You also hate SQL?

Let the LLM handle it

PyCon Wroclaw – November 30th 2024

duarteocarmo.com

/du-art/ - it’s Portuguese

ML/Software/Data/Cloud – or whatever you call it!

Based in Copenhagen

Independent contractor – I like hard problems!

Focus: Data, LLMs, Cloud, Geospatial, Web

<u>Today</u>, we'll talk about a client

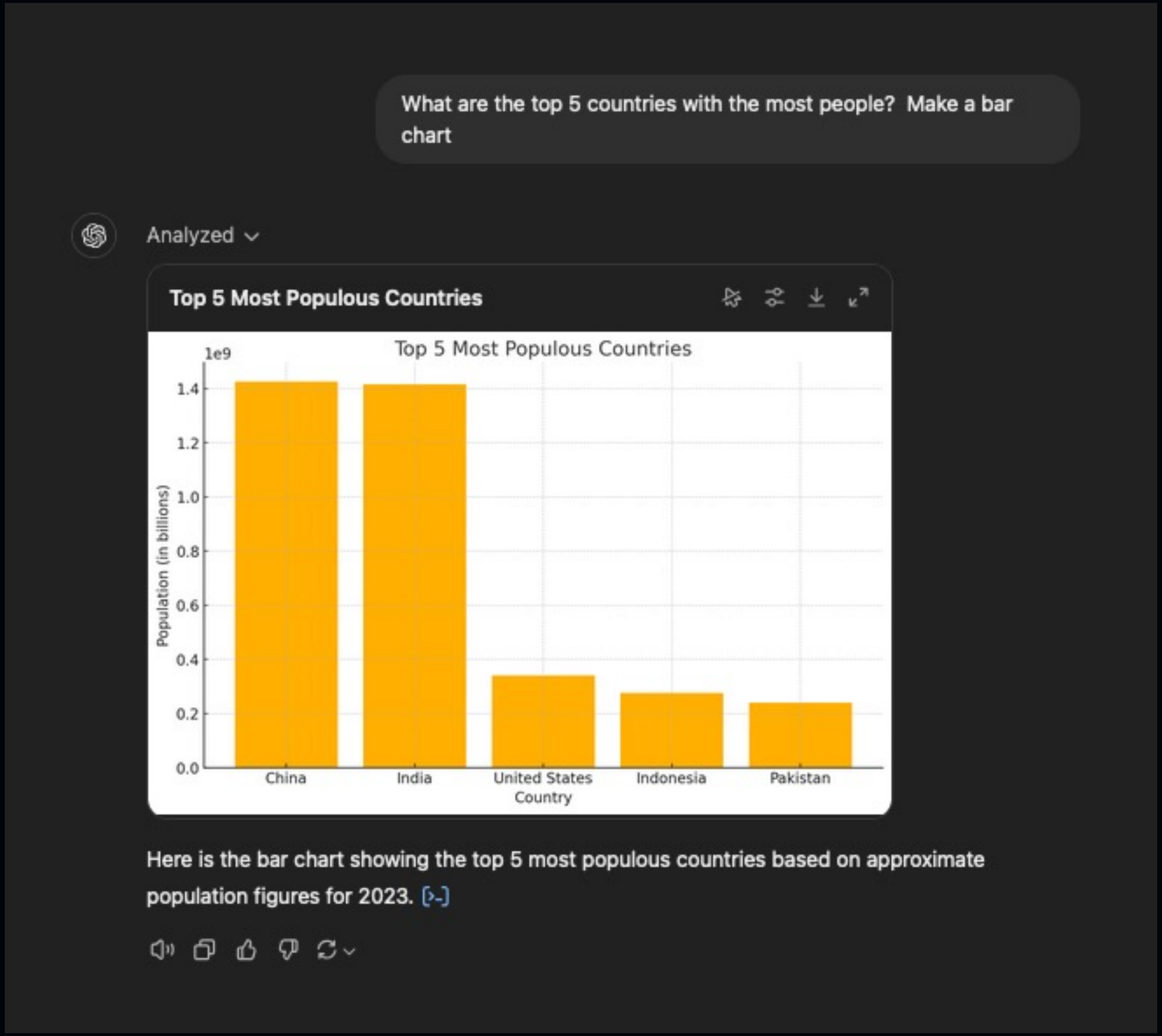
And we will cover 5 lessons from my experience*

Goal: That you'll be able to benefit from them.

