Domain-Aware Prompting with LLMs for Data Science Notebooks

Sanjeet Chatterjee

Large Language Models (LLMs)

Transformer Architecture

Autoregressive neural networks that employ self-attention. This allows elements to interact directly and capture dependencies efficiently.

Emergent Capabilities

Scaling such models to billions of parameters has led to emergent behaviour such as in-context learning, multi-step reasoning and instruction following.

Coding Assistants

LLMs have demonstrated impressive results in translating natural language to code, giving rise to coding assistants such as GitHub Copilot.

Data Science Notebooks

Rapid Prototyping

Interactive Jupyter notebooks interleaving code and natural language markdown cells have become the standard for data science workflows.

Repetitive Workflows

Data scientists often go through repetitive workflows in data exploration and preparation.

Challenging for LLMs

Notebooks specifically offer additional challenges due to the multi-step nature of notebooks.

ARCADE: Exploratory Analysis

Table 2.1: Existing Datasets for Data Science Notebooks

Dataset	No. Notebooks	No. Tasks	Tasks / Notebook	Evaluation Method
JuICE	1457	3946	2.7	Exact Match + BLEU
MS DSP	305	1096	3.6	Unit Tests
ExeDS	277	534	1.9	Output Match
ARCADE	133	1082	8.55	Output Match

ARCADE: Exploratory Analysis

Table 2.1: Existing Datasets for Data Science Notebooks

Dataset	No. Notebooks	No. Tasks	Tasks / Notebook	Evaluation Method
JuICE	1457	3946	2.7	Exact Match + BLEU
MS DSP	305	1096	3.6	Unit Tests
ExeDS	277	534	1.9	Output Match
ARCADE	133	1082	8.55	Output Match

Table 2.4: Pandas Method Groups

Task Type	%	Methods
Aggregation	42.7	groupby, agg ...
Transformation	27.8	apply, pivot_table, explode, cut, pct_change ...
Combination	3.0	concat, merge, append, join ...
Selection	68.5	loc, query, nlargest, sort_values, filter, isnull ...
Cleaning	22.4	fillna, rename, drop_duplicates, to_numeric ...
Strings	23.7	extract, startswith, str, replace, contains...
Computation	81.3	max, quantile, value_counts, div, corr ...
Datetime	2.7	to_datetime, strftime, period_range, to_timedelta ...
Visualization	5.9	plot, boxplot, barplot, scatter, hist ...

ARCADE: Exploratory Analysis

Table 2.1: Existing Datasets for Data Science Notebooks

Dataset	No. Notebooks	No. Tasks	Tasks / Notebook	Evaluation Method
JuICE	1457	3946	2.7	Exact Match + BLEU
MS DSP	305	1096	3.6	Unit Tests
ExeDS	277	534	1.9	Output Match
ARCADE	133	1082	8.55	Output Match

Table 2.4: Pandas Method Groups

Task Type	%	Methods
Aggregation	42.7	groupby, agg ...
Transformation	27.8	apply, pivot_table, explode, cut, pct_change ...
Combination	3.0	concat, merge, append, join ...
Selection	68.5	loc, query, nlargest, sort_values, filter, isnull ...
Cleaning	22.4	fillna, rename, drop_duplicates, to_numeric ...
Strings	23.7	extract, startswith, str, replace, contains...
Computation	81.3	max, quantile, value_counts, div, corr ...
Datetime	2.7	to_datetime, strftime, period_range, to_timedelta ...
Visualization	5.9	plot, boxplot, barplot, scatter, hist ...

