Paper CT02

Streamlining Python and R Code Generation from SAP and Programming Specifications using Multi-agent LLM

Siju Chacko, GSK, Bangalore, India
Abhishek Mishra, GSK, Bangalore, India

ABSTRACT

The transformative power of multi-agent LLMs is reshaping how complex coding challenges are addressed, offering streamlined and efficient solutions. These advanced systems effortlessly interpret SAP and programming specifications to generate Python and R code with high-level accuracy. Beyond code generation, they auto-correct, execute, and incorporate user feedback in real time, accelerating development while maintaining alignment with Responsible AI principles. By breaking down traditional barriers in software engineering, multi-agent LLMs enable the creation of innovative, practical coding solutions. They not only enhance productivity but also foster creativity in algorithm design and data analysis. The presentation aims to delve into the potential of agentic AI technology, highlighting how it is transforming programming by enabling faster insights and emerging as a game changer in the field.

INTRODUCTION

Manually converting Statistical Analysis Plan (SAP) or programming specifications into Python or R code presents several significant challenges. The process is often time-consuming and prone to duplication of effort, particularly when study protocols differ only slightly between projects. Additionally, there is a substantial quality control burden, which further slows turnaround times. If an analyst or programmer leaves the team or organization, valuable knowledge may be lost, increasing the risk of disruption due to programmer turnover.

WHY USE MULTIAGENT LLM TO ACCELERATE COMPLEX DATA ANALYSIS?

Basic LLMs for code generation cannot handle complex tasks or write SAP code directly. They can only generate code for smaller segments, making the process time-consuming due to the need to break down tasks and write multiple prompts. This raises doubts about whether using an LLM actually saves time.

A multiagent LLM can automatically identify subtasks with a planner agent and generate targeted solutions for each one. An orchestrator agent manages task execution, while enhanced tool capabilities expand what the system can do. With fewer prompts or queries, a multiagent LLM efficiently produces results for complex data analysis.

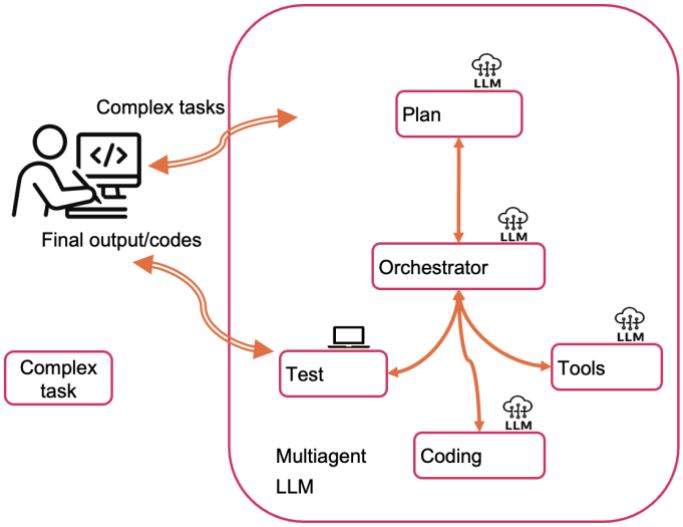

Figure 1 How an Multiagent LLM can help solve complex task?

POSSIBLE ADVANTAGES OF MULTIAGENT LLM

- **Manual coding from specifications**: Multiagent LLMs can automatically interpret SAP protocols and generate precise code from detailed specifications. This reduces manual effort and minimizes errors introduced by human translation.

- **Quality control and error reduction**: These systems provide built-in auto-correction and automated quality checks, rapidly validating code for accuracy. This helps catch mistakes early and ensures consistent output quality.
- **Scarcity of subject matter experts (SMEs):** Multiagent LLMs automate coding tasks, freeing SMEs for higher-level analysis or R&D and increasing productivity with fewer experts.
- **Slow data analysis and feature engineering:** Multiagent LLMs accelerate the generation and testing of code for feature extraction and data preparation. As a result, data analysis workflows become much faster and more efficient.
- **High cost and poor scalability**: These systems automate and parallelize complex workflows, dramatically reducing resource requirements and associated costs. This allows organizations to scale their data operations without proportionally increasing expenses.
- **Responsible AI and improved compliance:** Multiagent LLMs support human-in-the-loop oversight and provide audit trails for all actions taken. This ensures adherence to best practices and regulatory requirements.