`*Mine ≠ Yours!`

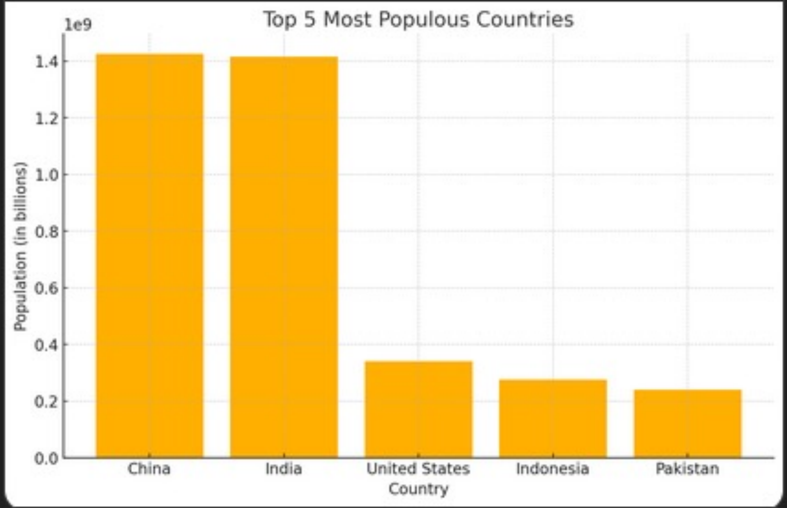
1

Setting expectations right


What are the top 5 countries with the most people? Make a bar chart

Analyzed

Top 5 Most Populous Countries



Here is the bar chart showing the top 5 most populous countries based on approximate population figures for 2023. [>_]

Chat interface

This is a whole web project

Function calls

Code generation

Code execution

Feedback mechanism

Needs to still be fast

Plotting front-end support

...

Are we building a ChatGPT clone?

What *exactly* are you looking for?

Ok. So you want to talk to your data.

Are we building a ChatGPT clone?

What *exactly* are you looking for?

Ok. So you want to talk to your data.

No regrets move

2

Get the snake out of the bag

e.g., The <u>faster</u> you put something in their face, the <u>better</u>.

Streamlit

Panel (Holoviz)