Figure 2.5: Tasks per Notebook

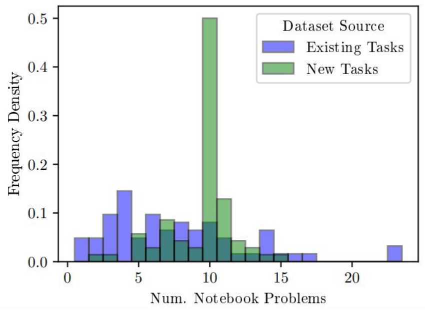
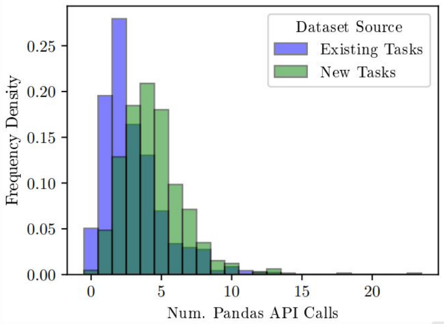


Figure 2.6: Pandas API Calls per Task



ARCADE: Sample Notebook

[1] c_1

```python
import pandas as pd

df = pd.read_csv('dataset/Gamepass_Games_v1.csv')
```

[2] u_1 Extract min and max hours as two columns

c_2

```python
def get_avg(x):
  try: return float(x[0]) , float(x[1])
  except: return 0, 0

df['min'], df['max'] = zip(*df['TIME'].str.replace(
  ' hours','').str.split("-").apply(get_avg))
```

[3] c_3

```python
df['ADDED'] = pd.to_datetime(
    df['ADDED'],format="%d %b %y",errors='coerce')
```

[4] u_2 In which year was the most played game added?

c_4

```python
df['GAMERS']=df['GAMERS'].str.replace(
                             ',',' ').astype(int)
added_year=df[df['GAMERS'].idxmax()]['ADDED'].year
```

[5] u_3 For each month in that year, how many games that has a rating of more than four?

c_5

```python
df[(df['ADDED'].dt.year== added_date.year) &
(df['RATING']>4)].groupby(
                df["ADDED"].dt.month)['GAME'].count()
```

[6] u_4 What is the average maximum completion time for all fallout games added in 2021?

c_6

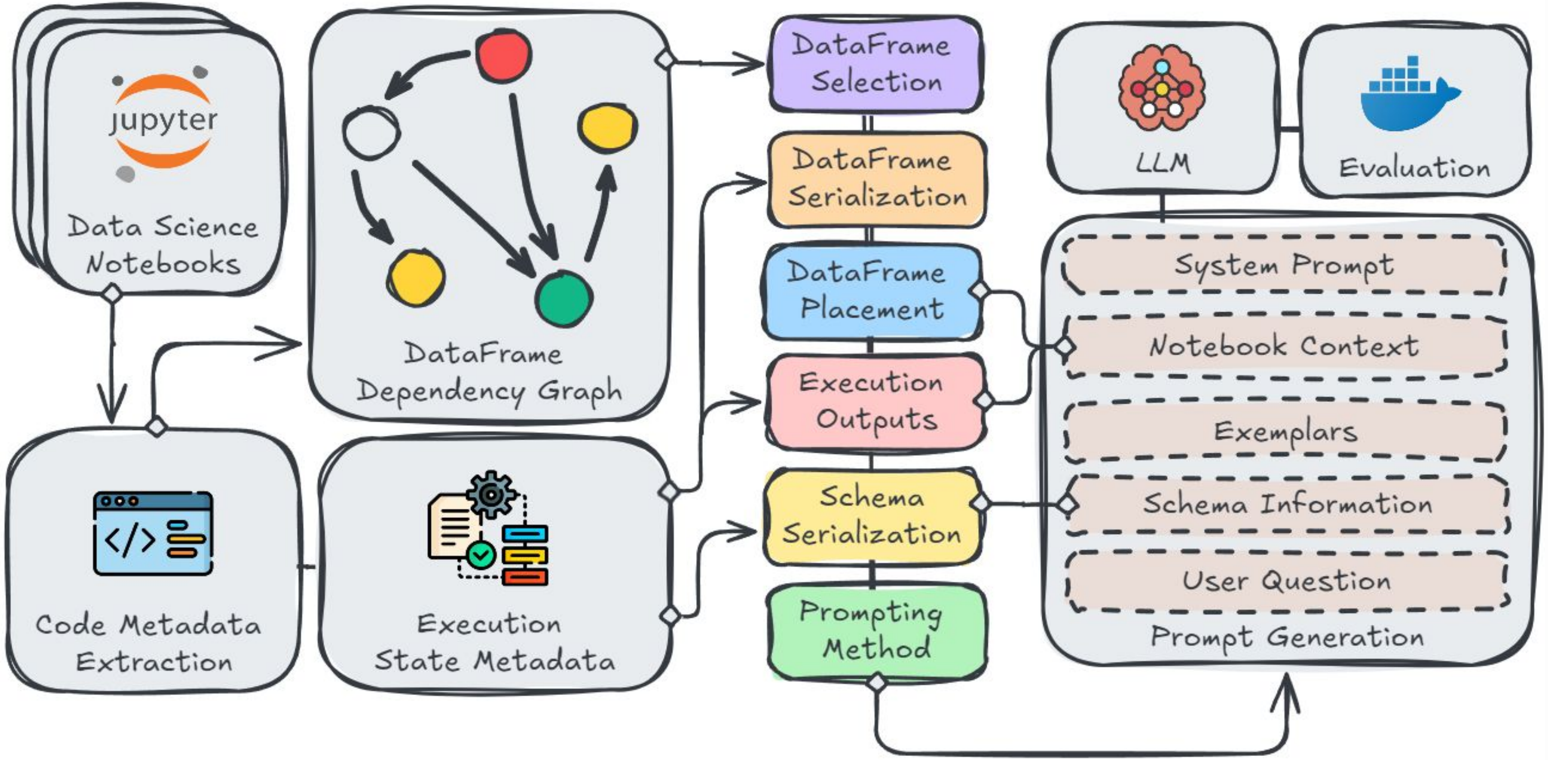
```python
fallout=df[df['GAME'].str.contains('Fallout')]
fallout.groupby(fallout['ADDED'].dt.year).get_group(
  2021)['max'].mean()
```