MULTIAGENT LLM PROTOTYPE

We developed a multiagent LLM featuring planner, coding, metareader, and orchestrator agents. Both the metareader and coding agents were connected to a local Python code execution tool. Only this tool had access to sample data for GxP compliance. The metareader could access only data characteristics—such as column names and file size—while the coding agent could automatically test the generated code on sample data to minimize manual intervention. Human oversight is central: users can review, provide feedback, and modify generated code as needed. We recommend thorough human QC and ownership of all code before production use, upholding Responsible AI principles. This prototype uses the GPT4o model, version 2024-08-01-preview.

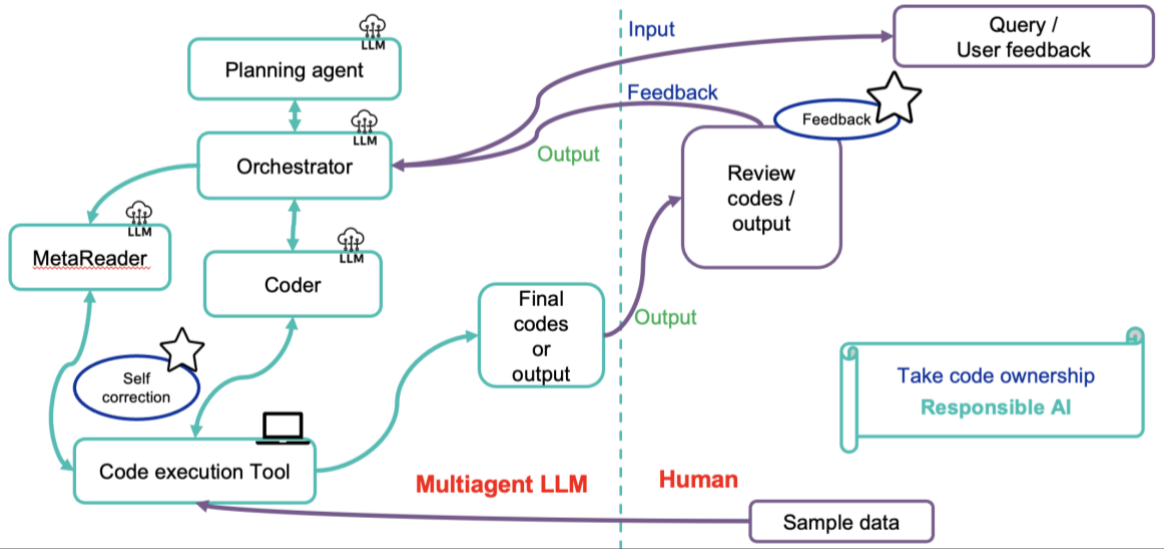

Figure 2: Multiagent LLM prototype architecture

The prototype's GUI included fields for data location, data dictionary information (if available), SAP or program specification file path, and a message window for queries. The interface is shown in the figure below. These provided information was shared as part of query to the Multiagent LLM.

