CAESURA: Language Models as Multi-Modal Query Planners

Matthias Urban
Technical University of Darmstadt
Carsten Binnig
Technical University of Darmstadt & DFKI

Motivation

Traditional query planners translate SQL queries into query plans to be executed over relational data.

- *a logical plan is first obtained from parsing a SQL query and then optimized by improving the order of the query’s operators.*
- *In a second step, each logical operator is mapped to a concrete implementation to obtain a physical plan, which is eventually executed.*

Query Planning in Traditional DBMS

This traditional approach to query planning is fundamentally limited in two important aspects:

Firstly, it only applies to query languages such as SQL, where semantics of queries are clear and can be easily parsed into a (at least canonical) query plan to execute the query.

Secondly, it is impossible to query other data modalities, such as images, text, or video stored in modern data systems such as data lakes using these query planners.

Trends in Data and Queries

Trend 1: Multi-Modal Data.

In today's industries, huge amounts of non-relational multi-modal data (e.g. images, documents, sensor data, ...) need to be stored and processed.

For instance, medical clinics need to store and analyze MRI scans and patient reports along with structured patient metadata.

Limitations of current works in multi-modal data:

- *limited in complexity (e.g. only simple queries with a single value as answer are supported*
- *or non-relational modalities are only used as filters*
- *or they are limited to only a very few modalities*

Trends in Data and Queries

Trend 2: *Natural Language Interfaces.*

Formulating complex queries in SQL requires profound knowledge of the language, making databases inaccessible to domain experts and management staff.

In recent years, Natural Language Interfaces for databases have emerged, which would allow laypersons to query databases using natural language. However, existing approaches typically translate natural language into SQL and are therefore limited to relational data.

Another direction is question-answering systems which work on modalities beyond tables. However, question-answering systems only support queries that are much less expressive than what can be done with SQL.

Vision Behind this Work

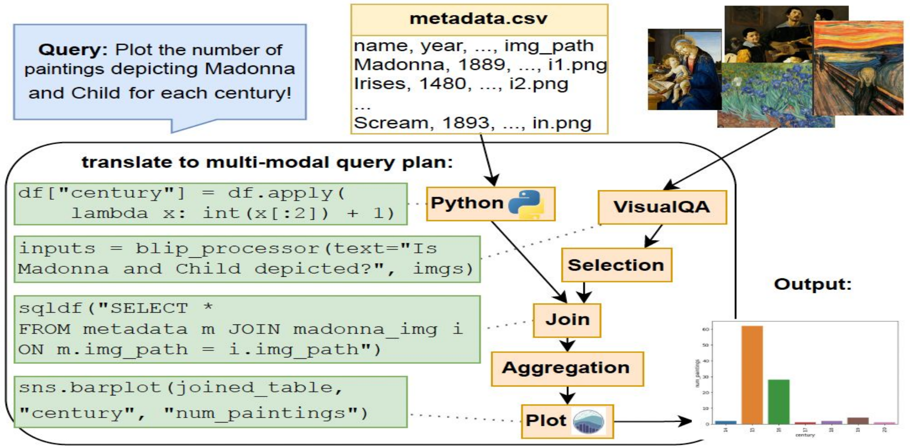
A vision of a data system that can be used by laypersons using natural language and can query arbitrary data modalities while enabling complex user queries way beyond classical SQL.

The result for user queries in Caesura can range from single values, over tables, to even a plot.

Example of a Multi-modal query

In the example, a user queries a data lake of a museum that stores both metadata (stored as a table) and pictures of artworks (stored as images) exhibited in the museum.

- *The Python operator extracts the century from a metadata column that stores the inception dates as strings and*
- *The VisualQA operator is used to select all pictures that depict Madonna and Child.*


Query: Plot the number of paintings depicting Madonna and Child for each century!
metadata.csv
name, year, ..., img_path
Madonna, 1889, ..., i1.png
Irises, 1480, ..., i2.png
...
Scream, 1893, ..., in.png
translate to multi-modal query plan:
Python
VisualQA
Selection
Join
Aggregation
Plot
Output:

```
df["century"] = df.apply(
    lambda x: int(x[:2]) + 1)
```

```
inputs = blip_processor(text="Is
Madonna and Child depicted?", imgs)
```

```
sqldf("SELECT *
FROM metadata m JOIN madonna_img i
ON m.img_path = i.img_path")
```

```
sns.barplot(joined_table,
"century", "num_paintings")
```

Figure 1: Example illustrating how a natural language query is automatically translated into a multi-modal query plan containing relational operators, machine generated Python UDFs, and a VisualQA Machine Learning model. The output is presented as a plot, making it easy and fast to gain insights from multi-modal data. The green boxes show code snippets that are executed when executing the plan.

