

APPENDIX A IMPLEMENTATION DETAILS

This appendix introduces prompt examples and a history of analysis during the analysis process.

A. Task Planning and Execution Algorithm

Here is the task planning and execution algorithm we introduce in Section IV-A.

Algorithm 1 Task Planning and Execution

```

1: function TPE(Goal  $G$ , Task  $T$ , Data  $D$ , History  $H$ ,
   Decomposition_depth  $k = 1$ )
2:   if  $T$  is empty then
3:      $T_{\text{new}} \leftarrow \text{Recommender}(G, D)$ 
4:      $H \leftarrow \text{Update}(H, T_{\text{new}})$   $\triangleright$  Add new tasks to history
5:     return  $T_{\text{new}}, H$ 
6:   else
7:     if  $k > k_{\text{max}}$  then return  $\{\}, \text{False}$ 
8:      $\text{result} \leftarrow \text{Executor}(T, D)$ 
9:      $\text{completed} \leftarrow \text{Decomposer}(T, \text{result})$   $\triangleright$  Evaluate if
       the task is completed and output score and explanation
10:    if  $\text{completed}$  is False then
11:       $\text{subtasks}, \text{logic} \leftarrow \text{Decomposer}(T, D)$ 
12:      for each  $T_{\text{sub}}$  in  $\text{subtasks}$  do
13:         $\text{result}[T_{\text{sub}}], H \leftarrow \text{TPE}(G, T_{\text{sub}}, D, H, k + 1)$ 
14:         $\text{completed} \leftarrow \text{logic}(\text{result})$ 
15:        return  $\text{completed}, H$ 
16:    else
17:       $T_{\text{new}} \leftarrow \text{Recommender}(G, T, D, H)$ 
18:       $H \leftarrow \text{Update}(H, T_{\text{new}})$   $\triangleright$  Add new tasks to history
       and update progress of tasks and goal
19:      return  $T_{\text{new}}, H$ 
20:    return  $\text{completed}, H$ 

```

The algorithm aims to achieve a specified goal through recursive task decomposition and execution. Given a Goal G , Task T , Data D , History H , and Decomposition_depth k , the algorithm initially checks if the task is empty. If so, it calls a task recommender to generate a new list of tasks, which is then updated in the history. If a task T is already defined, the algorithm checks whether the current decomposition depth exceeds a maximum depth k_{max} ; if it does, it returns an empty result and a failure flag.

If the depth is within limits, the algorithm executes the current task using an executor and then assesses task completion with a decomposer. If the task is incomplete, the decomposer further breaks down the current task into a list of subtasks, which are then recursively processed by the algorithm. The results are combined based on a logical structure to determine overall task completion. The algorithm updates the history with any newly recommended tasks and finalizes completion status accordingly.

B. Prompt Examples

This section provides examples of prompts and output results when using LLM in multiple stages, including task

recommendation, visualization generation, insight generation, is completed, decompose planner, merge visualization and summarize insights.

Our prompt template introduced in Section IV-A consists of four abstract components:

- *Input*: The data or context provided to the model that is necessary for executing the task. Example: A summary of the data, the task description, and a code template.
- *Instruction*: Detailed directions that guide the model on how to perform the task. Example: Instructions to write Python code for data analysis and visualization using Altair, including adding interactive features like brush functions, tooltips, and legends.
- *Indicator*: Examples or formats that illustrate what the expected output should look like. Example: A partial code snippet or formatted output that helps the model understand the structure and format of the final result.
- *Output*: The result generated by the model based on the input and instructions provided. Example: The generated Python code, the resulting visualization, and a JSON object summarizing the insights.

1) *Task Recommendation*: This stage's prompt is designed to recommend new and previously unexplored tasks based on the goal and the analysis history.

Prompt Template 4 (Propose initial tasks).

Your need to come up with a short plan to help a user accomplish the goal: {goal}. Please recommend {n_0} exploratory tasks. For task type, you may consider the trend, correlation, category, distribution, etc., to explore the goal from different aspects. Do not use advanced modeling methods. At the end of your answer, you should write the data summary and tasks beginning with ``json and ending with ``. Summary no more than 100 words. Each task is a sentence. For data_variables, the task may not correspond to the data, so you need to go to the dataset and find the corresponding data_variables, and even if transformation is required, you need to write the original column name. There may be more than one column of data so it's an array. No need to say Here is the requested JSON format listing the tasks:. Do not add other sentences after this json data.

```

1 {
2   "tasks": ["", "", ""],
3   "data_variables": [],
4   "task_type": ["", "", ""]
5 }

```

Prompt Template 5 (Propose new tasks).

Based on the goal: "My goal is to analyze the fuel efficiency", previously explored tasks and unexplored tasks, (Extracted from analysis history)

please recommend 3 new tasks. The dataset named cars.csv has already been uploaded for you.

First, supplement some new tasks for explored tasks by considering the results of tasks already explored if needed. You just need to record the id in source in your json output,

if no just leave it none. Every new task has a source. Second, it is recommended that previous unexplored tasks be revisited if they are suitable for analysis at this stage by considering the explored tasks. You just need to record the id in tasks to review in your json output, if no just leave it none.

You also need to label a task type for new tasks.

At the end of your answer, you should write the tasks beginning with ``json and ending with ````, and the json data includes a list of new tasks and their source, the reviewed tasks and insight type will need to be analyzed. Like this:

```
1 {
2   "tasks": [""],
3   "source": [""],
4   "tasks_to_review": [],
5   "data_variables": [],
6   "task_type": [""],
7 }
```

Each task is a sentence.

An output example of recommendation.

``json

```
1 {
2   "tasks": [
3     "Determine the optimal number of cylinders that
4     balance fuel efficiency and vehicle weight.",
5     "Investigate the impact of year of manufacture on
6     the relationship between weight and fuel
7     efficiency.",
8     "Perform a cluster analysis to identify groups of
9     cars with similar weight and fuel efficiency
10    profiles."
11  ],
12  "source": [
13    "9",
14    "3",
15    "3"
16  ],
17  "tasks_to_review": [
18    "4",
19    "5"
20  ],
21  "task_type": [
22    "Optimization",
23    "Historical Analysis",
24    "Cluster Analysis"
25  ],
26  "data_variables": [
27    ["cylinders", "MPG", "Weight\in\_lbs"],
28    ["year", "Weight\in\_lbs", "MPG"],
29    ["Weight\in\_lbs", "MPG"]
30  ]
31 }
```

2) *Visualization Generation*: The following prompt is the visualization generation part of task execution. By inputting the task and data introduction, the agent generates the visual and data analysis code using Python.

Prompt Template 6 (Visualization Generation).