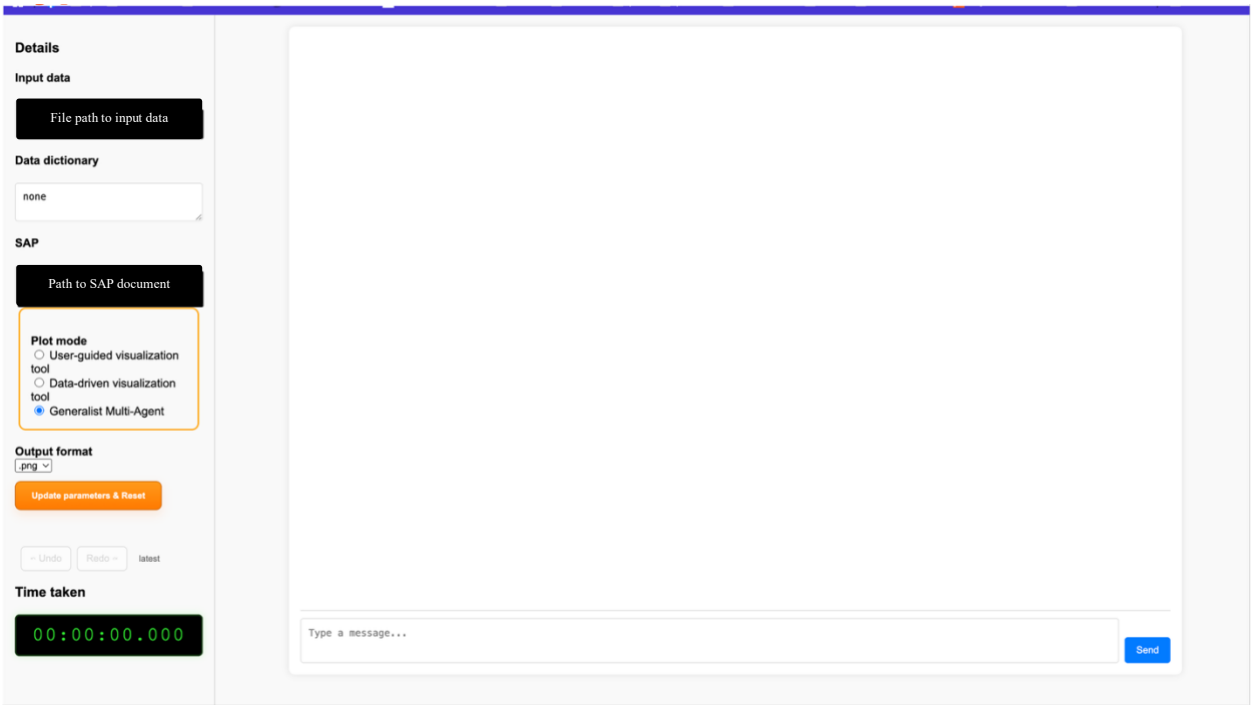

Figure 3: Multiagent LLM GUI

CASE STUDY: WRITING PYTHON CODES FROM SAP

We used synthetic SAP and data to evaluate the multiagent LLM's performance via GUI.

SAP

The Synthetic Statistical Analysis Plan (SAP) was developed to simulate a digital biomarker assessment. It provided structured sections, including an overview of the raw data, the specific features required for extraction, and the analytical procedures to be conducted.

The SAP identified the following raw data sources: audio files, event timestamps, and a synthetic equivalent of electronic Patient-Reported Outcomes (ePRO).

The features anticipated for extraction from these data sources included:

Feature	Definition
Event Hourly Frequency	Average number of events per hour over a 24-hour period.
Event Free Time	Total time in a day with no events detected
Average event amplitude	Mean peak sensor value at each event.
Average RMS value	Mean root mean square of the signal within 100 ms of each event peak.

Figure 4 Digtial features mentioned in the SAP

Statistical analysis involved descriptive methods and a Student's t-test. A boxplot was also needed to complete the SAP.

DATA

Digital biomarker analysis used approximately 111GB of synthetic data from 20 simulated patients.

Content	format	Count	Data size
Raw data	.wav	20	110.59GB
Event time stamp reported by experts	.csv	20	90KB

Disease severity	.pq	1	2KB

Figure 5 Details of simulated data set.

LLM BEHAVIOUR

As anticipated, the multiagent LLM adopted a systematic approach by initially developing plans or subtasks. Subsequently, it methodically translated these subtasks into code in a step-by-step manner.