Overview of Caesura

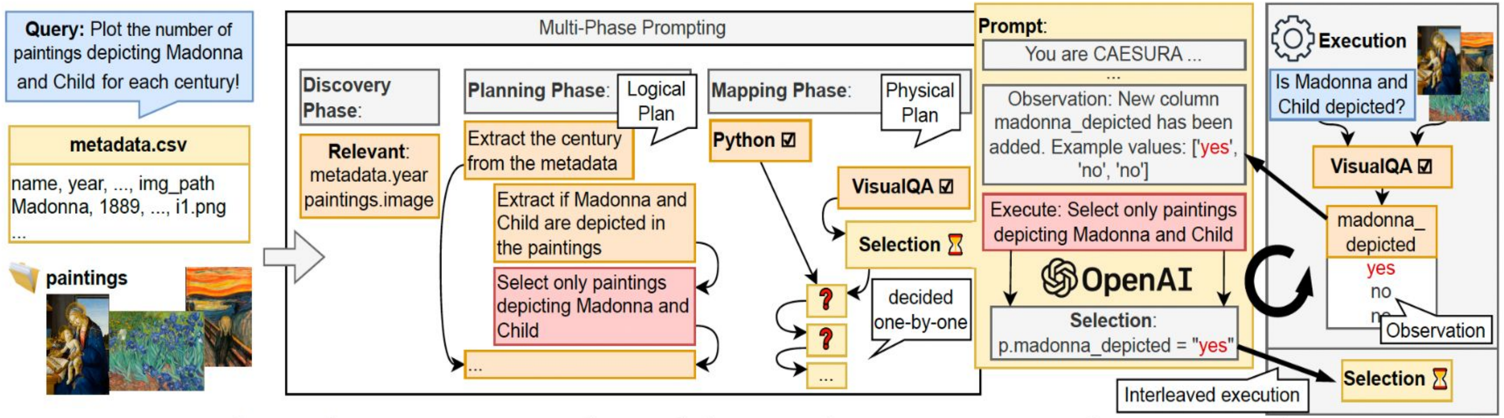

Query: Plot the number of paintings depicting Madonna and Child for each century!
metadata.csv
name, year, ..., img_path
Madonna, 1889, ..., i1.png
...
paintings
Multi-Phase Prompting
Discovery Phase:
Relevant: metadata.year paintings.image
Planning Phase:
Logical Plan
Extract the century from the metadata
Extract if Madonna and Child are depicted in the paintings
Select only paintings depicting Madonna and Child
...
Mapping Phase:
Physical Plan
Python ☑
VisualQA ☑
Selection
decided one-by-one
...
Prompt:
You are CAESURA ...
...
Observation: New column madonna_depicted has been added. Example values: ['yes', 'no', 'no']
Execute: Select only paintings depicting Madonna and Child
OpenAI
Selection: p.madonna_depicted = "yes"
Interleaved execution
Execution
Is Madonna and Child depicted?
VisualQA ☑
madonna_ depicted
yes
no
no
Observation
Selection

Figure 2: CAESURA transforms the query into a multi-modal query-plan using a series of prompts. In the *Discovery Phase*, the LLM is prompted to identify data items relevant for the query, such as relevant columns and datasets. In the *Planning Phase*, the LLM is prompted to construct a sequence of steps to satisfy the user request (Logical Plan). The final *Mapping Phase* is interleaved with *Execution*: a physical operator is chosen for each of the logical steps and executed incrementally. Once an operator is executed, we feed the results of the previous operator back to the LLM to choose the next operator (right). That allows the LLM to take the output of previous executions into account when choosing the physical operator and operator arguments (e.g. selection conditions as depicted in the figure) which avoids that faulty plans are generated.

Contribution

A Language-Model-Driven Query Planning, a new paradigm of query planning that uses Language Models to translate natural language queries into executable query plans.

Different from relational query planners, the resulting query plans can contain complex operators that are able to process arbitrary modalities.