[7] u_5 What is the amount of games added in each year for each month? <u>(show a table with index as years, columns as months and fill null values with 0)</u>

c_7

```python
pd.pivot_table(df, index=df['ADDED'].dt.year, ...,
  aggfunc=np.count_nonzero,
  fill_value='0').rename_axis(
    index='Year', columns='Month')
```

System Overview



Execution State Metadata

- ➔ Imports
- ➔ Variables
- ➔ Functions

Python Runtime

- ➔ Markdown
- ➔ Code Cells
- ➔ Outputs

Notebook

Cell Outputs

Cell execution results

DataFrame Variables

Pandas variables content

DataFrame Serialization

NEW

```
# In[]:
print(df_anim)

# Out[]:
[showing 3 sample rows out of 1812 rows]
                              theme (list)              name (str)  rating (float64)
124    ["Historical", ..., "Gag Humor"]               "Gintama"              9.05
1303                           ["School"]        "Kareshi Jijou"              7.60
1488                     ["Mahou Shoujo"]  "Delicious Precure"              7.24
```

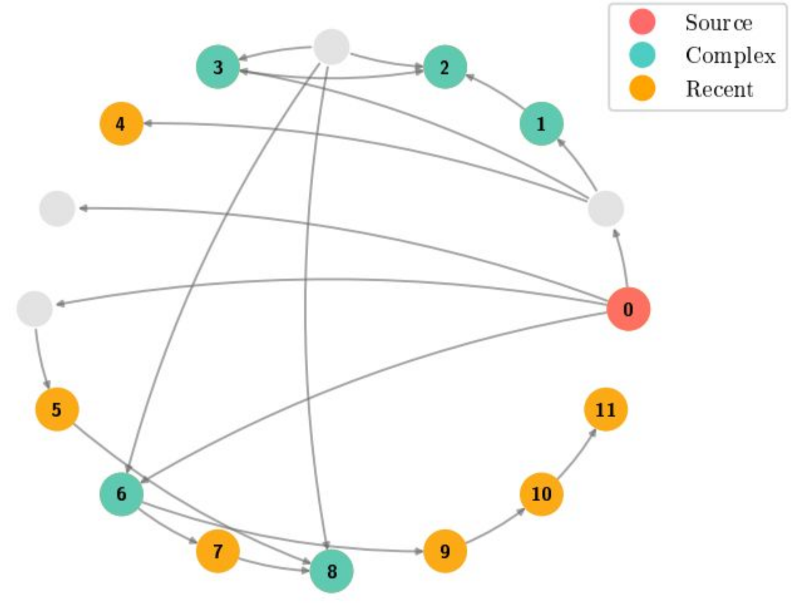
ORIGINAL

```
# Columns in df_anim with example values:
# theme ([School]), name (Kareshi Jijou), rating (7.60)
```