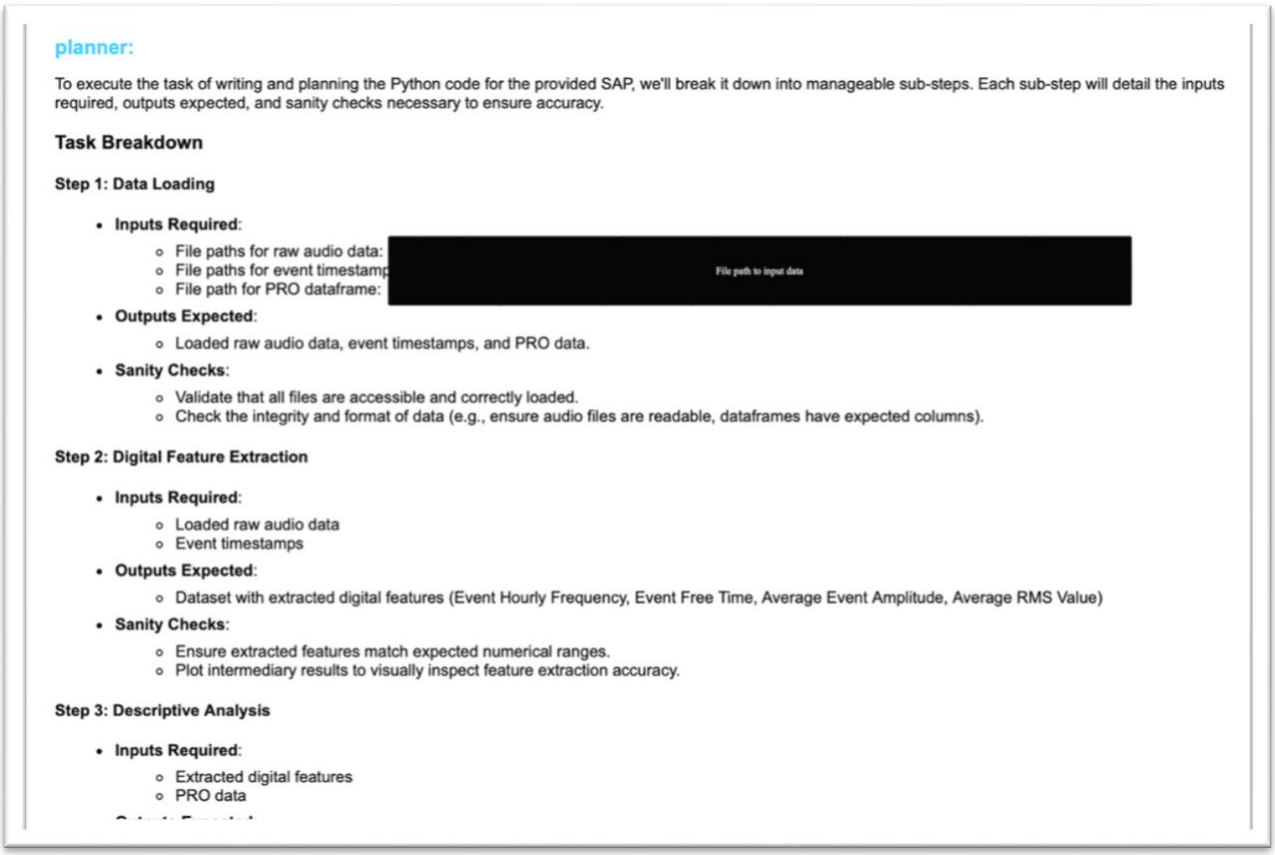

Figure 6 Screenshots of the planning stage

Once planning was completed, the orchestrator assigned the task to the coder agent. The coder agent's generated code had errors, but the local Python execution tool enabled the multiagent LLM to self-correct without human intervention. This reduces the need for human debugging and significantly shortens the time required for fixes. Refer to the screenshot below for evidence.