A first GPT-4 based prototype called CAESURA and show the general feasibility of this idea on two datasets.

Mechanics of CAESURA

A new multi-phase compilation strategy,

- *leverages carefully designed prompts that contain all the necessary information about (1) the multi-modal data sources, (2) the available operators and (3) the query, which allows the LLM to come up with a query plan.*
- *is easy to plug in new operators (e.g. to process more modalities), as long as we provide all necessary information about their behavior in the prompt.*

- *based on the intuition of the phases of traditional query planning: first, by using a first prompt, the query planner maps the user query to a logical plan, containing a high-level step-by-step (textual) description of what needs to be done. Afterwards, using a separate prompt, each step is mapped to a concrete operator to obtain a physical plan that can be executed.*

Challenges

Plan Executability. *There are many causes why LLM-generated query plans might not be executable.*

- *For instance, the LLM might provide the wrong inputs to an operator (e.g. a collection of images*

as input for a traditional SQL selection).

Plan Correctness. *Even if a query plan produced by an LLM is executable, there might still be "logical flaws" in the plan, which can lead to wrong query results.*

- *For instance, important steps (e.g., a join) might be missing. This is especially challenging since there is no feedback from the LLM on whether an executed plan is correct or not.*

Challenges

Plan Optimization. *the plan generated by an LLM is not optimized, which is a problem since running non-optimized plans can result in huge runtime overheads. However, optimizing multi-modal query plans requires reasoning over the runtime of complex multi-modal operators, such as VisualQA or Python, which is non-trivial.*

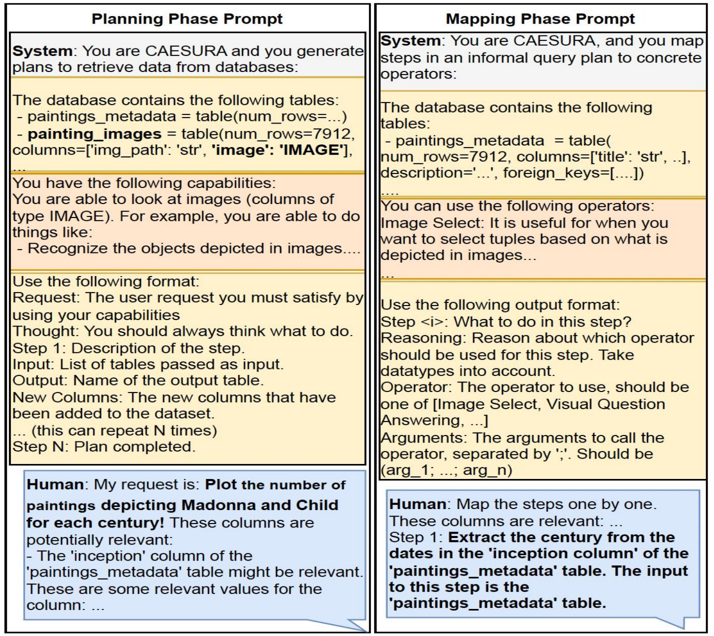
Phases of Query Planning in Caesura

- Identify the relevant data sources,
- In the planning phase, let the LLM generate the logical plan,
- And finally, in the mapping phase, let the LLM select the operators to obtain a physical plan.

Discovery Phase. *Decide which data sources (e.g., in a data lake) provide relevant information for the current Query.*
CAESURA first narrows down the relevant tables, image collections, etc. using dense retrieval.
Afterwards, for tabular data sources, we prompt the LLM to decide which columns of the retrieved data are relevant to the user query.
The identified relevant data items are used to construct prompts for the next phases, e.g., to present the LLM with some relevant example values that help it to generate correct selection conditions.

Phases of Query Planning (Planning Phase)

The LLM is prompted to come up with a logical query plan that contains a natural language description of all steps necessary to satisfy the user's request. Figure 3 (left) shows an example prompt
for this. The prompt consists of several parts: (1) a description of the data, (2) the capabilities of CAESURA, (3) an output format description, and (4) finally the user query and an instruction telling the model to come up with a plan.



Planning Phase Prompt

System: You are CAESURA and you generate plans to retrieve data from databases:

The database contains the following tables:
- paintings_metadata = table(num_rows=...)
- **painting_images** = table(num_rows=7912, columns=['img_path': 'str', **'image': 'IMAGE'**],
...

