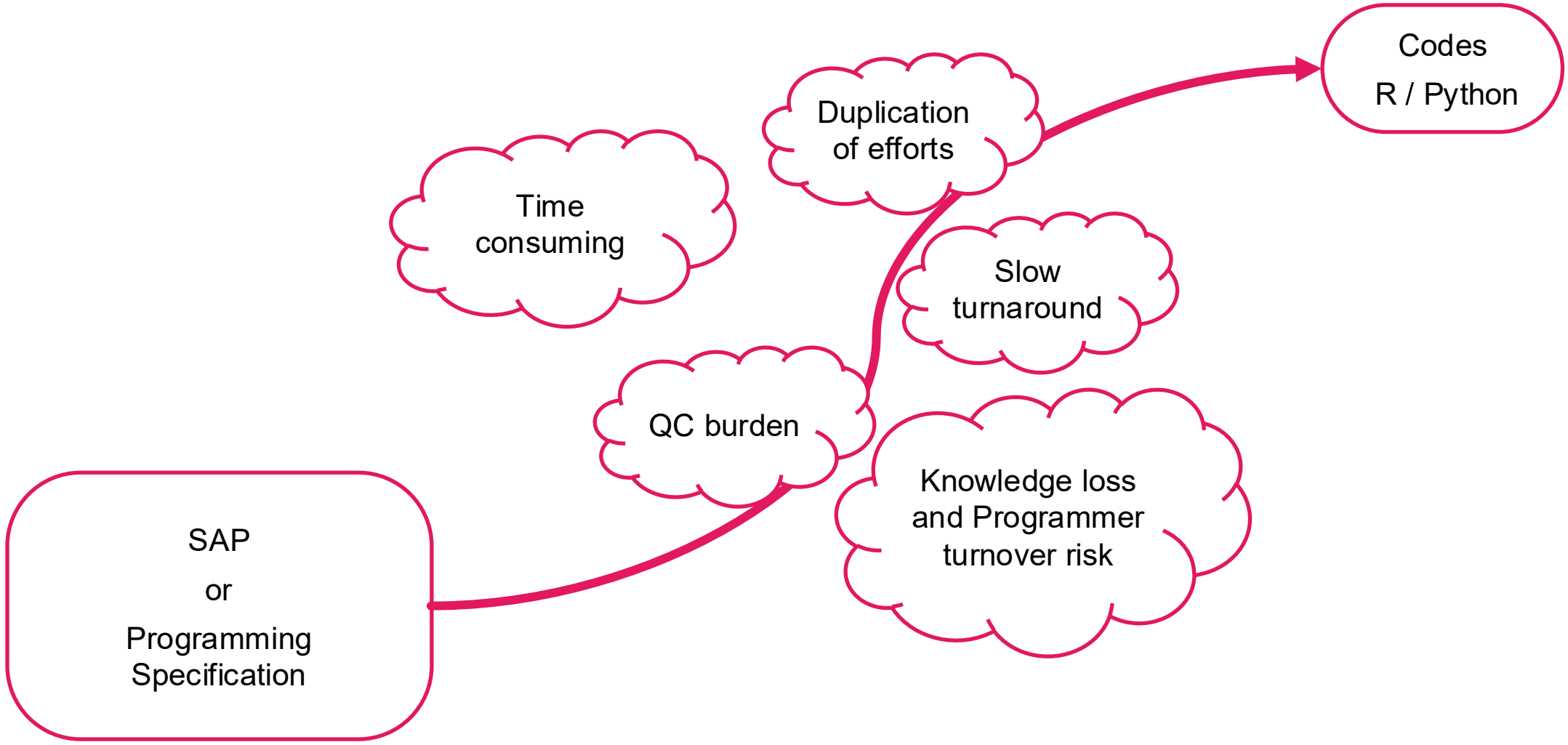
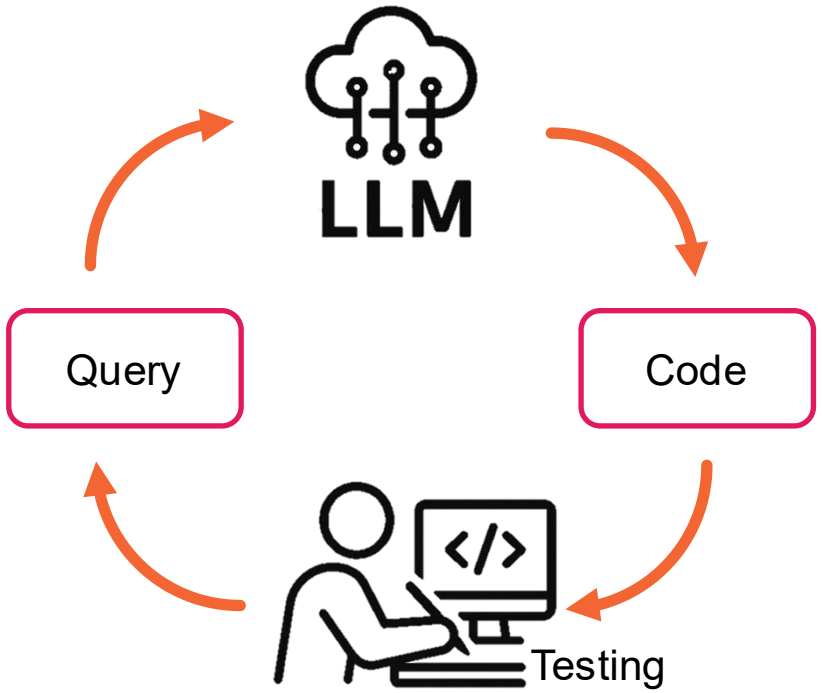
Streamlining Python and R Code Generation from SAP and Programming Specifications using Multi-agent LLM