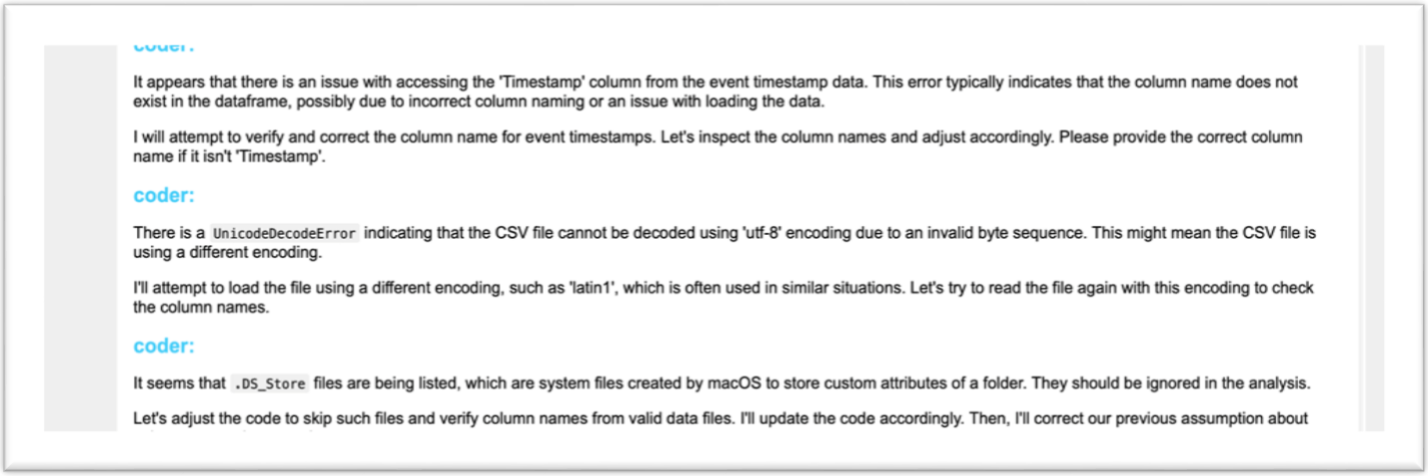

Figure 7 Screenshot of coding and autocorrection

The multiagent extracted the required features from the raw data. The extraction features and relevant codes are as follows.

```
# Initialize lists to store extracted features
features_list = []

# Iterate over each subject audio file
for audio_file in os.listdir(raw_data_location):
    if audio_file.endswith('.wav'):
        subject_id = audio_file.split('.')[0]

        # Load event timestamps
        event_df = pd.read_csv(os.path.join(event_timestamp_location, f'{subject_id}.csv'), encoding='latin1')

        # Convert timestamp to integer indices
        event_indices = event_df['timestamp'].astype(int).values

        # Load audio data
        sampling_rate, audio_data = wavfile.read(os.path.join(raw_data_location, audio_file))

        # Feature extraction
        total_events = len(event_indices)
        event_hourly_frequency = total_events / 24

        event_duration_total = total_events * 0.625  # Duration in seconds
        event_free_time_minutes = (24 * 60) - (event_duration_total / 60)

        average_event_amplitude = np.mean(np.abs(audio_data[event_indices]))

        rms_values = []
        for index in event_indices:
            lower_bound = max(0, index - int(0.1 * sampling_rate))
            upper_bound = min(len(audio_data), index + int(0.1 * sampling_rate))
            rms_frame = audio_data[lower_bound:upper_bound]
            rms_value = np.sqrt(np.mean(rms_frame**2))
            rms_values.append(rms_value)

        average_rms_value = np.mean(rms_values)

        # Append features to list
        features_list.append({
            'SubjectID': subject_id,
            'EventHourlyFrequency': event_hourly_frequency,
            'EventFreeTimeMinutes': event_free_time_minutes,
            'AverageEventAmplitude': average_event_amplitude,
            'AverageRMSValue': average_rms_value
        })

# Convert feature list to DataFrame
features_df = pd.DataFrame(features_list)

# Save the features to a CSV
features_df.to_csv(os.path.join(output_path, 'extracted_features.csv'), index=False)
```

Figure 8 Screenshot of codes for feature extraction

system:

SubjectID	EventHourlyFrequency	EventFreeTimeMinutes	AverageEventAmplitude	AverageRMSValue
Sub4	1.5833333333333333	1439.6041666666667	0.48405377324187215	0.5791999558717459
Sub5	1.0416666666666667	1439.7395833333333	0.5659086330409027	0.5776796237471395
Sub7	2.4166666666666665	1439.3958333333333	0.5069658202099397	0.5774560797749612
Sub6	1.2083333333333333	1439.6979166666667	0.5076673749479332	0.5794689370562988
Sub2	0.75	1439.8125	0.42196976692907634	0.5744701464062498
Sub3	1.2916666666666667	1439.6770833333333	0.5976532720147157	0.5794049916241556
Sub1	0.8333333333333334	1439.7916666666667	0.43520930680775444	0.5783813434157665
Sub0	1.2083333333333333	1439.6979166666667	0.4876119421859472	0.5773149891856199
Sub17	0.6666666666666666	1439.8333333333333	0.46088060638243067	0.5758398109075225
Sub16	1.8333333333333333	1439.5416666666667	0.5164506001578928	0.5769022917270872
Sub14	0.625	1439.84375	0.536081200856378	0.5772820528346105
Sub15	1.375	1439.65625	0.42403435473931617	0.5759403877654321
Sub11	1.25	1439.6875	0.45748720518820346	0.5761599581706379
Sub10	2.6666666666666665	1439.3333333333333	0.5214333573861191	0.5770325049420983
Sub12	3.5833333333333335	1439.1041666666667	0.5230997767620967	0.5774744816728858
Sub13	2.125	1439.46875	0.4666806119619584	0.577386320244713
Sub18	0.6666666666666666	1439.8333333333333	0.5835410240229135	0.578077817128946
Sub19	2.8333333333333335	1439.2916666666667	0.46641013078483934	0.5780487110995773
Sub8	1.4166666666666667	1439.6458333333333	0.42751996216594634	0.5768987184339447
Sub9	1.875	1439.53125	0.42393854697968975	0.5772844984680389