For the given task “Evaluate the relationship between vehicle weight and fuel efficiency.”, data summary, (Add a data introduction) you need to write Python codes to analyze the data and solve this task. The dataset cars.csv has

been read into a dataframe by `pandas.read_csv()`. Just use the variable ‘data’ to refer to this dataframe and do not read another data. After finishing the insight analysis, you should continue to use Altair to generate the visualization. Ensure that each generated visualization has a brush function that allows you to select a subset of the data. Please ensure that only one view is generated. No combination of views is required. Please give me the whole code, including data analysis and Altair, in a function called ‘plot(data)’. You only need to give the function definition of plot(data), not execute it.

please write the codes in this structure: ``python ````, there is an example:

``python

```
1 import altair as alt
2 import pandas as pd
3
4 def plot(data: pd.DataFrame):
5     # Data preprocessing
6     <codes>
7     # Chart generation
8     chart = alt.Chart().mark_bar().encode().properties
9     ()
10    return chart
11
12 ``python
```

An output example of visualization generation.

``python

```
1 import altair as alt
2 import pandas as pd
3
4 def plot(data: pd.DataFrame):
5     # Ensure correct data types
6     data = data.astype({
7         'Weight_in_lbs': float,
8         'Miles_per_Gallon': float
9     }).dropna(subset=['Weight_in_lbs', '
10    Miles_per_Gallon']) # Remove rows with
11    missing values
12
13    # Create selection features
14    click = alt.selection_multi(fields=['Name'])
15    brush = alt.selection(type='interval', encodings=['x
16    '])
17
18    # Base chart for scatter plot
19    scatter = alt.Chart(data).mark_circle(size=60).encode
20    (
21        x=alt.X('Weight_in_lbs', axis=alt.Axis(title='
22        Vehicle Weight in lbs')),
23        y=alt.Y('Miles_per_Gallon', axis=alt.Axis(title
24        ='Fuel Efficiency (MPG)')),
25        color=alt.condition(brush, 'Origin:N', alt.value
26        ('lightgray')),
27        opacity=alt.condition(click, alt.value(0.9), alt.
28        value(0.2)),
29        tooltip=['Name:N', 'Weight_in_lbs', '
30        Miles_per_Gallon', 'Year:T', 'Origin:N']
31    ).add_selection(
32        brush
33    ).add_selection(
34        click
35    ).properties(
36        width=600,
37        height=400
38    )
39
40    # Base chart for histogram
41    histogram = alt.Chart(data).mark_bar().encode(
42        x=alt.X('count()', axis=alt.Axis(title='Number
43        of Cars')),
```

```

34     y=alt.Y('Origin:N', axis=alt.Axis(title='
        Country of Origin')),
35     color=alt.Color('Origin:N', legend=alt.Legend(
        title="Origin")),
36     opacity=alt.condition(click, alt.value(0.9), alt.
        value(0.2)),
37 ).transform_filter(
38     brush
39 ).add_selection(
40     click
41 ).properties(
42     width=600
43 )
44
45 return alt.vconcat(scatter, histogram)
...

```

In actual analysis, challenges may arise, such as overly large datasets, unclear meanings of data attribute names, or highly domain-specific data, which may prevent an LLM from fully understanding the uploaded dataset. Summarizing the dataset in natural language and using this as a prompt can help the LLM perform the analysis task more effectively.

The data introduction includes a natural language description of the column names of each variable in the data, the types of variables and some statistical indicators. The given output json format template and some existing descriptions of the data (or not) are passed to LLM as prompts and let it generate data introduction according to the given format.

This is an example that we provide for the cars dataset.

```

1 class DataProperty {
2     column: string; // Column Name
3     description: string; // Natural language
      description
4     dtype: string; // Data type of the column
5     std: number | null; // Standard deviation, null if
      not applicable
6     samples: string[]; // Sample values from the
      column
7     nunique: number; // Number of unique values in the
      column
8     min: string; // Minimum value in the column
9     max: string; // Maximum value in the column
10 }

1 {
2     "columns": [
3         "Name",
4         "Miles_per_Gallon",
5         "Cylinders",
6         "Displacement",
7         "Horsepower",
8         "Weight_in_lbs",
9         "Acceleration",
10        "Year",
11        "Origin"
12    ],
13    "properties": [
14        {
15            "column": "Name",
16            "description": "The make and model of the car."
17        },
18        {
19            "column": "Miles_per_Gallon",
20            "description": "Fuel efficiency of the car,
21            measured in miles per gallon (MPG).",
22            "dtype": "number",
23            "std": 7.8159843125657815,
24            "samples": [
25                "ford galaxie 500",
26                "oldsmobile delta 88royale",
27                "ford mustang ii"
28            ],
29            "nunique": 129,
30            "min": 9.0,
31            "max": 46.6
32        }
33    ]
34 }

```

```

27     "max": "vw rabbit custom"
28     },
29     {
30         "column": "Miles_per_Gallon",
31         "description": "Fuel efficiency of the car,
32         measured in miles per gallon (MPG).",
33         "properties": {
34             "dtype": "number",
35             "std": 7.8159843125657815,
36             "samples": [
37                 19.8,
38                 18.0,
39                 16.0
40             ],
41             "nunique": 129,
42             "min": 9.0,
43             "max": 46.6
44         }
45     }
46 ]
47 }

```

3) *Insight Generation*: The following prompt is the insight generation part of task execution. Based on the python codes in visualization generation, the agent generate the structured insight with examples.

Prompt Template 7 (Insight Generation).

Please provide insight for this task: "Evaluate the relationship between vehicle weight and fuel efficiency." with the initial visualization and analysis codes. Based on previous data transformation process, you need to conduct insight analysis further to report an important insight. Before generating natural language insight text, you need to specify the column names (data variables) and data_values in the data involved in the insight, as well as the insight type. Improve insight text based on these. Note that if data variables are used in insight text, use the original column names directly. Here is an example for your output:

```

1 {
2     "text": "There is a correlation between A and B,
3     indicating xx generally exhibit lower B for xx in
4     C.",
5     "type": "Correlation",
6     "parameters": "negative",
7     "data_variables": ['A', 'B'],
8     "data_values": {
9         'A': '2000 to 5000lbs',
10        'B': '10 to 35MPG',
11        'C': 'xx'
12    }
13 }

```

An output example of insight generation.

```

1 {
2     "text": "There is a strong negative correlation between
3     Weight_in_lbs and Miles_per_Gallon, indicating
4     heavier cars generally exhibit lower
5     Miles_per_Gallon.",
6     "type": "Correlation",
7     "parameters": "negative",
8     "data_variables": [
9         "Weight_in_lbs",
10        "Miles_per_Gallon"
11    ],
12    "data_values": {
13        "Weight_in_lbs": "1613 to 5140lbs",
14        "Miles_per_Gallon": "9 to 46.6 MPG"
15    }
16 }

```

```
12 }
13 }
```

4) *Merge Visualization*: For the different selected views, given generates altair and vega-lite codes for these views, merges these views and adds interactive functions between views according to the given interaction rules.

Prompt Template 8 (Merge visualization).