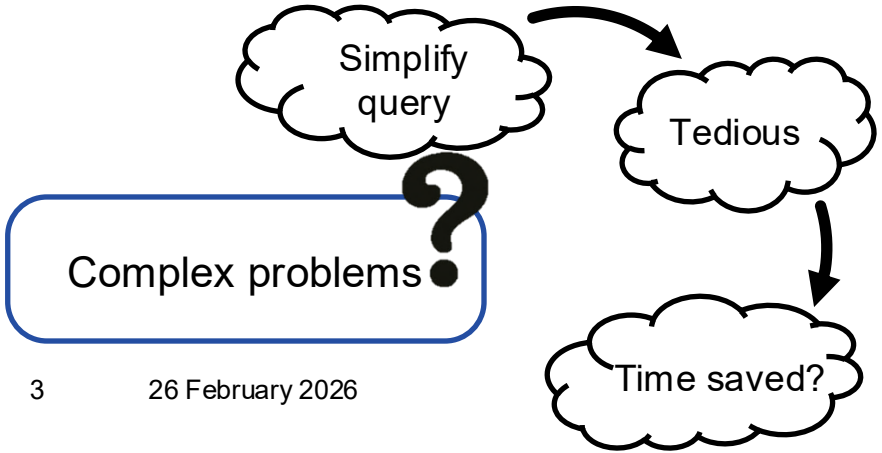
Dr Siju Chacko
Abhishek Mishra

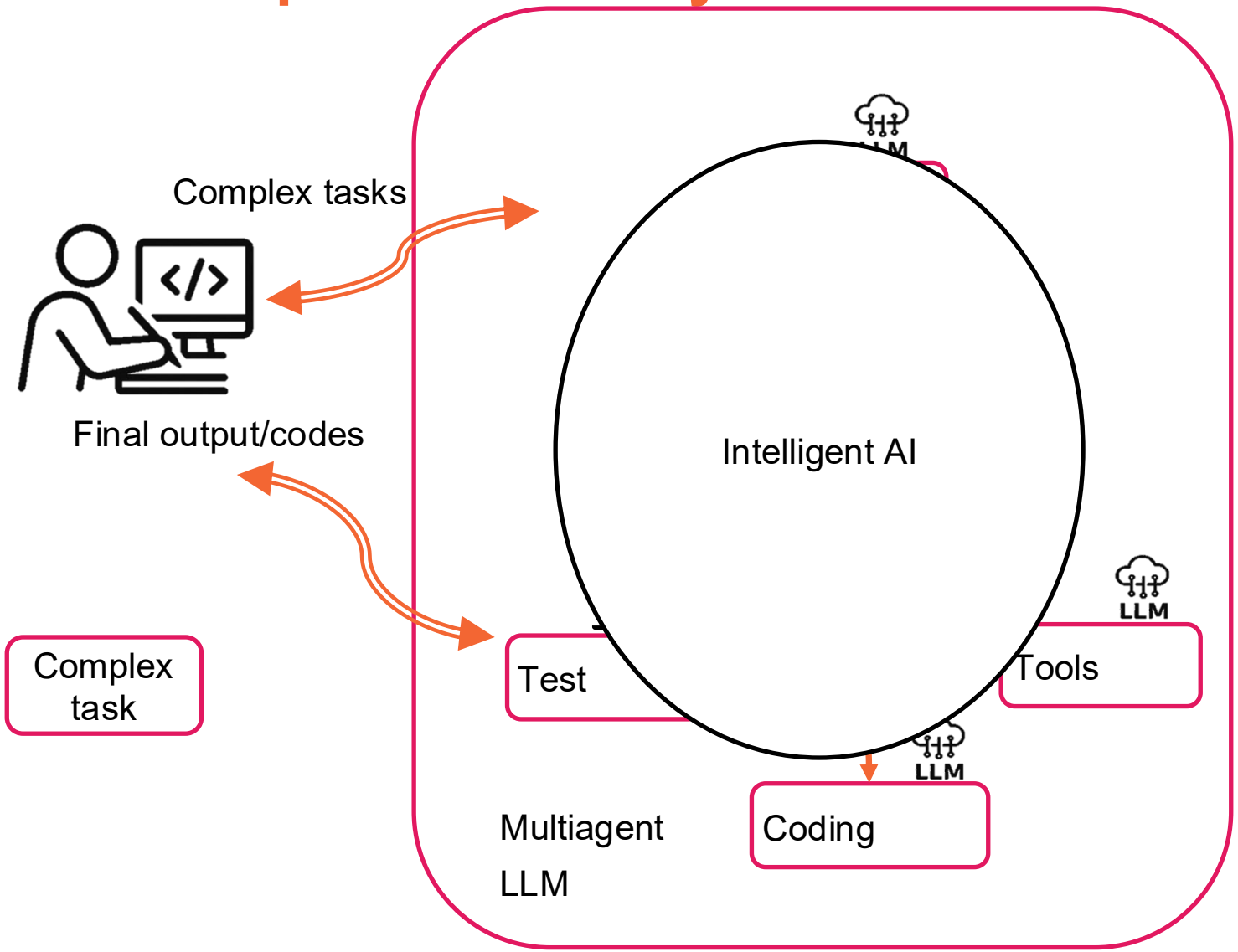