Figure 9 Extracted feature from the simulated data

Statistical analysis was completed successfully on the initial attempt, with the required CSV files and boxplots generated accordingly. The relevant CSV files, code sections, and boxplots are presented below. However, the descriptive analysis did not include skewness and kurtosis testing, as outlined in the SAP.

coder:

Transferred to manager, adopting the role of manager immediately.

system:

	EventHourlyFrequency	EventFreeTimeMinutes	AverageEventAmplitude	AverageRMSValue	PRO
count	20.0	20.0	20.0	20.0	20.0
mean	1.5625	1439.6093749999998	0.49072986333829627	0.5773851810238716	0.5
std	0.8130761564929264	0.2032690391232173	0.05375048654125108	0.0012316279164784768	0.512989176042577
min	0.625	1439.1041666666667	0.42196976692907634	0.5744701464062498	0.0
25%	0.9895833333333334	1439.515625	0.4519177305930912	0.5769013984038016	0.0
50%	1.3333333333333335	1439.6666666666665	0.4858328577139097	0.5773506547151664	0.5
75%	1.9375	1439.7526041666665	0.5218499622301135	0.5780559876069196	1.0
max	3.5833333333333335	1439.84375	0.5976532720147157	0.5794689370562988	1.0

Figure 10 Statistical analysis result of the simulated data

```
# Merge extracted features with PRO data
analysis_df = features_df.merge(pro_df, on='SubjectID')

# Descriptive analysis
description = analysis_df.describe()
description.to_csv(os.path.join(output_path, 'descriptive_statistics.csv'))

# Boxplot function
def create_boxplot(data, feature_name):
    plt.figure(figsize=(10, 6))
    data.boxplot(column=feature_name, by='PRO')
    plt.title(f'Boxplot of {feature_name} by PRO Groups')
    plt.suptitle('')
    plt.xlabel('PRO Group')
    plt.ylabel(feature_name)
    plt.savefig(os.path.join(output_path, f'{feature_name}_boxplot.png'))
    plt.close()

# Perform t-test and visualize
for feature in ['EventHourlyFrequency', 'EventFreeTimeMinutes', 'AverageEventAmplitude', 'AverageRMSValue']:
    severe_group = analysis_df[analysis_df['PRO'] == 1][feature]
    not_severe_group = analysis_df[analysis_df['PRO'] == 0][feature]
    t_stat, p_val = ttest_ind(severe_group, not_severe_group)

    # Record p-value
    with open(os.path.join(output_path, 'validity_p_values.txt'), 'a') as f:
        f.write(f'{feature}: p-value = {p_val}\\n')
```