There are some Altair codes:

(Add visualization codes)

for creating visualizations, please combine them into one chart and add interactive features. Do not change the number of charts in the codes list. Just use the variable 'data' to refer to this dataframe and do not read another data. You should continue use altair to generate linked visualization using given charts. I want these charts can be brushed to filter each other. Please give me the whole codes including data analysis and chart part in a function 'plot(data)'. You only need to give the function definition of plot(data), not execute it. please write the codes in this structure: ``python ``

An example of generated multiple linked-view.

```
1 import altair as alt
2 import pandas as pd
3
4 def plot(data: pd.DataFrame):
5     # Data Analysis part for the first chart
6     data['issue_date'] = pd.to_datetime(data['
7         issue_date'])
8     # Calculate total quantity of trading activity
9     # per day
10    trading_activity = data.groupby('issue_date').agg(
11        total_quantity=('quantity', 'sum')).
12        reset_index()
13    # Calculate percentage change in total
14    # quantity from the previous day
15    trading_activity['quantity_change'] =
16        trading_activity['total_quantity'].pct_change()
17
18    # Data Analysis part for the second chart
19    # Group by issue_date and calculate
20    # descriptive statistics for 'amount'
21    stats = data.groupby('issue_date')['amount'].
22        describe().reset_index()
23    stats = stats.melt(id_vars=['issue_date'],
24        value_vars=['min', '25%', '50%', '75%', 'max'],
25        var_name='statistic', value_name='
26        amount')
27
28    # Create a shared selection for interactivity
29    shared_brush = alt.selection_interval(encodings=['x'
30    ])
31
32    # Create the first chart: Trading Activity
33    # Over Time
34    chart1 = alt.Chart(trading_activity).mark_line().
35        encode(
36            x=alt.X('issue_date:T', axis=alt.Axis(title='
37                Issue Date')),
38            y=alt.Y('total_quantity:Q', axis=alt.Axis(title=
39                'Total Quantity')),
40            tooltip=['issue_date:T', 'total_quantity:Q']
41        ).properties(
42            title='Trading Activity Over Time',
43            width=600,
```

```
29         height=400
30     ).add_selection(
31         shared_brush
32     )
33
34     # Create the second chart: Distribution of
35     # Transaction Amounts
36     bar_chart = alt.Chart(stats).mark_bar().encode(
37         x=alt.X('statistic:N', axis=alt.Axis(title='
38             Statistic')),
39         y=alt.Y('amount:Q', axis=alt.Axis(title='Amount
40             ')),
41         color=alt.Color('issue_date:T', legend=None),
42         tooltip=['issue_date:T', 'statistic', 'amount
43             ']
44     ).properties(
45         title='Distribution of Transaction Amounts',
46         width=200,
47         height=400
48     ).transform_filter(
49         shared_brush
50     )
51
52     # Combine the charts using hconcat
53     combined_chart = alt.hconcat(chart1, bar_chart).
54         resolve_scale(color='shared')
55
56     return combined_chart
```

5) *Is Completed*: Based on the task content and the insights generated in the previous step, the agent scores the task according to the degree of completion and gives explanations.

Prompt Template 9 (Is completed).

For the given task: "Evaluate the relationship between vehicle weight and fuel efficiency.", and some derived insights: "There is a strong negative correlation between Weight_in_lbs and Miles_per_Gallon, indicating heavier cars generally exhibit lower Miles_per_Gallon." and codes to solve this task.

You need to judge whether the task requires further analysis. The complexity of the task can be considered from data, data mining, and visualization methods. If the task appears incomplete, it needs to be decomposed further. Please rate the task from 1-10. For example, if the initial solution adequately segments the data or if the task requires advanced statistical analysis. You should output a score and explanation of no more than 20 words for your evaluation. You should write the result beginning with ``json and ending with ```, and the json data includes a score and an explanation. The output structure is like:

```
1 {
2   "score": "",
3   "explanation": ""
4 }
```

The output is:

```
1 {
2   "score": 7/10,
3   "explanation": "Metadata extracted and relationship
4   identified, but further segmentation and in-depth
5   analysis could reveal more insights."
```

6) *Whether Decompose*: The agent determines whether the task needs to be decomposed based on the task's completion

score and explanation. If so, it returns the decomposition plan, otherwise it returns empty.

Prompt Template 10 (Decompose Planner).

For the given goal: “My goal is to analyze the fuel efficiency,” and task: “Evaluate the relationship between vehicle weight and fuel efficiency.”, the generated derived some insights: “There is a strong negative correlation between Weight_in_lbs and Miles_per_Gallon, indicating heavier cars generally exhibit lower Miles_per_Gallon.” and visualizations, discriminating whether this task requires further analysis, such as using more detailed data segmentation or advanced statistical methods.

The completion score of this task is 7/10, with the reasons: “Metadata extracted and relationship identified, but further segmentation and in-depth analysis could reveal more insights.”. If needed, please generate no more than 3 subtasks and indicate the methods they use respectively based on this. You should write the tasks begin with ``json and end with ``` , and the json data includes a list of tasks like this:

```
1 {
2   "tasks": [""],
3   "execution_order": "(Task_1 AND Task_2 AND Task_3)",
4   "methods": [""],
5   "data_variables": [[""], [], []],
6   "task_type": [""],
7 }
```

For data_variables, the task may not correspond to the data, so you need to go to the data set and find the corresponding data variables, and even if transformation is required, you need to write the original column. There are two operators “AND” and “DOWN” in the execution order to connect tasks. “AND” indicates that all subtasks need to complete successfully but there is no requirement for execution order, “DOWN” indicates that subtasks need to be executed step by step in order. Each task is a sentence no more than 20 words, methods include data segmentation, statistical methods. If do not need to decompose, output {“tasks”: ‘null’}. No need to say Here is the requested JSON format listing the tasks:.

An output example of whether decompose.

```
1 {
2   "tasks": ["Segment data by Origin to compare regional
3     differences in weight and fuel efficiency
4     correlations.", "Apply linear regression to model the
5     relationship between weight and fuel efficiency.", "
6     Investigate impact of Cylinders on fuel efficiency
7     within different weight categories."],
8   "execution_order": "(Task_1 AND Task_2 AND Task_3)",
9   "methods": ["Data segmentation", "Regression analysis", "
10    Segmentation analysis"],
11   "data_variables": [["Weight_in_lbs", "Miles_per_Gallon",
12     "Origin"], ["Weight_in_lbs", "Miles_per_Gallon"], ["
13     Cylinders", "Miles_per_Gallon", "Weight_Category"]],
14   "task_type": ["Analysis", "Modeling", "Investigation"]
15 }
```

Another two examples of whether decompose. In both examples, the ‘tasks’ represent three subtasks decomposed from the previous task, and the execution order is ‘AND’ and ‘DOWN’,

which means that the decomposition tasks are executed in the following logic and order: the first subtask is executed in parallel with the second subtask, i.e., the decomposition task is triggered again when either subtask fails, and then the third subtask is executed in serial order.

```
1 {
2   "tasks": ["Segment data by Origin to compare regional
3     differences in weight and fuel efficiency.", "Apply
4     linear regression to model the relationship between
5     weight and fuel efficiency.", "Investigate impact of
6     Cylinders on fuel efficiency within different
7     weight categories."],
8   "execution_order": "(Task_1 AND Task_2 DOWN Task_3)"
9 }
1 {
2   "tasks": ["Investigate polynomial relationships between
3     weight and MPG for each cylinder category.", "Model
4     interaction terms between cylinders, horsepower, and
5     displacement to assess combined effects on MPG.", "
6     Perform residual analysis to evaluate model fit and
7     discover potential outlier influence."],
8   "execution_order": "(Task_1 AND Task_2 DOWN Task_3)"
9 }
```

7) *Merge Visualization*: For the different selected views, the given generates Altair and Vega-Lite codes for these views, merges these views, and adds interactive functions between views according to the given interaction rules.