Challenges in Code Generation



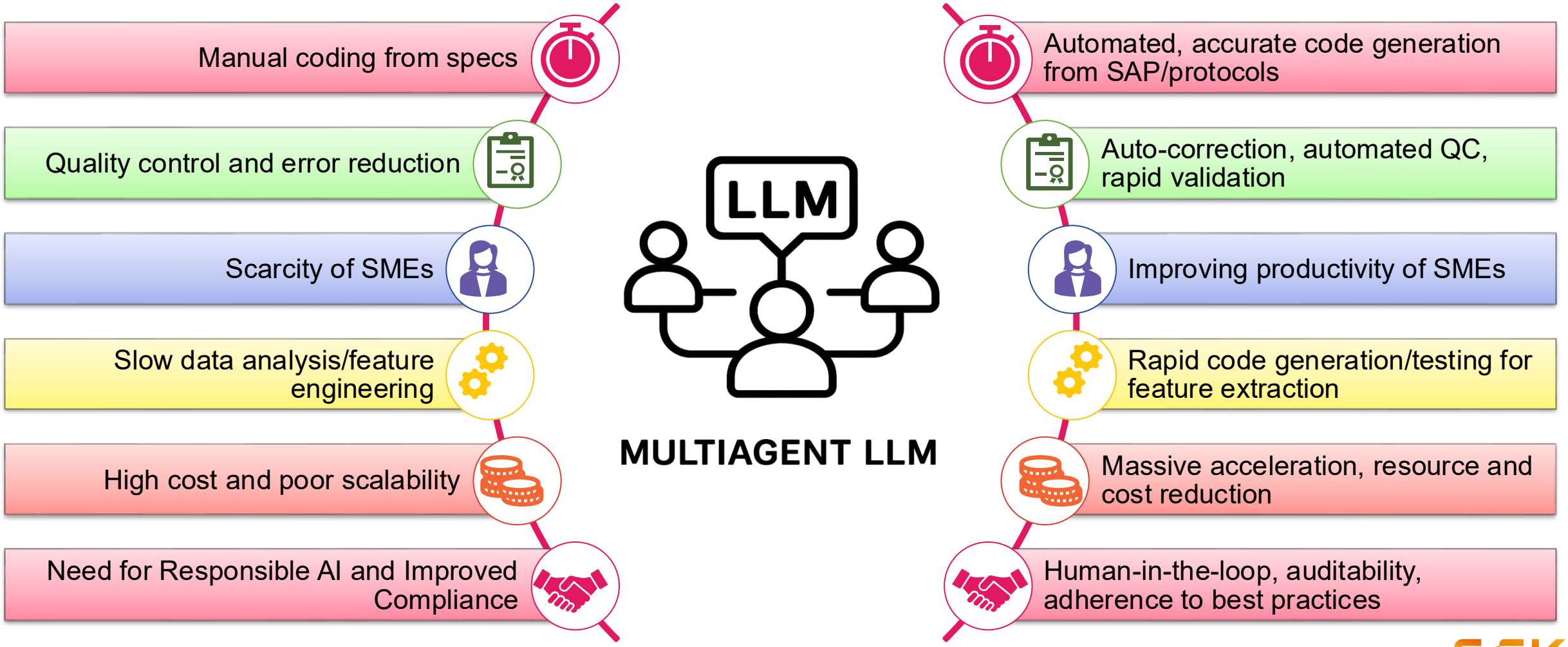
Why enhance Generative AI for complex data analysis?