- Rounding Floats
- Truncating Strings
- Column Types
- Limit Rows
- String Quotes
- DataFrame Placement

DataFrame Dependency Graph

```python
from pandas import Series, DataFrame
import pandas as pd

df = pd.read_csv('NYC_Restaurants.csv', dtype=str)

df_noduplicates = df.drop_duplicates(subset='RESTAURANT')
df_notchains = df_noduplicates.groupby("DBA").filter(lambda x: len(x) == 1)
boro_notchain_pivot = pd.pivot_table(df_notchains, index = 'BORO', values = 'RESTAURANT', aggfunc = lambda x: len(x.unique()))
boro_restaurant_pivot = pd.pivot_table(df_noduplicates, index = 'BORO', values = 'RESTAURANT', aggfunc = lambda x: len(x.unique()))
boro_notchain_pivot['TOTAL RESTAURANTS'] = boro_restaurant_pivot

cuis = df_noduplicates['CUISINE DESCRIPTION'].value_counts()
mask = (df['VIOLATION CODE']).isnull()
no_violations = df[mask]
no_violations[['CUISINE DESCRIPTION', 'RESTAURANT']]

cuisine_no_violations = no_violations['CUISINE DESCRIPTION'].value_counts()
mask = (df['VIOLATION CODE']).notnull()
violations = df[mask]
cuisine_violations = violations['CUISINE DESCRIPTION'].value_counts()

total_cuisine = pd.concat([cuisine_violations , cuisine_no_violations], axis = 1)
violations = pd.crosstab(df['BORO'], df['VIOLATION DESCRIPTION']).query('BORO != ["Missing"]')

vstack = violations.stack()
violations2 = vstack.unstack('BORO')
mostcommon = DataFrame({'Most Common Complaint':violations2.idxmax(),'Number of Complaints':violations2.max()})
```



Abstract Syntax Tree (AST) Parsing

Library Imports

Imported modules
and aliases.

pandas

numpy as np

Function Calls

Method call chains.
DataFrame variables identified
by last return method call.

df_agg

df
.groupby
.agg
.reset_index

DataFrame Graphs

Modified & used DataFrames.
Parent & derived variables
with creation methods.

df_agg

df

df_agg
— (groupby) —
df

Schema Serialization

```
DATAFRAMES: orders, cleaned_orders
COLUMNS: Country (str), Customer Name (str), Delivery Year  (float64), Engine (str),
     Model Series (str), Order Month (str), Order Year (float64), Region (str),
    Delivery Total (float64), Order Total (float64), Unfilled Orders (float)

DATAFRAMES: df
COLUMNS: Order Total (float64)

DATAFRAMES: region
COLUMNS: region (str), year (float64)
```

- ❑ Column Types
- ❑ Aggregated Schemas

Prompting

```
You are a genius Python data science assistant.
Your task is to continue the notebook by answering the user question with the
    provided notebook context.
You must output Pandas Python code that will be parsed and executed in a stateful
    Jupyter notebook environment.
Think carefully about the DataFrames, columns, methods and use sound logical
    reasoning in your response.
You must output your answer in the requested format.
If no format is specified, your output should be a DataFrame or Series.
I will tip $10000000 if your code is clean and correct.
```