Prompt Template 11 (Merge visualization).

There are some Altair codes:

(Add visualization codes)

for creating visualizations, please combine them into one chart and add interactive features. Do not change the number of charts in the codes list. Just use the variable ‘data’ to refer to this dataframe and do not read other data. You should continue to use Altair to generate linked visualization using the given charts. I want these charts can be brushed to filter each other. First, you should ensure both charts contain common key columns with consistent data formats for the selection fields. Second, you should add selections based on the chart type, such as a brush on the time axis for the line chart, dual brushes for the scatter plot, and click interactions for the bar chart. Third, the layout should be no more than three charts in a row. Please give me the whole code, including the data analysis and chart part, in a function ‘plot(data)’. You only need to give the function definition of the plot(data), not execute it. please write the codes in this structure: ``python ``

8) *Summarize Insights*: Given a batch of the same task, decompose it into subtasks and their insights and execution orders, and summarize their insights.

Prompt Template 12 (Summarize Insights).

Now the decomposed sub-tasks for the task “Evaluate the relationship between vehicle weight and fuel efficiency.” are executed. The decomposition logic is “(Task_1 AND Task_2 AND Task_3)”. You need to give a summary insight for the source task based on insight from each

subtask. Here is the results for each sub-task: “There is a strong negative correlation between Weight_in_lbs and Miles_per_Gallon, indicating heavier cars generally exhibit lower Miles_per_Gallon.”. Your output should be begin with ```json and end with ``` , like this:

```
1 {"summary_insight":""}.
```

The output example of merge visualization.

```
1 {"summary_insight":"Vehicle weight negatively impacts
  fuel efficiency, with the greatest effects seen in
  heavier, USA-made cars. A linear model quantified
  this decrease as -0.0077 MPG per pound. Additionally
  , increasing engine cylinders reduces MPG across all
  weight categories, more so in heavier vehicles."}.
```

C. Task Flow

Analysis history stores the path of user historical exploration. It is a task-driven directed graph flow. Each node in the graph represents a task and the edges represent the relationship between tasks. There are three types of nodes: goal, task, subtask, and the edges include Decomposition, enlightened, and review. Subtask refers to the node obtained by task decomposition. The edge type connected from the parent task node to the child subtask node is decomposition. The edge of recommendation represents the agent recommending new tasks, and the edge of review represents the agent recommending review of nodes that have been proposed but not explored. We provide an analysis history example.

```
1 {
2   "nodes": [
3     {
4       "id": 1,
5       "group": 1,
6       "task": "My goal is to analyze influencing factors
          of fuel efficiency\n",
7       "type": "goal",
8       "variables": null,
9       "task_type": null,
10      "visualization": null,
11      "insight": null,
12      "refined_insight": null,
13      "summary_insight": null,
14      "chat_text": "The dataset contains 406 entries of
          vehicle data, including name, MPG, engine
          attributes, weight, acceleration, year, and
          origin. It has some missing values and
          requires date-time conversion for the 'Year'
          column.",
15      "error_message": null,
16      "execution_order_list": null,
17      "progress": 0.5,
18      "codes": null
19    },
20    {
21      "id": 2,
22      "group": 1,
23      "task": "Analyze the distribution of fuel
          efficiency across the dataset.",
24      "type": "task",
25      "variables": [
26        "Miles_per_Gallon"
27      ],
28      "task_type": "distribution",
29      "visualization": null,
30      "insight": null,
31      "refined_insight": null,
32      "summary_insight": null,
```

```
33      "chat_text": null,
34      "error_message": null,
35      "execution_order_list": null,
36      "progress": 0.0,
37      "codes": null
38    },
39  ],
40  {
41    "id": 3,
42    "group": 0,
43    "task": "Evaluate the relationship between vehicle
          weight and fuel efficiency.",
44    "type": "task",
45    "variables": [
46      "Weight_in_lbs",
47      "Miles_per_Gallon"
48    ],
49    "task_type": "correlation",
50    "visualization": {
51      "$schema": "https://vega.github.io/schema/vega-
          lite/v5.15.1.json",
52      "config": {
53        "view": {
54          "continuousHeight": 300,
55          "continuousWidth": 300
56        }
57      },
58      "data": {
59        "url": "http://127.0.0.1:5000/data/data_507d1
          eff-14b2-426d-b87a-b8cf18c9c741.json"
60      },
61      "height": 400,
62      "layer": [
63        {
64          "encoding": {
65            "color": {
66              "field": "Origin",
67              "legend": {
68                "title": "Origin"
69              },
70              "type": "nominal"
71            },
72            "opacity": {
73              "condition": {
74                "param": "param_3",
75                "value": 0.9
76              },
77              "value": 0.2
78            },
79            "tooltip": [
80              {
81                "field": "Name",
82                "type": "nominal"
83              },
84              {
85                "field": "Weight_in_lbs",
86                "type": "quantitative"
87              },
88              {
89                "field": "Miles_per_Gallon",
90                "type": "quantitative"
91              },
92              {
93                "field": "Year",
94                "type": "temporal"
95              },
96              {
97                "field": "Origin",
98                "type": "nominal"
99              }
100            ]
101          },
102          "x": {
103            "axis": {
104              "title": "Vehicle Weight (lbs)"
105            },
106            "field": "Weight_in_lbs",
107            "type": "quantitative"
108          },
```

```

107         "y": {
108             "axis": {
109                 "title": "Fuel Efficiency (MPG)"
110             },
111             "field": "Miles_per_Gallon",
112             "type": "quantitative"
113         },
114     },
115     "mark": {
116         "size": 60,
117         "type": "circle"
118     },
119     "name": "view_3"
120 },
121 {
122     "encoding": {
123         "color": {
124             "field": "Origin",
125             "legend": {
126                 "title": "Origin"
127             },
128             "type": "nominal"
129         },
130         "opacity": {
131             "condition": {
132                 "param": "param_3",
133                 "value": 0.9
134             },
135             "value": 0.2
136         },
137         "size": {
138             "condition": {
139                 "test": {
140                     "not": {
141                         "param": "param_3"
142                     },
143                     "value": 30
144                 },
145                 "value": 60
146             },
147             "tooltip": [
148                 {
149                     "field": "Name",
150                     "type": "nominal"
151                 },
152                 {
153                     "field": "Weight_in_lbs",
154                     "type": "quantitative"
155                 },
156                 {
157                     "field": "Miles_per_Gallon",
158                     "type": "quantitative"
159                 },
160                 {
161                     "field": "Year",
162                     "type": "temporal"
163                 },
164                 {
165                     "field": "Origin",
166                     "type": "nominal"
167                 }
168             ],
169             "x": {
170                 "axis": {
171                     "title": "Vehicle Weight (lbs)"
172                 },
173                 "field": "Weight_in_lbs",
174                 "type": "quantitative"
175             },
176             "y": {
177                 "axis": {
178                     "title": "Fuel Efficiency (MPG)"
179                 },
180                 "field": "Miles_per_Gallon",
181                 "type": "quantitative"
182             },
183             "mark": {
184                 "size": 60,
185                 "type": "circle"
186             }
187         },
188         "params": [
189             {
190                 "name": "param_3",
191                 "select": {
192                     "encodings": [
193                         "x"
194                     ],
195                     "type": "interval"
196                 },
197                 "views": [
198                     "view_3"
199                 ]
200             },
201             {
202                 "name": "param_4",
203                 "select": {
204                     "fields": [
205                         "Name"
206                     ],
207                     "type": "point"
208                 },
209                 "views": [
210                     "view_3"
211                 ]
212             }
213         ],
214         "resolve": {
215             "scale": {
216                 "color": "independent",
217                 "size": "independent"
218             },
219             "width": 600
220         },
221         "insight": {
222             "text": "There is a strong negative correlation
223                 between Weight_in_lbs and
224                 Miles_per_Gallon, indicating heavier cars
225                 generally exhibit lower Miles_per_Gallon
226                 .",
227             "type": "Correlation",
228             "parameters": "negative",
229             "data_variables": [
230                 "Weight_in_lbs",
231                 "Miles_per_Gallon"
232             ],
233             "data_values": {
234                 "Weight_in_lbs": "1613 to 5140lbs",
235                 "Miles_per_Gallon": "9 to 46.6 MPG"
236             }
237         },
238         "refined_insight": null,
239         "summary_insight": "Vehicle weight negatively
240             impacts fuel efficiency, with the greatest
241             effects seen in heavier, USA-made cars. A
242             linear model quantified this decrease as -0.00
243             77 MPG per pound. Additionally, increasing
244             engine cylinders reduces MPG across all weight
245             categories, more so in heavier vehicles.",
246         "chat_text": null,
247         "error_message": "null",
248         "execution_order_list": [
249             "AND",
250             "DOWN"
251         ],
252         "progress": 1,
253         "codes": "import altair as alt\nimport pandas as
254             pd\nndef plot(data: pd.DataFrame):\n #