You have the following capabilities:
You are able to look at images (columns of type IMAGE). For example, you are able to do things like:
- Recognize the objects depicted in images....

Use the following format:
Request: The user request you must satisfy by using your capabilities
Thought: You should always think what to do.
Step 1: Description of the step.
Input: List of tables passed as input.
Output: Name of the output table.
New Columns: The new columns that have been added to the dataset.
... (this can repeat N times)
Step N: Plan completed.

Human: My request is: **Plot the number of paintings depicting Madonna and Child for each century!** These columns are potentially relevant:
- The 'inception' column of the 'paintings_metadata' table might be relevant. These are some relevant values for the column: ...

Mapping Phase Prompt

System: You are CAESURA, and you map steps in an informal query plan to concrete operators:

The database contains the following tables:
- paintings_metadata = table(num_rows=7912, columns=['title': 'str', ..], description='...', foreign_keys=[....])
....

You can use the following operators:
Image Select: It is useful for when you want to select tuples based on what is depicted in images...
...

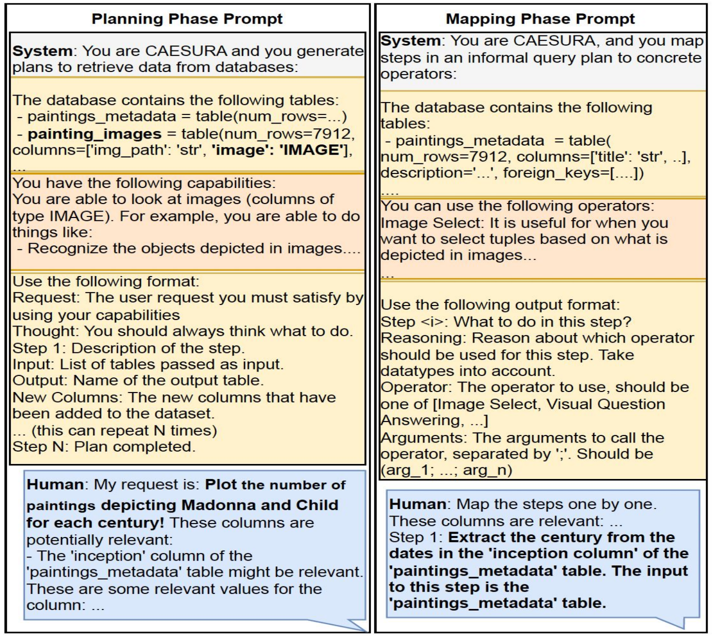
Use the following output format:
Step <i>: What to do in this step?
Reasoning: Reason about which operator should be used for this step. Take datatypes into account.
Operator: The operator to use, should be one of [Image Select, Visual Question Answering, ...]
Arguments: The arguments to call the operator, separated by ';'. Should be (arg_1; ...; arg_n)

Human: Map the steps one by one.
These columns are relevant: ...
Step 1: **Extract the century from the dates in the 'inception column' of the 'paintings_metadata' table. The input to this step is the 'paintings_metadata' table.**

Figure 3: Example prompts for the Planning and Mapping Phase. Each prompt consists of two messages and contains all relevant information, e.g. data descriptions, operator/capability descriptions, etc. as well as an instruction telling the LLM what to do. In the planning phase, we additionally utilize in-context learning and provide a few example translations from query to logical plan for different domains at the very beginning of the prompt (not depicted).

Phases of Query Planning
(Mapping Phase)

In the last phase, each previously determined logical step is mapped to a physical operator (and its input arguments) using a prompt similar to the one in Figure 3 (right). The prompt for this phase contains a short summary of the operators and what they can be used for. Moreover, different from traditional query planning, we do not decide on all the physical operators for all logical steps at once. Instead, we incrementally decide for each step and then execute it directly.



Planning Phase Prompt

System: You are CAESURA and you generate plans to retrieve data from databases:

The database contains the following tables:
- paintings_metadata = table(num_rows=...)
- **painting_images** = table(num_rows=7912, columns=['img_path': 'str', **'image': 'IMAGE'**],
...

You have the following capabilities:
You are able to look at images (columns of type IMAGE). For example, you are able to do things like:
- Recognize the objects depicted in images....

Use the following format:
Request: The user request you must satisfy by using your capabilities
Thought: You should always think what to do.
Step 1: Description of the step.
Input: List of tables passed as input.
Output: Name of the output table.
New Columns: The new columns that have been added to the dataset.
... (this can repeat N times)
Step N: Plan completed.