Listing 3.7: Default System Prompt

Prompting: Chain of Thought

- Detailed Instructions
- Message Tags
- Output Format

```
You are a genius AI programming data science expert.
Your goal is to output clear detailed steps such that a 5-year old can follow the
    steps to write Python code.
The current notebook code and markdown cells are provided in between the <
    notebook_context> tags.
Notebook variables with column information is provided in the <notebook_variables>
    tags.

<notebook_context>
...
</notebook_context>

<notebook_variables>
...
</notebook_variables>

My question is as follows:
<question>
...
</question>

For each natural language step, think carefully about the following:
1. All dataframes, series, column names, functions and variables should be in
    quotation marks.
2. Think carefully about column data types. Observe dataframe contents and data
    formats. Consider if parsing is required, such as dates. Ensure null/empty
    values are handled correctly.
3. Think carefully about every column in the dataframe and its relationship with the
     user's question.
4. Do not make assumptions. You have been given all context required. Ensure your
    steps match the question requirements.

Write your steps in between <scratchpad> tags.
Write your code in between <python> tags.
Do not output any other text.

Take a deep breath and work on this problem step-by-step.
```

Prompting: Chain of Thought

```
<scratchpad>
To find the average division population for different countries having events involving heavy rain, we need to follow these steps:
1. Extract the events involving heavy rain from the 'glc' dataframe.
2. Filter the events to only include those with a 'landslide_trigger' of 'Heavy Rain'.
3. Extract the 'country_name' and 'admin_division_population' columns from the filtered events.
4. Group the data by 'country_name' and calculate the average 'admin_division_population' for each country.
</scratchpad>
<python>
heavy_rain_events = glc[glc.landslide_trigger == 'Heavy Rain']
avg_division_population = heavy_rain_events.groupby('country_name') \
    ['admin_division_population'].mean()
print(avg_division_population)
</python>
```

Prompting: Multi-Step

REFLECTION + ERROR MESSAGES

```
##################
# Initial Prompt #
##################
[...]
Convert Levy and Mileage columns from object to integer.
[...]

##################
# Model Response #
##################

<scratchpad>
Step 1: Identify the columns that need to be converted from object to integer.
In this case, the columns are "Levy" and "Mileage".

Step 2: Remove any non-numeric characters from the "Levy" column.
We can use the "str.replace" method to remove any non-numeric characters.

Step 3: Convert the "Levy" column to integer.
We can use the "pd.to_numeric" function to convert the column to integer.

Step 4: Remove any non-numeric characters from the "Mileage" column.
We can use the "str.replace" method to remove any non-numeric characters.

Step 5: Convert the "Mileage" column to integer.
We can use the "pd.to_numeric" function to convert the column to integer.

Step 6: Assign the converted columns back to the original dataframe.
```

```
</scratchpad>
<python>
df['Levy'] = df['Levy'].str.replace('[^0-9]', '', regex=True).astype(int)
df['Mileage'] = df['Mileage'].str.replace('[^0-9]', '', regex=True).astype(int)
</python>

########################################
# Follow-Up Prompt with Error Feedback #
########################################
Your solution was executed in the stateful Jupyter notebook environment.
An error occurred during execution of the code you submitted:
<error>
RuntimeError: ValueError: invalid literal for int() with base 10: ''
</error>
[...]

############################
# Corrected Model Response #
############################
<analysis>
The error occurred because the code attempted to convert an empty string to an
    integer, which is not a valid operation. This is likely due to the presence of
    missing or empty values in the "Levy" or "Mileage" columns.

The question is asking to convert the "Levy" and "Mileage" columns from object type
    to integer type. However, the original code did not properly handle the
    possibility of missing or non-numeric values in these columns.

Upon re-examining the problem, it is clear that the code needs to handle these cases
     to ensure correctness. One way to do this is to use the "errors" parameter of
    the "pd.to_numeric" function to specify how to handle errors during the
    conversion process.
</analysis>
<python>
df['Levy'] = df['Levy'].str.replace('[^0-9]', '', regex=True).apply(pd.to_numeric,
    errors='coerce')
df['Mileage'] = df['Mileage'].str.replace('[^0-9]', '', regex=True).apply(pd.
    to_numeric, errors='coerce')
</python>
```

- ❏ Reflection
- ❏ Reflection + Error Messages
- ❏ Reflection + Solution Outputs

Experiment Setup

Model	No. Params	No. Training Tokens	Context Length
PaLM	62B	3.6T	2048
Llama 3 70B	70B	15T	8192
Llama 3 8B	8B	15T	8192