```

```

248     },
249     {
250         "id": 7,
251         "group": 0,
252         "task": "Segment data by Origin to compare
253                 regional differences in weight and fuel
254                 efficiency correlations.",
255         "type": "sub_task",
256         "variables": [
257             "Weight_in_lbs",
258             "Miles_per_Gallon",
259             "Origin"
260         ],
261         "task_type": "Analysis",
262         "visualization": null,
263         "insight": null,
264         "refined_insight": null,
265         "summary_insight": null,
266         "chat_text": null,
267         "error_message": null,
268         "execution_order_list": null,
269         "progress": 0.0,
270         "codes": null
271     },
272     {
273         "id": 8,
274         "group": 0,
275         "task": "Apply linear regression to model the
276                 relationship between weight and fuel
277                 efficiency.",
278         "type": "sub_task",
279         "variables": [
280             "Weight_in_lbs",
281             "Miles_per_Gallon"
282         ],
283         "task_type": "Modeling",
284         "visualization": null,
285         "insight": null,
286         "refined_insight": null,
287         "summary_insight": null,
288         "chat_text": null,
289         "error_message": null,
290         "execution_order_list": null,
291         "progress": 0.0,
292         "codes": null
293     }
294 ],
295 "edges": [
296     {
297         "source": 1,
298         "target": 2,
299         "weight": 1.0,
300         "type": "Enlightenment",
301         "labels": [],
302         "differences": {
303             "changes": {
304                 "variables_change": {
305                     "original": [],
306                     "new": [
307                         "Miles_per_Gallon"
308                     ]
309                 },
310                 "type_change": {
311                     "original": [
312                         "goal"
313                     ],
314                     "new": [
315                         "task"
316                     ]
317                 },
318                 "insight_change": {
319                     "original": [],
320                     "new": [
321                         "DISTRIBUTION"
322                     ]
323                 }
324             },
325             "costs": {
326                 "variables_cost": 1.0,
327                 "type_cost": 1.0,
328                 "insight_cost": 1.0,
329                 "total_difference_cost": 1.0
330             }
331         }
332     },
333     {
334         "source": 1,
335         "target": 3,
336         "weight": 1.0,
337         "type": "Enlightenment",
338         "labels": [],
339         "differences": {
340             "changes": {
341                 "variables_change": {
342                     "original": [],
343                     "new": [
344                         "Weight_in_lbs",
345                         "Miles_per_Gallon"
346                     ]
347                 },
348                 "type_change": {
349                     "original": [
350                         "goal"
351                     ],
352                     "new": [
353                         "task"
354                     ]
355                 },
356                 "insight_change": {
357                     "original": [],
358                     "new": [
359                         "CORRELATION",
360                         "CORRELATION"
361                     ]
362                 }
363             },
364             "costs": {
365                 "variables_cost": 1.0,
366                 "type_cost": 1.0,
367                 "insight_cost": 1.0,
368                 "total_difference_cost": 1.0
369             }
370         }
371     }
372 ]

```



```

366     }
367   },
368 },
369 {
370   "source": 3,
371   "target": 4,
372   "weight": 0.2760920571391645,
373   "type": "Decomposition",
374   "labels": [],
375   "differences": {
376     "changes": {
377       "variables_change": {
378         "original": [
379           "Weight_in_lbs",
380           "Miles_per_Gallon"
381         ],
382         "new": [
383           "Weight_in_lbs",
384           "Miles_per_Gallon",
385           "Origin"
386         ]
387       },
388       "type_change": {
389         "original": [
390           "task"
391         ],
392         "new": [
393           "sub_task"
394         ]
395       },
396       "insight_change": {
397         "original": [
398           "CORRELATION",
399           "CORRELATION"
400         ],
401         "new": [
402           "CORRELATION",
403           "ANALYSIS"
404         ]
405       }
406     },
407     "costs": {
408       "variables_cost": 0.18350341907227408,
409       "type_cost": 1.0,
410       "insight_cost": 0.29289321881345254,
411       "total_difference_cost": 0.276092057139164
412     }
413   },
414 },
415 {
416   "source": 3,
417   "target": 5,
418   "weight": 0.200000000000000018,
419   "type": "Decomposition",
420   "labels": [],
421   "differences": {
422     "changes": {
423       "variables_change": {
424         "original": [
425           "Weight_in_lbs",
426           "Miles_per_Gallon"
427         ],
428         "new": [
429           "Weight_in_lbs",
430           "Miles_per_Gallon"
431         ]
432       },
433       "type_change": {
434         "original": [
435           "task"
436         ],
437         "new": [
438           "sub_task"
439         ]
440       },
441       "insight_change": {

```

```

442     "original": [
443       "CORRELATION",
444       "CORRELATION"
445     ],
446     "new": [
447       "REGRESSION ANALYSIS",
448       "MODELING"
449     ]
450   },
451   "costs": {
452     "variables_cost": 2.220446049250313e-16,
453     "type_cost": 1.0,
454     "insight_cost": 1.0,
455     "total_difference_cost": 0.2000000000000000
456   }
457 },
458 },
459 },
460 {
461   "source": 5,
462   "target": 2,
463   "weight": 0.0,
464   "type": "Review",
465   "labels": [],
466   "differences": "Node(s) not found"
467 }
468 ]
469 }

```

APPENDIX B LLM PERFORMANCE TEST

This appendix analyzes the performance of GPT-3.5-turbo and GPT-4-turbo on data analysis and visualization tasks. We tested 160 tasks across two datasets, focusing on error types and model efficiency. The results highlight common error categories and compare the models' performance in terms of error rates and time costs.