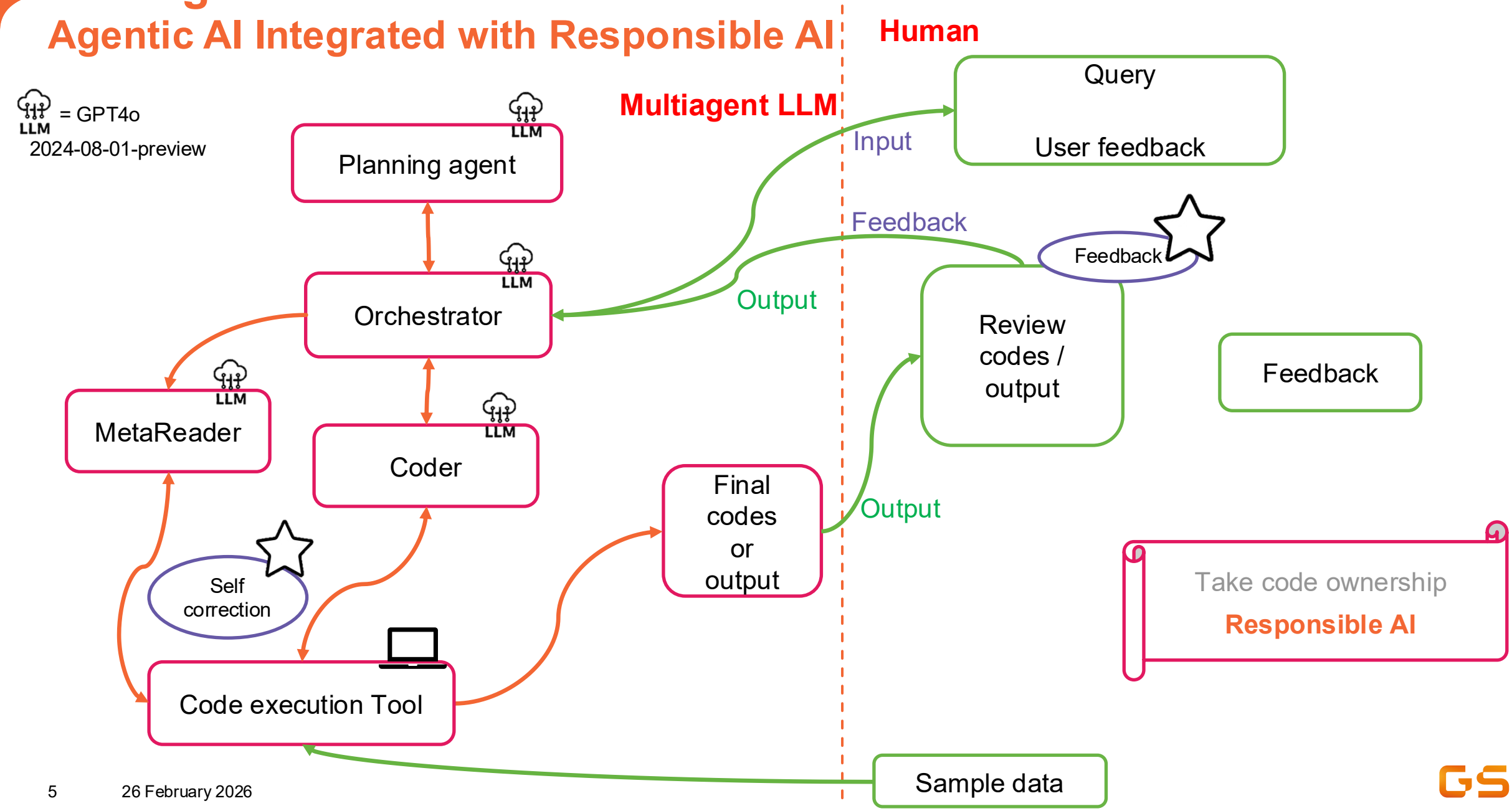


Possible Acceleration with Multiagent LLM

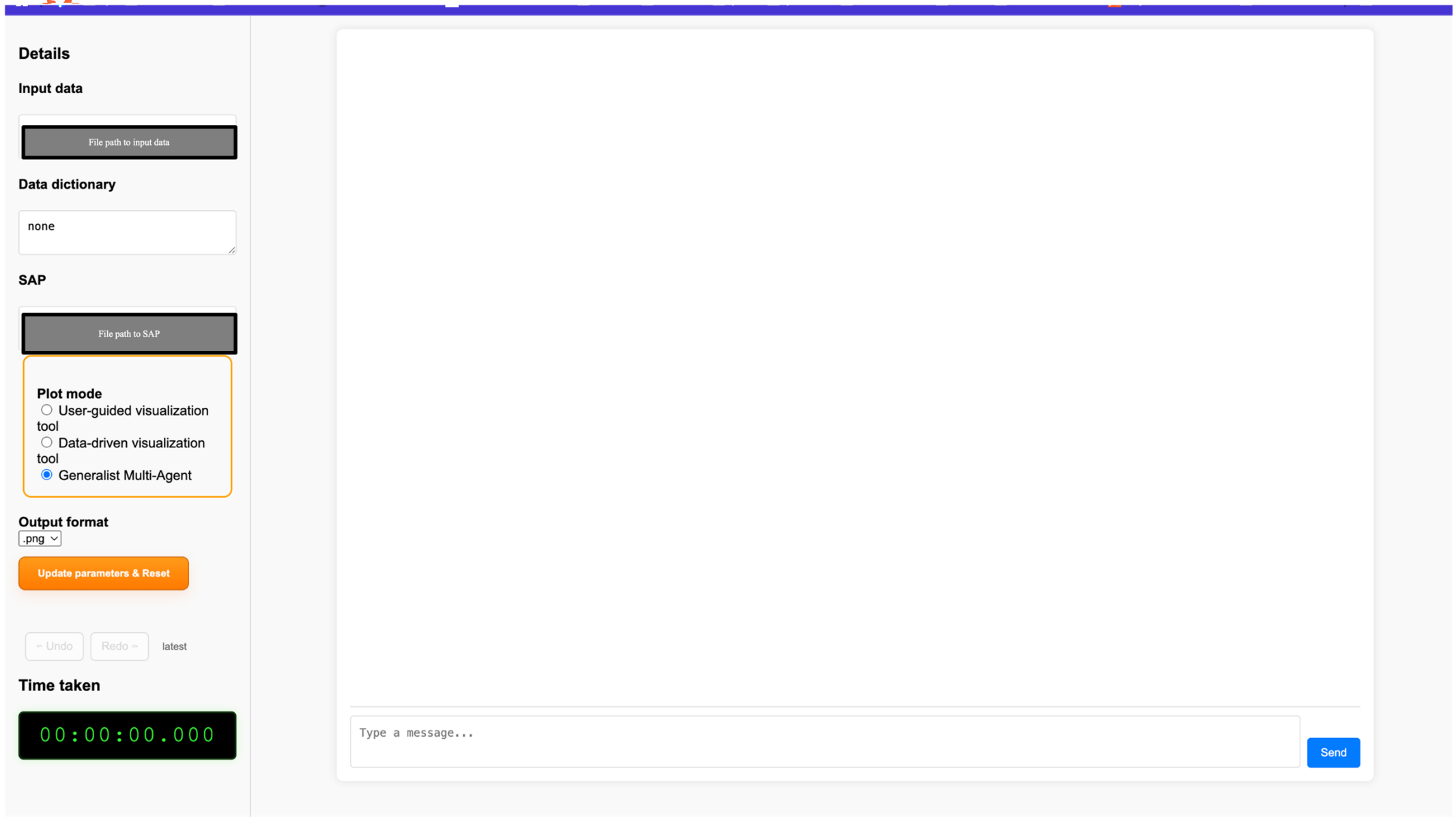


Multiagent LLM

Agentic AI Integrated with Responsible AI



Multiagent LLM - GUI



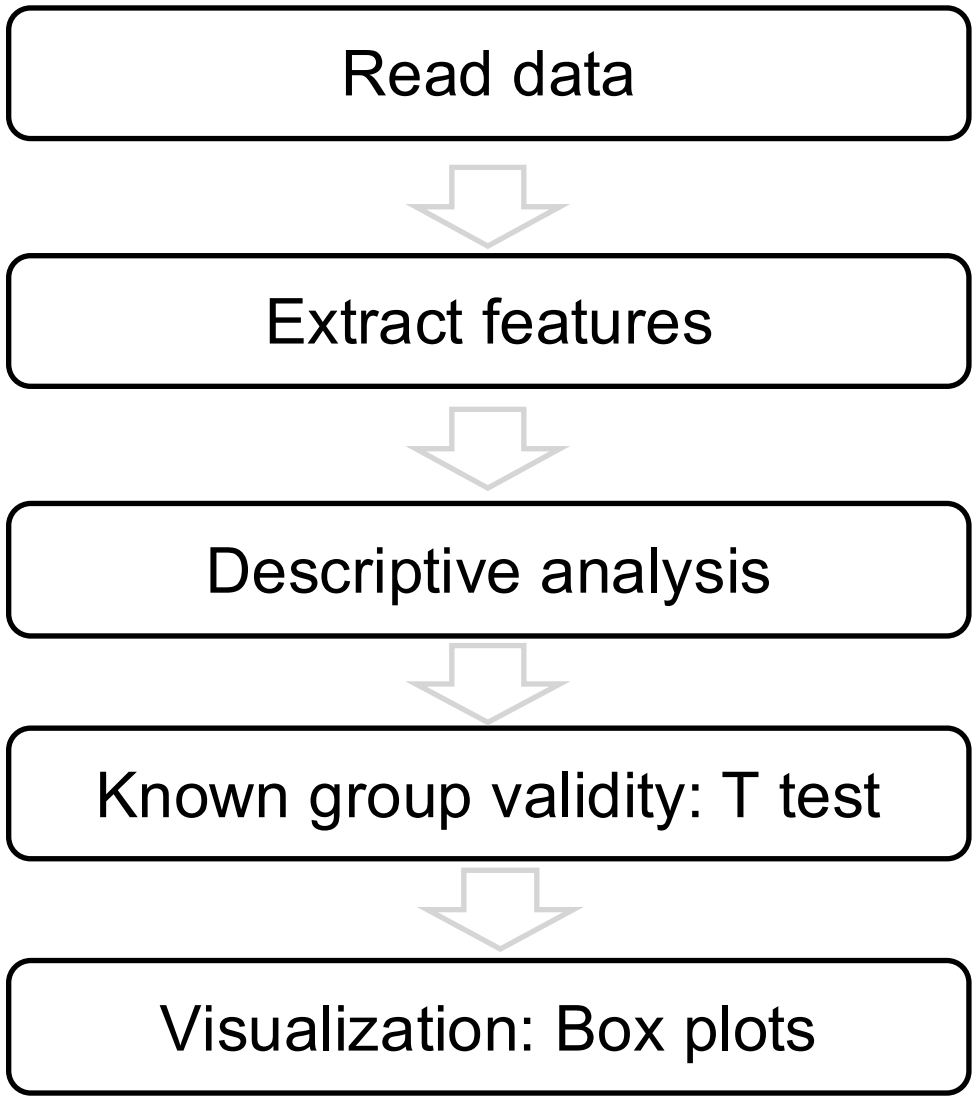
Case study

Understanding the SAP

Data

Content	format	Count	Data size
Raw data	.wav	20	110.59 GB
Event time stamp reported by experts	.csv	20	90KB
Disease severity	.pq	1	2KB

Task



Case study: Writing codes from SAP: Python

Planning stage

planner:

To execute the task of writing and planning the Python code for the provided SAP, we'll break it down into manageable sub-steps. Each sub-step will detail the inputs required, outputs expected, and sanity checks necessary to ensure accuracy.