Table 4.1: Model Comparison

~~~~python
def extract_code_from_response(text):
    pattern = r'```(?:python\s+)?(.*?)(?:```|$)|<code>(.*?)</|<python>(.*?)(?:</|$)'
    match = re.search(pattern, text, re.IGNORECASE | re.DOTALL)
    if match:
        code_block = next(group for group in match.groups() if group)
        return code_block.strip(), text
    return text, text
~~~~

Listing 4.1: Code Extraction

$$\text{pass@k} := \mathbb{E}\left[1 - \left(\frac{C(n-c,k)}{C(n,k)}\right)\right]$$

Figure 2.10: The pass@k metric

Results: pass@5

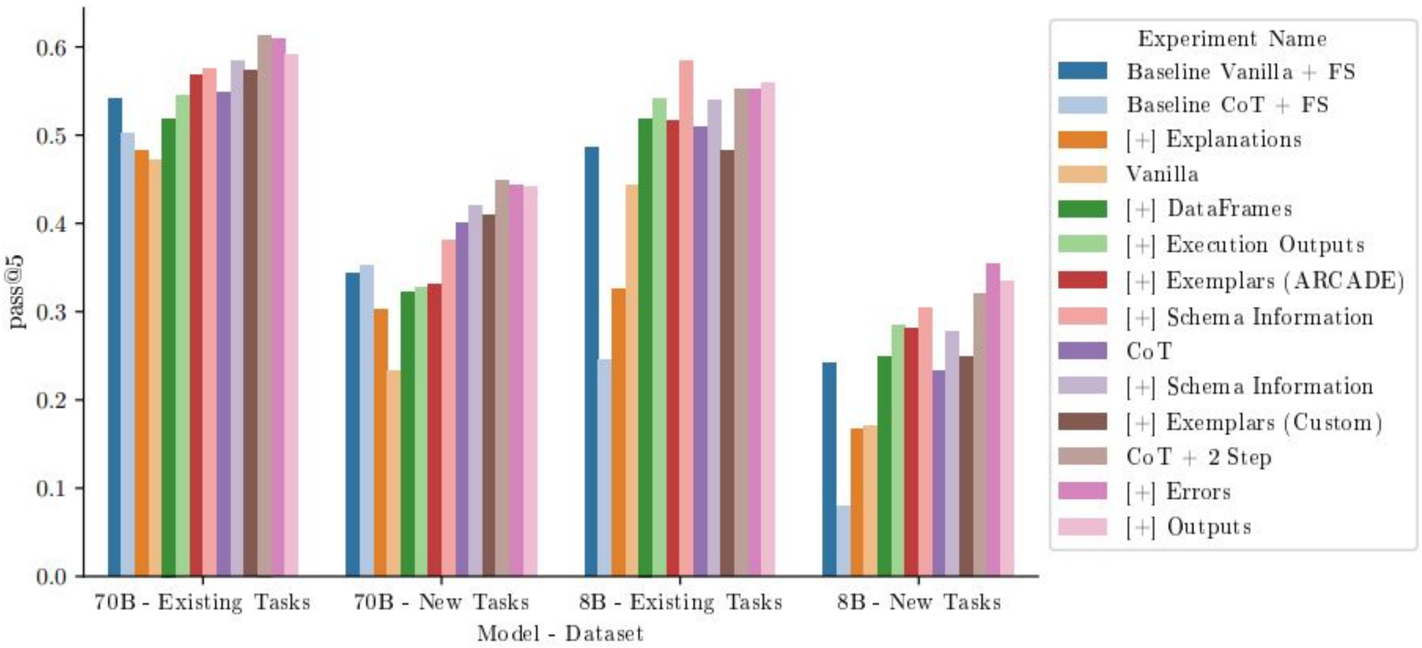


Figure 4.1: pass@5 evaluation results on the ARCADE dataset

Results: Errors

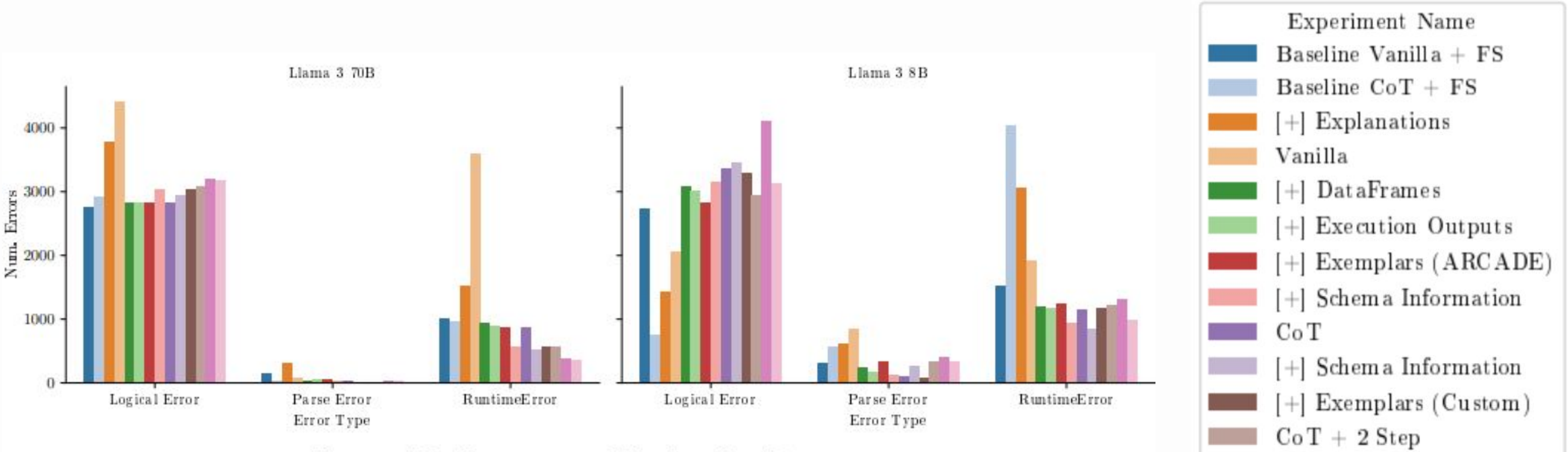


Figure 4.2: Frequency of Evaluation Errors

Ablation Studies

Table 4.4: pass@5 evaluation results comparing DataFrame placement

Experiment Name	Existing Tasks		New Tasks	
	Llama 3 70B	Llama 3 8B	Llama 3 70B	Llama 3 8B
CoT (Inline DataFrames)	**58.8**	**50.8**	**40.1**	23.3
CoT (Appended DataFrames)	56.5	50.6	37.2	**24.8**