A. Error Analysis

To enhance the robustness of our system, we conducted tests focusing on 2 datasets, with 20 tasks per dataset, utilizing GPT-3.5-turbo and GPT-4-turbo. Each task was tested with data analysis code generation and insight annotation. Both of these two steps tested two prompts to support the choice of better results. This setting resulted in a total of $2 \text{ (datasets)} \times 10 \text{ (tasks per dataset)} \times 2 \text{ (models)} \times 2 \text{ (codes generation prompts)} \times 2 \text{ (insight annotation prompts)} = 160$ initial results.

To improve error handling, we allow the model to self-debug again while we prompt the model with cached error messages, yielding an overall 204 experimental results. In the first round, out of the 160 tasks, 44 errors were identified. Upon a second verification, 18 of these errors were resolved, while 26 remained erroneous.

1) *Error Types*: We classified these 70 (44 + 26) errors we found into five categories, as illustrated in Fig. 8.

- **Unfamiliar Dataset**: These errors highlight the model's lack of understanding regarding the structure and methods applicable to the given dataset. The model attempts to use non-existent attributes or incorrect methods, indicating a gap in its knowledge about the dataset's schema and the operations that can be performed on it.

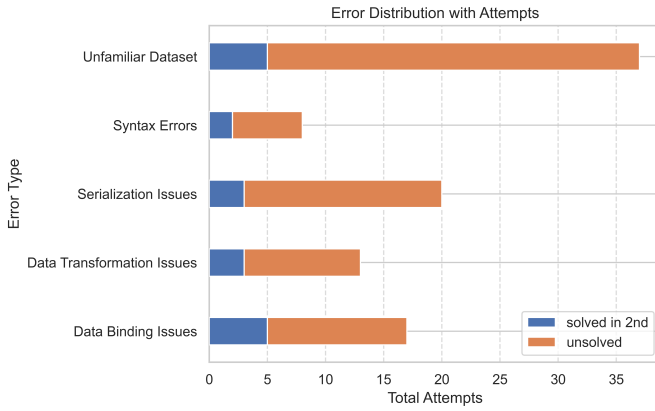


Fig. 8. Distribution of error types encountered across various attempts, categorized by whether the issue was resolved in a second attempt or remained unsolved. Error types include issues related to unfamiliar datasets, syntax, serialization, data transformation, and data binding.

- ‘DataFrame’ object has no attribute ‘reset_src_index’
- Cannot reset_index inplace on a Series to create a DataFrame
- **Data Binding Issues:** These errors occur when the model incorrectly references variable names that do not exist in the given context. This suggests that the model has difficulty maintaining consistency in variable naming and ensuring that all referenced variables are defined and accessible within the code.
 - name ‘actuator_line’ is not defined
 - name ‘hp_preusline’ is not defined
- **Serialization Issues:** These errors arise when the model attempts to use data types that are not compatible with JSON serialization, which is required for certain operations such as visualization with Altair. These errors indicate that the model does not adequately understand the requirements for data serialization and the need to use JSON-compatible data types.
 - Object of type Interval is not JSON serializable
 - Object of type Period is not JSON serializable
- **Data Transformation Issues:** These errors indicate the model’s difficulty in converting data from one type to another and identifying correct data types. The first example demonstrates a failure to convert a string of concatenated names into a numeric format, while the second highlights the challenge of correctly identifying and spelling field names.
 - Could not convert ... to numeric
 - Unable to determine data type for the field "Origin"
- **Syntax Errors:** Syntax errors occur when the model generates code that does not adhere to the correct syntax rules of the programming language. These errors suggest that the model may overlook essential syntax elements such as matching parentheses or correctly structuring the code.
 - invalid syntax (<string>, line 37)

2) *Comparison of Models:* Based on the analysis of error type distribution with model comparison, we found that:

GPT-3.5-turbo has lower overall error counts: In terms of these code generation task types, the overall *error rate ratio* of GPT-3.5-turbo and GPT-4-turbo is 1:2. This indicates that GPT-4-turbo commits more errors overall compared to GPT-3.5-turbo, as shown by the larger counts of unsolved errors.

Repair Ability of GPT-4-turbo: Despite the higher error rate, GPT-4-turbo shows significant repair ability in the second round, solving many errors that were initially unsolved (Fig. 9). This repair ability is evident across all error types, particularly in *Unfamiliar Dataset* and *Data Binding Issues*.



Fig. 9. Distribution of error types across two models (GPT-4-turbo and GPT-3.5-turbo), showing the count of errors resolved in a second attempt or left unsolved.

3) *Comparison of Tasks:* The bar chart illustrates the relationship between various *task types* and specific *error types* encountered in data analysis and visualization using large models. The tasks for the tests are shown in Table 1.

For *Correlation Analysis*, the primary errors are *Data Transformation Issues* and *Data Binding Issues*, with occasional *Syntax Errors*. In *Trend Analysis*, the most frequent issues are *Data Transformation Issues*, indicating challenges in converting data formats or structures, and *Data Binding Issues*, which point to problems in linking data with visualization elements.

Segmentation Analysis also suffers from *Data Transformation Issues* and *Data Binding Issues*. For *Predictive Modeling*, significant errors include *Serialization Issues*, suggesting difficulties in data serialization processes, along with *Data Binding Issues*.

Variance Analysis encounters substantial *Unfamiliar Dataset* issues, highlighting challenges in working with new or unexpected data, and occasional *Serialization Issues*. *Cluster Analysis* is predominantly affected by *Unfamiliar Dataset* issues, reflecting problems with novel dataset, and some *Data Transformation Issues*.

Aggregation Analysis shows minimal errors with an even distribution across different types, while *Distribution Analysis* is mainly affected by *Data Binding Issues*. *Margin Analysis* exhibits few errors, with *Unfamiliar Dataset* and *Data Binding Issues* being the most frequent.

Overall, *Unfamiliar Dataset* and *Data Binding Issues* are the most prevalent errors across various task types, underscoring the challenges in handling novel data and effectively linking data with visualization components.

