step */
run;
```

This adds comments to your code to make it more readable and understandable. It explains what each part of the code does.

APPENDIX 3

User: I'll provide you with a code, please identify syntax errors for me.

```
data advs;
set adam.advs;
if paramcd in ('BMI' 'WEIGHTKG') ablfl='Y';
if saffl="Y" and anl01fl="Y"
run;
```

SAS Code Analyzer: Your code has two syntax errors:

1. The if statement is not properly closed. It's missing a then statement.
2. The if statement after the if statement is missing a then or do statement.

Here's the corrected code:

```
Sas
1data advs;
2           set adam.advs;
3           if paramcd in ('BMI' 'WEIGHTKG') and ablfl='Y';
4
5           if saffl="Y" then do;
6                       if anl01fl="Y" then output;
7           end;
8run;
```

In this corrected code, I've added a then after the if statement to close it, and I've added a do after the second if statement to start a block of code. The output statement is used to write the observation to the output data set if the conditions are met.

APPENDIX 4

User: I'll provide you with a SAS macro, please help me understand it.

```
%macro desc(indat=, var=, label=, dec=, ord=);
proc sort data=&indat;
by trtn;
run;
proc means data=&indat;
var &var;
by trtn;
output out=desc_&var n=n mean=mean stddev=std median=median min=min max=max;
run;
data desc_&var.1;
set desc_&var.;
n_=strip(put(n, 8.0));
mean_=strip(put(mean, 8.%eval(&dec+1)));
std_=strip(put(std, 8.%eval(&dec+2)));
median_=strip(put(median, 8.%eval(&dec+1)));
min_=strip(put(min, 8.&dec.));
max_=strip(put(max, 8.&dec.));
run;
proc sort data=desc_&var.1;
by trtn;
run;
proc transpose data=desc_&var.1 out=desc_&var.2 prefix=col;
var n_ mean_ std_ median_ min_ max_;
id trtn;
run;
data desc_&var.3 (drop=name);
length var label col0 $50 col1-col6 $200;
set desc_&var.2;
col0=input(name, $stat.);
var=upcase("&var");
label="&label";
ord=&ord;
```

```
ord2=input(name, stat.);
run;
data all;
set all desc_&var.3(keep=var label col: ord:);
run;
%mend desc;
```

SAS Code Analyzer: This SAS macro named desc is designed to compute various statistical measures for each treatment (trtn) in a dataset (indat) and output the results in a specified format. Here's a breakdown of what it does:

Sorting the input dataset: The proc sort procedure is used to sort the input dataset by trtn.

Computing statistical measures: The proc means procedure is used to compute several statistical measures (n, mean, stddev, median, min, max) for the variable specified by var. The results are written to a new dataset named desc_&var.

Formatting the output: The macro creates a new dataset desc_&var.1 by formatting the statistical measures. It uses the strip and put functions to format the numerical values to the desired number of decimal places.

Transposing the output: The proc transpose procedure is used to transpose the dataset desc_&var.1, creating a new dataset desc_&var.2. The var statement specifies the variables to be transposed, and the id statement specifies the variable to be used as the new column names.

Creating final output: The macro creates a final dataset desc_&var.3. It creates new variables var, label, and ord, and assigns them the values of the macro variables var, label, and ord.

Updating the main dataset: The final dataset desc_&var.3 is appended to the main dataset all. If all does not exist, it is initialized as the final dataset.

The macro uses several SAS functions and procedures, including proc sort, proc means, proc transpose, and several functions for formatting and string manipulation. The macro also uses several SAS data step statements, including set, length, data, and run.

User: Can you help me write a simple SAS macro? The purpose of this macro is to calculate frequencies and percentages from a given dataset for a specific variable.

SAS Code Analyzer: Sure, I can help you write a simple SAS macro for that. Here's a basic example:

Sas

```
%macro freq_percent(dataset, var);

   ods output CrossTabFreqs=freqs;
   proc freq data=&dataset;
      tables &var*&var / list;
   run;
```

```
    data freqs;
        set freqs;
        percent = colpctn;
        keep &var count percent;
    run;


%mend;
```

You can use this macro by calling %freq_percent(dataset, var), where dataset is the name of your dataset and var is the variable you want to calculate frequencies and percentages for.

This macro uses the ods output statement to create a new SAS dataset named freqs that contains the frequency counts. It then modifies this dataset to include a new variable percent which contains the percentage of the total for each category.

Please note that this is a very basic macro and may not cover all your needs. If you need more complex functionality, you may need to modify or extend it.