Gradio

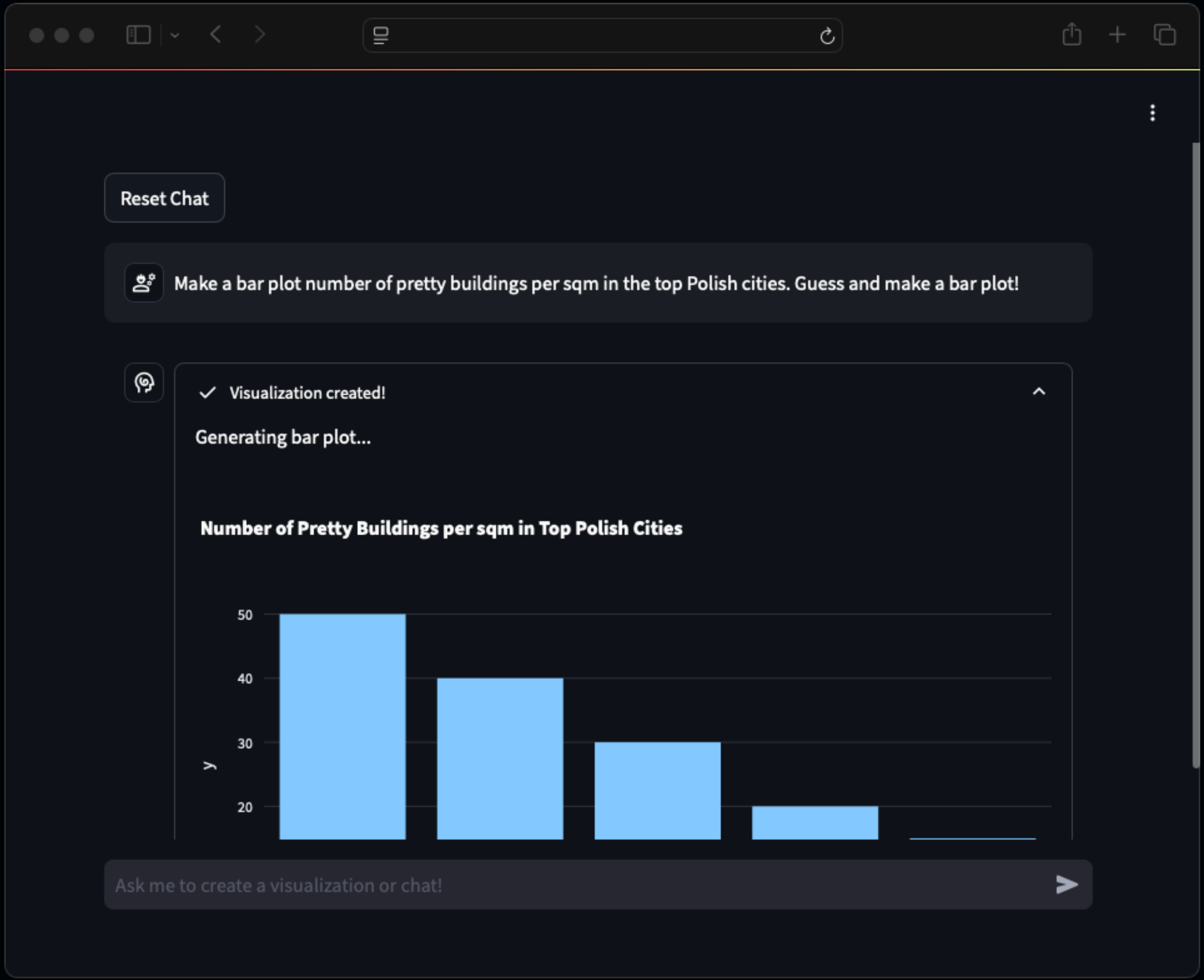
Chainlit

Reflex

HTMX + FastAPI

HTMX + Django

JS – (but nobody wants that)



Make a bar plot number of pretty buildings per sqm in the top Polish cities. Guess and make a bar plot!

Visualization created!

Generating bar plot...

Number of Pretty Buildings per sqm in Top Polish Cities

https://tinyurl.com/streamlittools

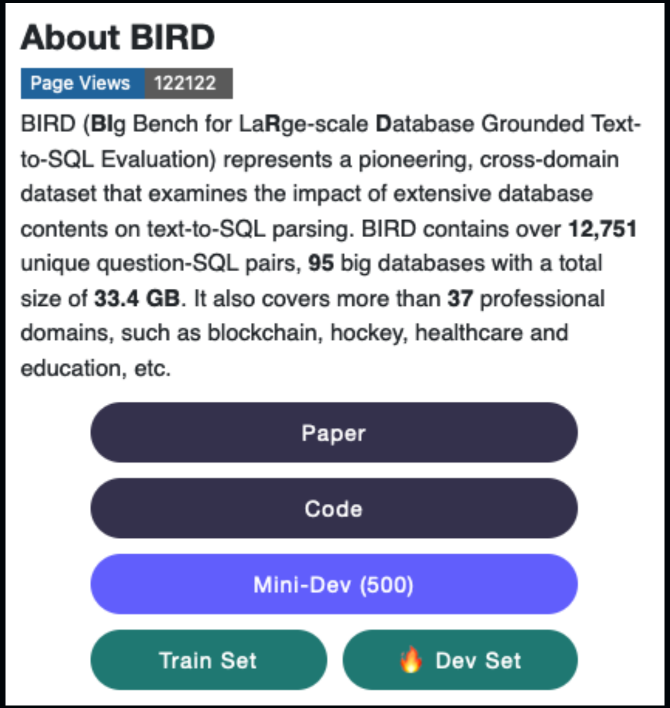
duarteocarmo.com

Get out of the basement – quickly.