Task Breakdown

Step 1: Data Loading

- **Inputs Required**:
  - File paths for raw audio data: [redacted]
  - File paths for event timestamp
  - File path for PRO dataframe:
- **Outputs Expected**:
  - Loaded raw audio data, event timestamps, and PRO
- **Sanity Checks**:
  - Validate that all files are accessible and correctly loa
  - Check the integrity and format of data (e.g., ensure a

Step 2: Digital Feature Extraction

- **Inputs Required**:
  - Loaded raw audio data
  - Event timestamps
- **Outputs Expected**:
  - Dataset with extracted digital features (Event Hourly
- **Sanity Checks**:
  - Ensure extracted features match expected numerica
  - Plot intermediary results to visually inspect feature e

Step 3: Descriptive Analysis

- **Inputs Required**:
  - Extracted digital features
  - PRO data
- **Outputs Expected**:

- **Inputs Required**:
  - Loaded raw audio data
  - Event timestamps
- **Outputs Expected**:
  - Dataset with extracted digital features (Event Hourly Frequency, Event Free Time, Average Event Amplitude, Average RMS Value)
- **Sanity Checks**:
  - Ensure extracted features match expected numerical ranges.
  - Plot intermediary results to visually inspect feature extraction accuracy.

Step 3: Descriptive Analysis

- **Inputs Required**:
  - Extracted digital features
  - PRO data
- **Outputs Expected**:
  - CSV tables containing descriptive statistics (N, mean, median, SD, range, skewness, kurtosis).
- **Sanity Checks**:
  - Verify statistical calculations are performed correctly.
  - Cross-check summary statistics with expected values for sample data.

Step 4: Known Group Validity Test

- **Inputs Required**:
  - Extracted digital features
  - PRO data
- **Outputs Expected**:
  - Results of Student's t-test between digital features and PROs.
- **Sanity Checks**:
  - Ensure p-values are computed correctly.
  - Validate that p-value thresholds are properly enforced ($p < 0.05$).

Case study: Writing codes from SAP: Python

Programming and error correction

coder:

It appears that there is an issue with accessing the 'Timestamp' column from the event timestamp data. This error typically indicates that the column name does not exist in the dataframe, possibly due to incorrect column naming or an issue with loading the data.

I will attempt to verify and correct the column name for event timestamps. Let's inspect the column names and adjust accordingly. Please provide the correct column name if it isn't 'Time

coder:

There is a `Unic` using a different

I'll attempt to loa the column nam

coder:

It seems that `.D`

Let's adjust the column names i

SV file is

o check

nalysis.

n about

- Mistakes frequently occur in AI-generated code.
- The code execution tool allows agenticAI to independently test and correct code.
- This capability reduces the need for human participation in the debugging process.
- As a result, it significantly decreases the time required for debugging and fixes.

Case study: Writing codes from SAP: Python

Feature extraction: converting raw data to feature

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.

system:

SubjectID	EventHourlyFrequency	EventFreeTimeMinutes	AverageEven
Sub4	1.5833333333333333	1439.6041666666667	0.4840537732
Sub5	1.0416666666666667	1439.7395833333333	0.5659086330
Sub7	2.4166666666666665	1439.3958333333333	0.5069658202
Sub6	1.2083333333333333	1439.6979166666667	0.5076673749
Sub2	0.75	1439.8125	0.4219697669
Sub3	1.2916666666666667	1439.6770833333333	0.5976532720
Sub1	0.8333333333333334	1439.7916666666667	0.4352093068
Sub0	1.2083333333333333	1439.6979166666667	0.4876119421
Sub17	0.6666666666666666	1439.8333333333333	0.4608806063
Sub16	1.8333333333333333	1439.5416666666667	0.5164506001
Sub14	0.625	1439.84375	0.5360812008
Sub15	1.375	1439.65625	0.4240343547
Sub11	1.25	1439.6875	0.4574872051
Sub10	2.6666666666666665	1439.3333333333333	0.5214333573
Sub12	3.5833333333333335	1439.1041666666667	0.5230997767