TABLE I

TASKS ANALYSIS: THE TASKS ARE DERIVED FROM THE DATASETS USED IN THE EVALUATION AND PROPOSED BY THE AGENT. THE “Cars” DATASET IS ANALYZED IN THE FIRST TEN TASKS, WHILE THE “Superstore” DATASET IS ANALYZED IN THE SUBSEQUENT TEN TASKS. THE TYPES OF TASKS ARE LABELED BY THE LLM.

dataset	task	type
Cars	Analyze the distribution of fuel efficiency across the dataset.	Distribution Analysis
Cars	Evaluate the relationship between vehicle weight and fuel efficiency.	Correlation Analysis
Cars	Compare the average horsepower for vehicles of different origins.	Comparative Analysis
Cars	Examine how vehicle characteristics have changed over time.	Trend Analysis
Cars	Assess the distribution of engine cylinders in the dataset.	Distribution Analysis
Cars	Segment data by Origin to compare regional differences in weight and fuel efficiency correlations.	Segmentation Analysis
Cars	Apply linear regression to model the relationship between weight and fuel efficiency.	Predictive Modeling
Cars	Investigate the impact of Cylinders on fuel efficiency within different weight categories.	Correlation Analysis
Cars	Analyze the variance in MPG within each cylinder count across weight categories.	Variance Analysis
Cars	Perform a cluster analysis to identify groups of cars with similar weight and fuel efficiency profiles.	Cluster Analysis
Superstore	Analyze the correlation between sales and profit to understand how they are related at a product or category level.	Correlation Analysis
Superstore	Aggregate sales by category and sub-category to identify which product groups are performing best.	Aggregation Analysis
Superstore	Examine the distribution of sales and profit to determine common ranges and any outliers or anomalies in the data.	Distribution and Outlier Analysis
Superstore	Compare average discount rates across different segments (Consumer, Corporate, Home Office) to assess the impact of discounts on sales and profit.	Correlation Analysis
Superstore	Segment monthly sales data by product categories.	Segmentation Analysis
Superstore	Analyze sales trends of top-selling and bottom-selling products.	Trend Analysis
Superstore	Perform cluster analysis to group similar products based on sales trends and profitability.	Cluster Analysis
Superstore	Investigate the impact of promotions or discounts on monthly sales trends by product category.	Correlation Analysis
Superstore	Determine the product groups with the highest profit margin within each category.	Margin Analysis
Superstore	Analyze the impact of seasonal trends on profit margins by category and sub-category.	Trend Analysis

Here’s two generated codes and errors for *Trend Analysis* and For *Correlation Analysis*:

- *Trend Analysis* has the highest occurrence of *Data Binding Issues*, suggesting frequent difficulties in linking data attributes correctly for trend-related tasks.

– Line 21: name ‘top_slowing’ is not defined

```

1 import altair as alt
2 import pandas as pd
3
4 def plot(data: pd.DataFrame):
5     # Data Analysis: Calculate Total Sales and Total
      Profit for each Product
6     product_summary = data.groupby('Product Name').agg(
7         Total_Sales=pd.NamedAgg(column='Sales', aggfunc='
          sum'),
8         Total_Profit=pd.NamedAgg(column='Profit', aggfunc=
          'sum'),
9         Total_Quantity=pd.NamedAgg(column='Quantity',

```

```

      aggfunc='sum')
10 ).reset_index()
11
12 # Sort by Total Sales to find Top Selling and
      Bottom Selling Products
13 top_selling = product_summary.sort_values(by='
      Total_Sales', ascending=False).head()
14 bottom_selling = product_summary.sort_values(by='
      Total_Sales', ascending=True).head()
15
16 # Adding annotations: profit for top and bottom
      selling products
17 top_selling['Annotation'] = 'Top Selling'
18 bottom_selling['Annotation'] = 'Bottom Selling'
19
20 # Combine the data for top and bottom selling
      for visualization
21 annotated_data = pd.concat([top_slowing,
      bottom_selling], axis=0)
22
23 # Chart part

```

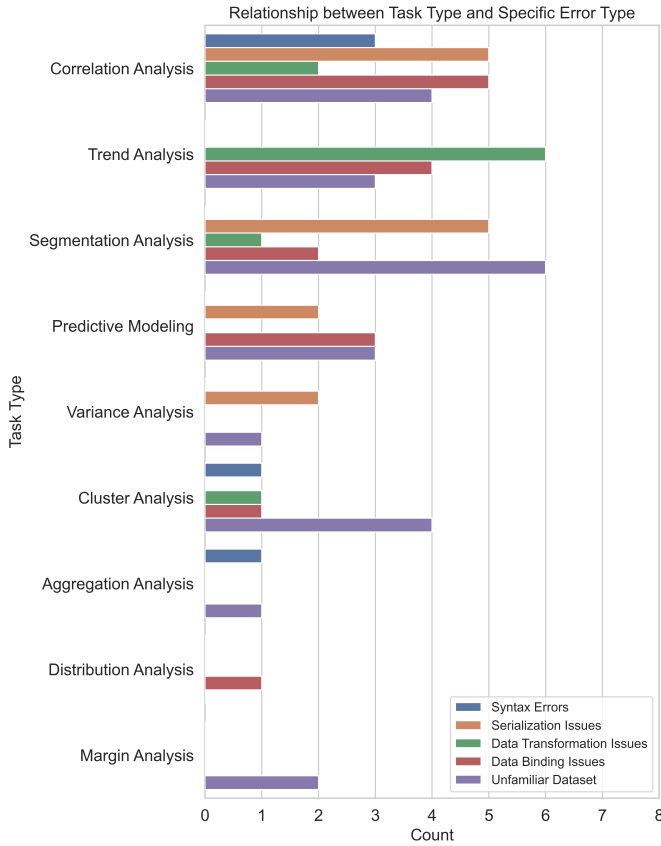


Fig. 10. Distribution of error types across various analytical tasks, highlighting the susceptibility of each task type (e.g., correlation, trend, segmentation) to specific errors like syntax issues, data binding problems, and unfamiliar datasets.

```

24 brush = alt.selection_interval(encodings=['x', 'y'],
25                               name='Brush select')
26 chart = alt.Chart(annotated_data).mark_bar().encode(
27   x=alt.X('Product Name:N', sort=None, title='
28     Product Name'),
29   y=alt.Y('Total_Sales:Q', title='Total Sales'),
30   color=alt.Color('Annotation:N', legend=alt.
31     Legend(title='Product Category')),
32   tooltip=['Product Name', 'Total Sales', '
33     Total_Profit', 'Total_Quantity']
34 ).add_selection(
35   brush
36 ).transform_filter(
37   brush
38 )
39 return chart

```

- **Correlation Analysis** frequently encounters *Syntax Errors* and *Data Transformation Issues*, indicating common problems with code syntax and data type handling in these tasks.
 - invalid syntax (<string>, line 37)

```

1 import altair as alt
2 import pandas as pd
3
4 def plot(data: pd.DataFrame):
5
6   # Data Analysis part
7   # Perform linear regression using Altair's
8     transform_regression
9   regression = alt.transform_regression(
10     'Weight_in_lbs', 'Miles_per_Gallon', method
11     ='linear"

```

```

12 # Chart part
13 brush = alt.selection_interval(encodings=['x
14   ', 'y'], name='brush')
15
16 # Points for the scatter plot - brushable
17 points = alt.Chart(data).mark_point(color='
18   blue').encode(
19   x='Weight_in_lbs:Q',
20   y='Miles_per_Gallon:Q',
21   tooltip=['Name', 'Weight_in_lbs', '
22     Miles_per_Gallon']
23 ).add_selection(
24   brush
25 ).transform_filter(
26   brush
27 )
28
29 # Line for linear regression - highlighted by
30 brush
31 trendline = alt.Chart(data).
32   transform_regression(
33     'Weight_in_lbs', 'Miles_per_Gallon',
34     method='linear'
35 ).mark_line(color='red').encode(
36   x='Weight_in_lbs:Q',
37   y='Miles_per_Gallon:Q'
38 ).transform_filter(
39   brush
40 )
41
42 # Combining both the points and trend line
43 chart = points + trendline
44
45 return chart

```

B. Visualization Annotations

To further assess the model's ability to generate visualizations, such as highlighting insights on visualizations, which is more difficult than drawing an overview, we tested these two different situations. By comparing the *error rate* with or without annotation requirements, we find that *adding annotations* does increase the probability of errors, especially making errors from unfamiliar datasets and data binding.