Human: My request is: **Plot the number of paintings depicting Madonna and Child for each century!** These columns are potentially relevant:
- The 'inception' column of the 'paintings_metadata' table might be relevant. These are some relevant values for the column: ...

Mapping Phase Prompt

System: You are CAESURA, and you map steps in an informal query plan to concrete operators:

The database contains the following tables:
- paintings_metadata = table(num_rows=7912, columns=['title': 'str', ..], description='...', foreign_keys=[....])
....

You can use the following operators:
Image Select: It is useful for when you want to select tuples based on what is depicted in images...
...

Use the following output format:
Step <i>: What to do in this step?
Reasoning: Reason about which operator should be used for this step. Take datatypes into account.
Operator: The operator to use, should be one of [Image Select, Visual Question Answering, ...]
Arguments: The arguments to call the operator, separated by ';'. Should be (arg_1; ...; arg_n)

Human: Map the steps one by one.
These columns are relevant: ...
Step 1: **Extract the century from the dates in the 'inception column' of the 'paintings_metadata' table. The input to this step is the 'paintings_metadata' table.**

Figure 3: Example prompts for the Planning and Mapping Phase. Each prompt consists of two messages and contains all relevant information, e.g. data descriptions, operator/capability descriptions, etc. as well as an instruction telling the LLM what to do. In the planning phase, we additionally utilize in-context learning and provide a few example translations from query to logical plan for different domains at the very beginning of the prompt (not depicted).

Error Handling - Part 1

Use the LLM for error handling

- *by adding the error message to the prompt and asking the LLM to fix the error.*

However, this comes with the challenge that the root cause of an error is not known. In particular, the root cause can also lie in any previously executed phase. For instance, in the planning phase, the LLM could have decided to filter by a non-existent column, but the mistake is only noticed after choosing an operator and executing the step.

Error Handling (Part-2)

To fix such errors, we thus use the LLM to identify in which phase the error occurred, backtrack to it, fix the error, and rerun the subsequent phases. For this purpose, we use an additional prompt containing a set of questions that encourage the LLM to reason about the error such as:
(1) What are the potential causes of this error?
(2) Explain in detail how this error could be fixed.
(3) Is there a flaw in my plan (Yes/No)?
(4) Is there a more suitable alternative plan (Yes/No)?
(5) Should a different tool be selected for any step (Yes/No)?
6) Do the input arguments of some of the steps need to be updated (Yes/No)?

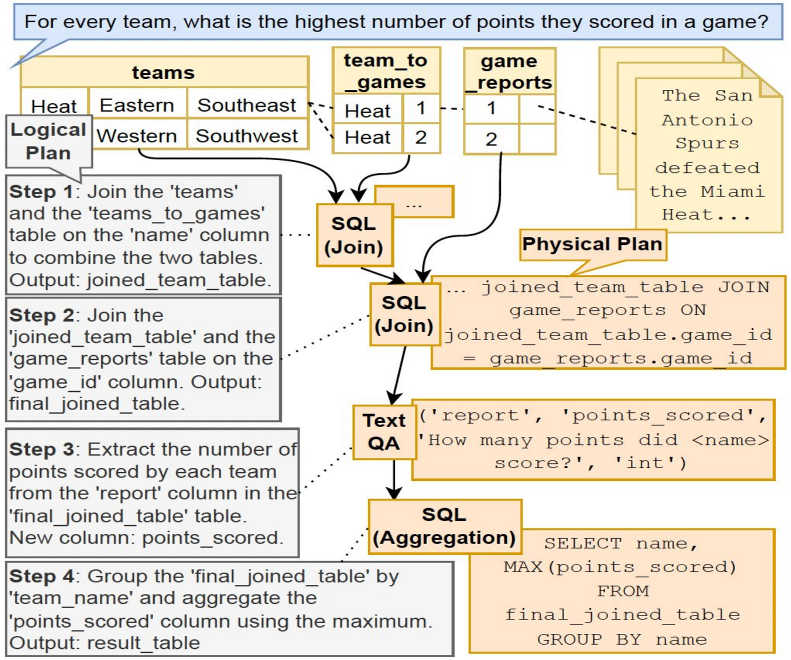
- *Parsing the responses to questions (3) + (4) allows us to determine whether to backtrack to the planning phase or if the mistake happened during the mapping phase.*
- *To finally fix the error, the ideas from questions (1) + (2) and the original error message are added to the prompt to which we backtracked to, before it (and potentially subsequent phases) is executed again.*

While this procedure allows CAESURA to fix non-executable plans in many cases, it clearly does not guarantee plan executability or even plan correctness.