Table 4.5: pass@5 evaluation results on the ARCADE dataset

Experiment Name	Existing Tasks		New Tasks	
	Llama 3 70B	Llama 3 8B	Llama 3 70B	Llama 3 8B
CoT (*)	**58.8**	50.8	40.1	23.3
+ Schema Information	58.4	**54.0**	**42.1**	**27.7**
CoT + Schema Information	55.0	49.8	39.3	23.6

(*) includes DataFrames and Execution Outputs

Conclusion

Table 5.1: pass@30 baseline comparisons on the New Tasks dataset

Experiment Name	New Tasks (pass@30)			
	PaLM 62B (†)	PaChiNCo 62B (‡)	Llama 3 70B	Llama 3 8B
Baseline Vanilla + FS	39.8	48.6	-	-
Baseline CoT + FS	-	52.9	-	-
CoT (*)	-	-	52.5	43.3
CoT (*) [Temp. Sampling]	-	-	**54.3**	42.7

(*) includes DataFrames, Execution Outputs and Schemas
(†) fine-tuned on Python
(‡) fine-tuned on Python and Notebooks

Future Work

Exemplars

Although adding exemplars to prompt has been shown to significantly boost results, our static crafted approach did not give good results.

Agentic Systems

Multi-step prompting naturally extends to agentic systems, unlocking potential for a more interactive, collaborative environment.