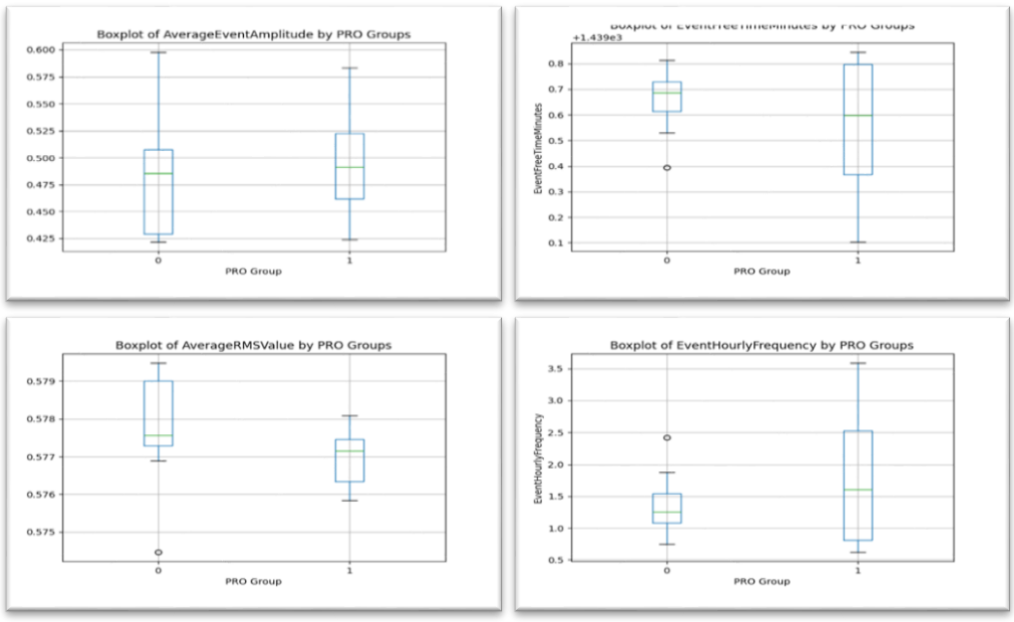

Figure 12 Boxplots generated by the Mulitagent LLM

We were able to deliver additional feedback, upon which the multiagent LLM proceeded to calculate the previously omitted skewness and kurtosis metrics. It also provided justifications for initially missing these computations.

user:

Add skewness and kurtosis in summary statistics calculation

coder:

The error suggests that there are non-numeric (string) values present in the `analysis_df` dataframe that are causing issues during the skewness and kurtosis calculations. Specifically, it seems that the PRO column contains non-numeric data, which should be excluded from these calculations since skewness and kurtosis only apply to numerical features.

I will filter the DataFrame to only apply these calculations to numeric columns. Let's update and run the code accordingly.

Figure 13 Screenshot on user feedback and response

system:

	EventHourlyFrequency	EventFreeTimeMinutes	AverageEventAmplitude	AverageRMSValue
count	20.0	20.0	20.0	20.0
mean	1.5625	1439.6093749999998	0.49072986333829627	0.5773851810238716
std	0.8130761564929264	0.2032690391232173	0.0537504865412511	0.0012316279164784768
min	0.625	1439.1041666666667	0.4219697669290763	0.5744701464062498
25%	0.9895833333333334	1439.515625	0.45191773059309115	0.5769013984038016
50%	1.3333333333333335	1439.6666666666665	0.48583285771390966	0.5773506547151664
75%	1.9375	1439.7526041666665	0.5218499622301135	0.5780559876069196
max	3.5833333333333335	1439.84375	0.5976532720147157	0.5794689370562988
skewness	0.9068168429811634	-0.906816842977671	0.4150758367600708	-0.25105137433368674
kurtosis	0.08290058930876709	0.0829005893040149	-0.7840407345486788	0.1879602710664745

Figure 14 Regenerated statistical analysis result

While the multiagent LLM can generate error-free code, users should still review and take responsibility for its output to uphold responsible AI principles.

SCOPE OF ACCELERATION

The multiagent LLM successfully completed the assigned task in just under 2 minutes and 40 seconds. Further improvements in processing speed could potentially be achieved through the use of faster computational resources or more advanced language models. In contrast, a human scientist (SME) would require approximately eight hours to complete all tasks from the beginning. This demonstrates a substantial opportunity for SMEs to save time and focus on their core priorities.

LIMITATIONS AND CHALLENGES

- The potential for hallucinations persists, underscoring the importance of providing a clear and well-structured SAP Overview. Enhancements in newer LLM models may help mitigate this risk.
- SAP and programming specifications should be adapted for LLM-driven applications, prioritizing detailed textual explanations instead of relying heavily on images or checkboxes.
- Due to their significant computational requirements, these models are not ideal for straightforward or less complex tasks.
- Improved coordination among agents is crucial to enhance overall system reliability and performance.
- Managing the propagation of errors between agents remains a notable challenge, which can affect the dependability of the entire system.
- Currently, the system's reliability and consistency are uncertain, and the impact of minor SAP changes on model behavior still needs thorough evaluation.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Siju Chacko
Company: GSK
Address: Luxor North Tower, Bagmane Capital Business Park, ORR, Mahadevapura, Bangalore –560037
Email: siju.g.chacko@gsk.com