Generated plans for the Rotowire Dataset

Figure 4: CAESURA using GPT-4 is able to correctly translate the user queries to multi-modal query plans that contain TextQA, VisualQA and Python operators.

The final physical plan, including input arguments for the operators, is shown in orange for both queries. For each operator, we also show the corresponding step of the GPT-4 generated logical plan (in grey).



Step 1: Join the 'teams' and the 'teams_to_games' table on the 'name' column to combine the two tables. Output: joined_team_table.

Step 2: Join the 'joined_team_table' and the 'game_reports' table on the 'game_id' column. Output: final_joined_table.

Step 3: Extract the number of points scored by each team from the 'report' column in the 'final_joined_table' table. New column: points_scored.

Step 4: Group the 'final_joined_table' by 'team_name' and aggregate the 'points_scored' column using the maximum. Output: result_table

```
... joined_team_table JOIN
game_reports ON
joined_team_table.game_id
= game_reports.game_id
```

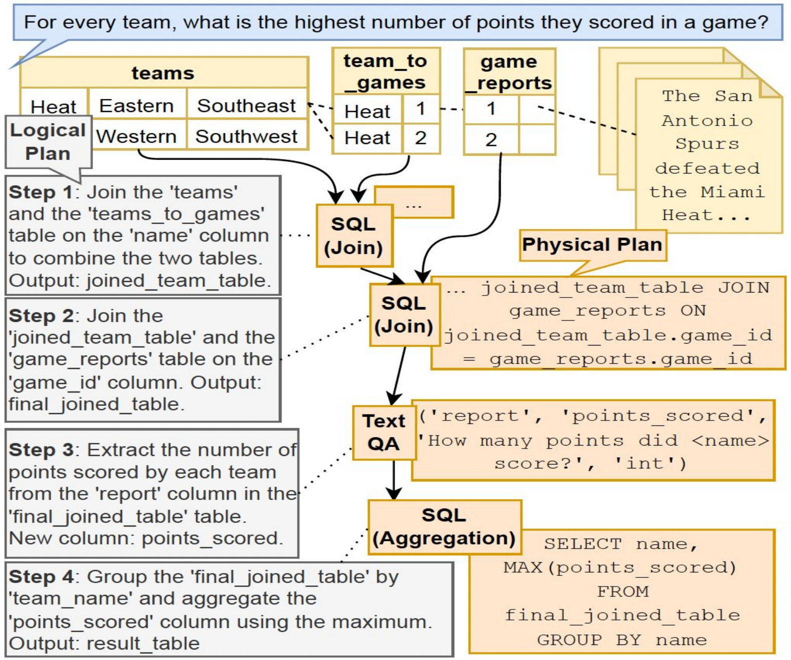
```
('report', 'points_scored',
'How many points did <name>
score?', 'int')
```

```
SELECT name,
MAX(points_scored)
FROM
final_joined_table
GROUP BY name
```

Generated plans for the Rotowire Dataset

CAESURA presents image and text collections as special tables (game_reports with columns game_id and report; painting_images with columns img_path and image) to the LLM, s.t. they can be the input to a regular join.

The TextQA operator takes a question template as input, which is translated to questions by inserting different team names from the values in the table. The Python operator takes a description as input, which is translated to code using GPT-4.



Step 1: Join the 'teams' and the 'teams_to_games' table on the 'name' column to combine the two tables. Output: joined_team_table.

Step 2: Join the 'joined_team_table' and the 'game_reports' table on the 'game_id' column. Output: final_joined_table.

Step 3: Extract the number of points scored by each team from the 'report' column in the 'final_joined_table' table. New column: points_scored.