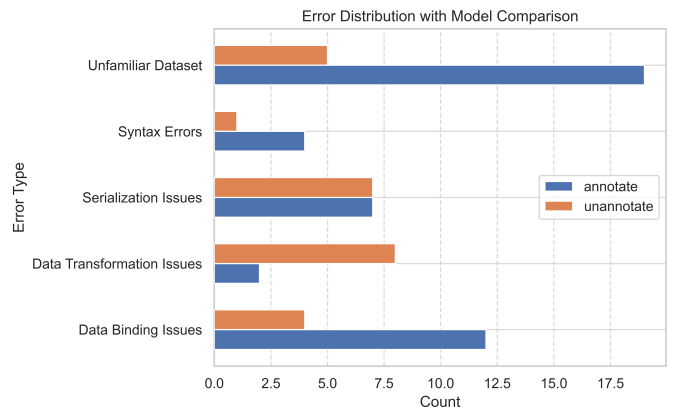


Fig. 11. Error rates analysis by visualization prompts (annotated or non-annotated).

Specifically, for data binding issues, there are 12 errors with annotations compared to 4 without, indicating that the complexity introduced by annotations increases the likelihood of errors. Similarly, for unfamiliar datasets, the model makes 19 errors with annotations but only 5 without, suggesting

that annotations make it harder for the model to handle unfamiliar data. On the other hand, for data transformation issues, the model performs better with annotations, making only 2 errors compared to 8 without. Serialization issues show an equal distribution of 7 errors for both annotated and unannotated cases, while syntax errors are slightly higher with annotations (4 errors) compared to without (1 error). Results highlight the model's difficulty in managing additional annotation complexity.

C. Insight Generation

Among all 160 tasks, after self-debugging, 134 successfully generated insight text content that provided some level of analysis of the tasks. However, among these, 18 did not generate the insight in the required JSON format.

We conducted a statistical analysis to compare the success rates of generating each key-value pair in the standardized insight dictionaries.

To generate the insights, we employed two different approaches. The first approach involved generating the insight in JSON format first, while the second approach generated the natural language text first. The following are the results of these two different methods.

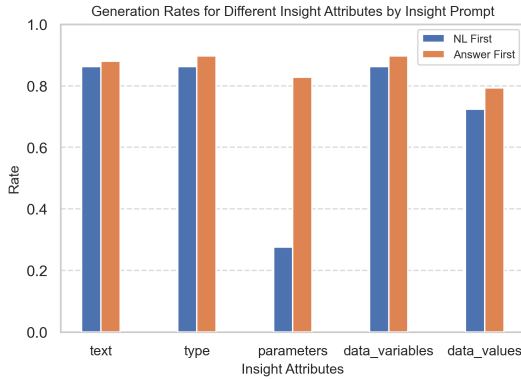


Fig. 12. The insight attributes generation success rates (e.g., text, type, parameters) based on different prompts, with Answer first or Natural Language (NL) first.

D. Time Cost

To evaluate the efficiency of the models in terms of processing time, we measured the time costs for both code generation and insight generation tasks. From the point of view of time, GPT-3.5-turbo is significantly less than 4, which is 1/4 of the time of GPT-4-turbo.

Code generation time: GPT-3.5-turbo demonstrates consistent performance with average times of 5.05 seconds for the cars dataset and 5.58 seconds for the superstore dataset, and relatively low variances of 1.32 and 2.34, respectively. In contrast, GPT-4-turbo has significantly higher average code generation times of 20.79 seconds for the cars dataset and 24.77 seconds for the superstore dataset, with much larger variances of 22.65 and 61.71, respectively, indicating greater variability and potentially less optimization.

Insight generation time: GPT-3.5-turbo again shows stable performance with average times of 3.36 seconds for the cars

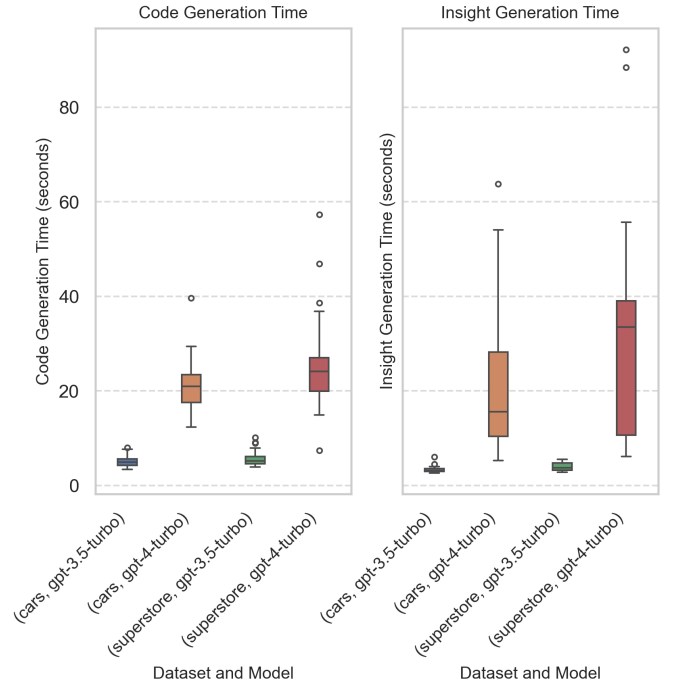


Fig. 13. Time cost analysis for code and insight generation across different datasets (cars, superstore) and models (GPT-3.5-turbo, GPT-4-turbo), illustrating the variations in generation time based on model and dataset combination.

dataset and 3.94 seconds for the superstore dataset, and low variances of 0.44 and 0.70, respectively. On the other hand, GPT-4-turbo has much higher average insight generation times of 21.39 seconds for the cars dataset and 32.43 seconds for the superstore dataset, with significantly larger variances of 216.26 and 558.72, respectively, suggesting substantial variability and a need for further optimization.

Overall, GPT-3.5-turbo demonstrates more efficient and consistent performance in both code and insight generation tasks across different datasets. GPT-4-turbo, while potentially more powerful, shows higher time costs but greater variability, suggesting it may be more sensitive to the complexity of tasks and datasets.