Case study: Writing codes from SAP: Python

Statistical analysis and plotting

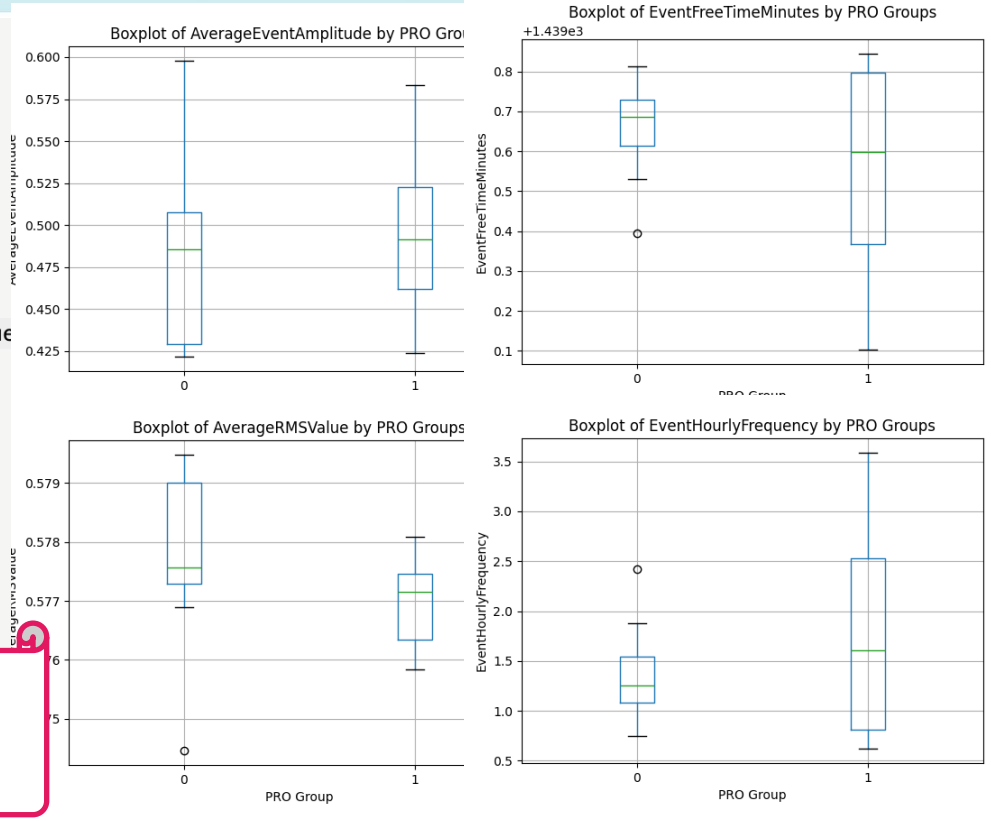
```
# Merge extracted features with PRO data
analysis_df = features_df.merge(pro_df, on='SubjectID')

# 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')

    # Create boxplot
    create_boxplot(analysis_df, feature)

print('Analysis complete and results saved to output path.')
```

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



It is important to review the code and take responsibility for it.

Responsible AI

Case study: Writing codes from SAP: Python

Providing feedback and enhancing AI behavior

The initial attempt overlooked the calculations for skewness and kurtosis.

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.

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

# Select only numeric columns for skewness and kurtosis calculations
numeric_cols = analysis_df.select_dtypes(include=[np.number])

# Calculate descriptive statistics
extended_description = numeric_cols.describe()

# Calculate skewness and kurtosis
skewness_values = numeric_cols.apply(lambda x: skew(x.dropna()))
kurtosis_values = numeric_cols.apply(lambda x: kurtosis(x.dropna()))

# Append results
extended_description.loc['skewness'] = skewness_values
extended_description.loc['kurtosis'] = kurtosis_values

# Save extended description
extended_description.to_csv(os.path.join(output_path, 'extended_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')

    # Create boxplot
    create_boxplot(analysis_df, feature)

print('Extended descriptive statistics including skewness and kurtosis have been saved.')
```

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

Scope of Acceleration

Multiagent LLM	Human Expert Data Scientist
2 min 40 sec	**~8hr**

A quicker processing unit has the potential to enhance performance even more.

Significant opportunity to help many SMEs save time and concentrate on their priorities

Limitations and challenges

- The risk of hallucinations remains, highlighting the need for a clear and concise SAP Overview. This issue could be reduced with improvements in newer LLM models.
- SAP and programming specifications should be modified for LLM applications, emphasizing textual explanations over heavy use of images or checkboxes.
- Given the higher computational demands, these models are not recommended for simpler tasks.
- Better coordination among agents is necessary to improve system reliability.
- Managing error propagation between agents continues to be challenging, impacting overall dependability.
- The system’s reliability and consistency are uncertain. Behavioral changes in response to minor SAP modifications is yet to be evaluated.

Thank you