Step 4: Group the 'final_joined_table' by 'team_name' and aggregate the 'points_scored' column using the maximum. Output: result_table

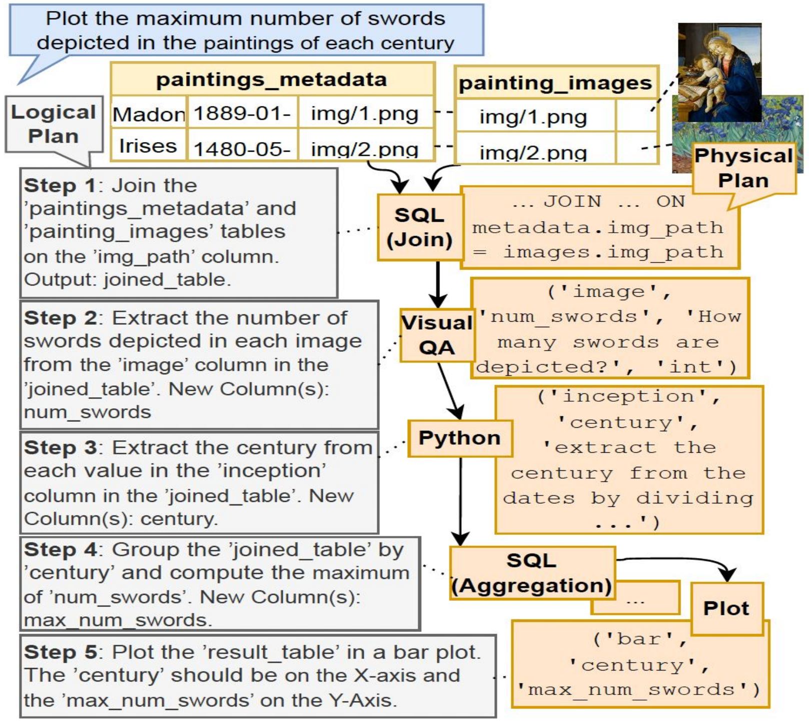
```
... joined_team_table JOIN
game_reports ON
joined_team_table.game_id
= game_reports.game_id
```

```
('report', 'points_scored',
'How many points did <name>
score?', 'int')
```

```
SELECT name,
MAX(points_scored)
FROM
final_joined_table
GROUP BY name
```

Generated plans for the Artwork Dataset

- *requires the inspection of the images, as well as the visualization of the results in the end.*
- *CAESURA is able to correctly translate this query using two multi-modal operators: Python and VisualQA followed by a plot operator.*



Plot the maximum number of swords depicted in the paintings of each century

paintings_metadata		
Madon	1889-01-	img/1.png
Irises	1480-05-	img/2.png

painting_images	
img/1.png	
img/2.png	

Step 1: Join the 'paintings_metadata' and 'painting_images' tables on the 'img_path' column. Output: joined_table.

Step 2: Extract the number of swords depicted in each image from the 'image' column in the 'joined_table'. New Column(s): num_swords

Step 3: Extract the century from each value in the 'inception' column in the 'joined_table'. New Column(s): century.

Step 4: Group the 'joined_table' by 'century' and compute the maximum of 'num_swords'. New Column(s): max_num_swords.

Step 5: Plot the 'result_table' in a bar plot. The 'century' should be on the X-axis and the 'max_num_swords' on the Y-Axis.

```
... JOIN ... ON
metadata.img_path
= images.img_path
```

```
('image',
'num_swords', 'How
many swords are
depicted?', 'int')
```

```
('inception',
'century',
'extract the
century from the
dates by dividing
...')
```

```
...
```

```
('bar',
'century',
'max_num_swords')
```

Access to Operators

prototype of CAESURA has access to four multi-modal operators:

(1) VisualQA based on BLIP-2
(2) TextQA based on BART
(3) Python UDFs, and
(4) Image Select, which selects images based on a description and is also based on BLIP-2.

It also has access to all relational operators supported by SQLite and a plotting operator based on seaborn.

Experiments

Two multi-modal datasets.

(1) The artwork dataset (with tables and images),

- *the dataset contains a table about painting metadata as well as an image collection containing images of the artworks.*
- *Wikidata to construct both the metadata* table as well as the image corpus:
- for the metadata table, extract title, inception, movement, etc. for all Wikidata entities that are instances of ’painting’ were extracted

Experiments

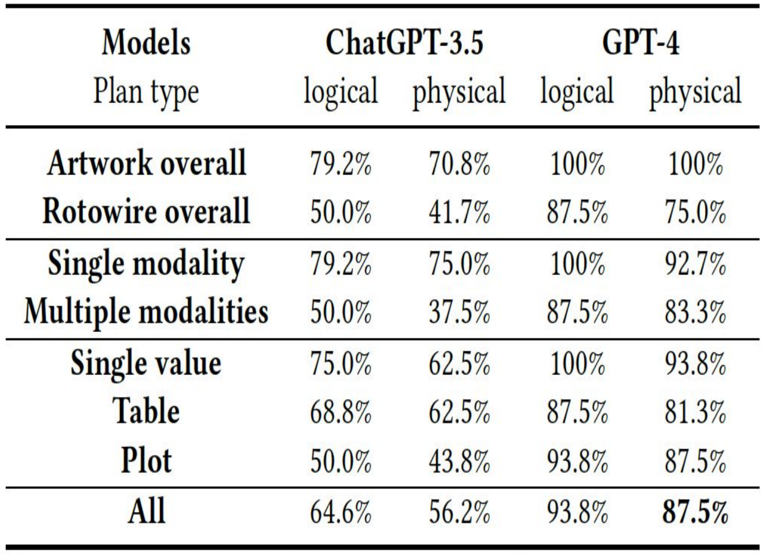
Two multi-modal datasets.

(2) The rotowire dataset (with tables and text) which consists of textual game reports of
basketball games, containing important statistics (e.g. the number of scored points) of players and teams that participated in each game.

- Extended the textual reports by two tables for players and teams constructed from Wikidata.
    - These contain general information, such as name, conference, division, etc. for every team, and name, height, nationality, etc. for every player

Evaluation of Plan Qualities

- Evaluated CAESURA on a larger set of queries, 24 for each dataset.
- 16 queries asking for a single result value,
- 16 that ask for an output table, and
- 16 that ask for a plot.
- Half of the queries require multi-modal data while the other half require only relational data.



Models	ChatGPT-3.5		GPT-4	
Plan type	logical	physical	logical	physical
Artwork overall	79.2%	70.8%	100%	100%
Rotowire overall	50.0%	41.7%	87.5%	75.0%
Single modality	79.2%	75.0%	100%	92.7%
Multiple modalities	50.0%	37.5%	87.5%	83.3%
Single value	75.0%	62.5%	100%	93.8%
Table	68.8%	62.5%	87.5%	81.3%
Plot	50.0%	43.8%	93.8%	87.5%
All	64.6%	56.2%	93.8%	**87.5%**

Table 1: Correctly translated plans for the different datasets, modalities, and output formats. We show the percentage of correctly generated logical plans, as well as physical plans.

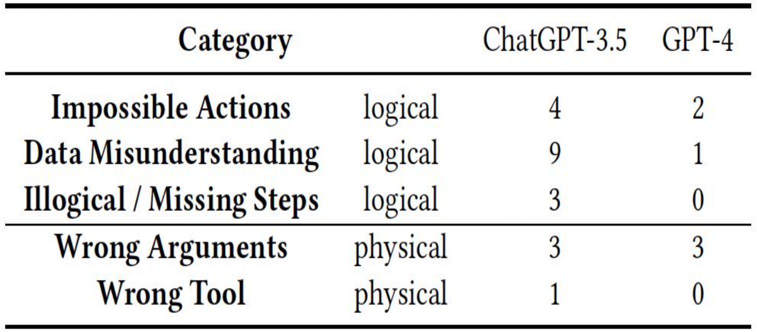
Error Analysis

CAESURA powered by GPT-4

- *it chose the wrong input arguments for physical operators (e.g., wrong parameters for SQL, a wrong question for QA, usage of non-existent column names).*

ChatGPT-3.5 often tried to

- *extract what is depicted in the image based on the title or the genre column of the metadata table. Thus, it often avoided the usage of multi-modal operators and instead tried to solve everything using SQL, resulting in flawed plans.*



Category		ChatGPT-3.5	GPT-4
Impossible Actions	logical	4	2
Data Misunderstanding	logical	9	1
Illogical / Missing Steps	logical	3	0
Wrong Arguments	physical	3	3
Wrong Tool	physical	1	0

Table 2: Number of specific kinds of mistakes CAESURA made during query planning. In the upper three categories the mistake occurred in the planning phase (i.e. wrong logical plan), and for the lower two the mistake occurred in the mapping phase. We see that the older model often does not understand the data correctly (e.g. it tries to determine what is depicted on a painting based on its title).