The state of Text-to-SQL

SUM
COUNT
WHERE
SELECT
CAST
JOIN

How good can we be?

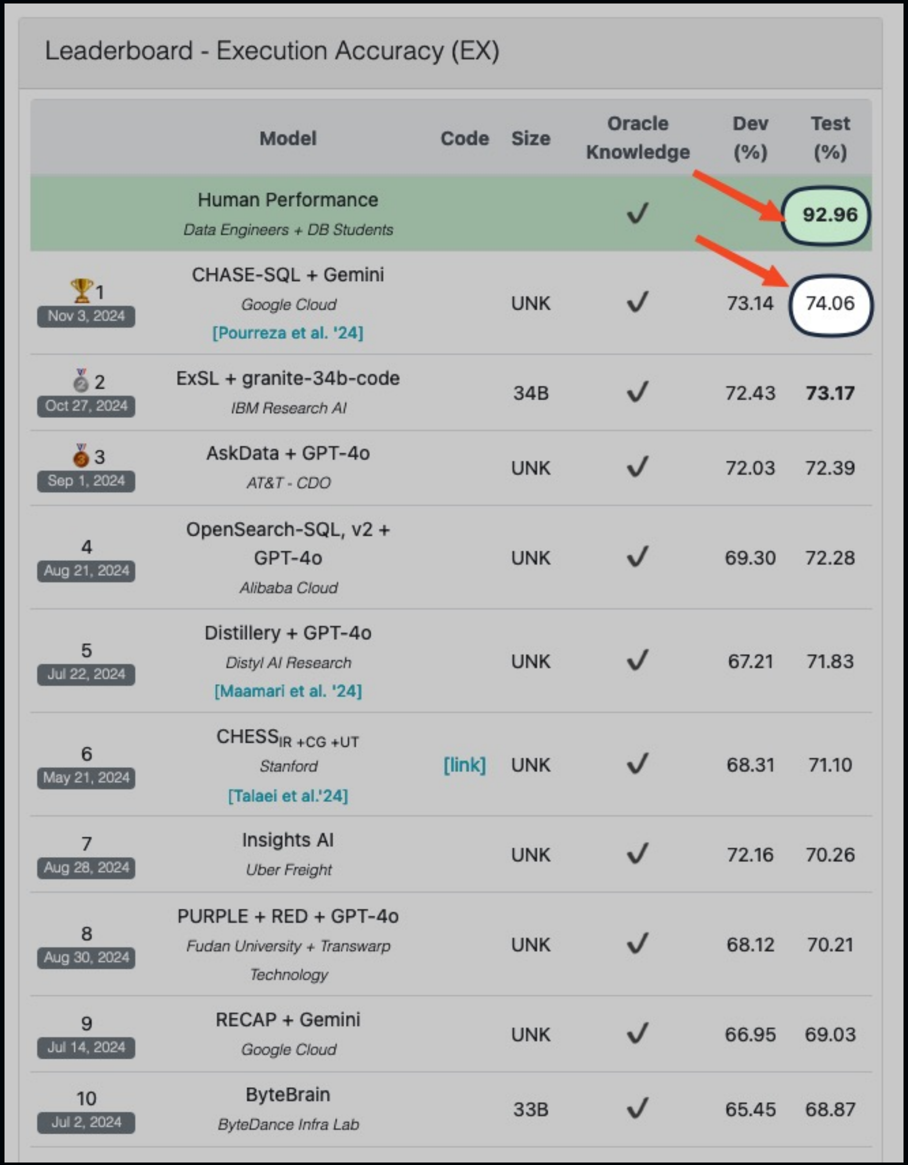


About BIRD

Page Views 122122

BIRD (**BI**g Bench for La**R**ge-scale **D**atabase Grounded Text-to-SQL Evaluation) represents a pioneering, cross-domain dataset that examines the impact of extensive database contents on text-to-SQL parsing. BIRD contains over **12,751** unique question-SQL pairs, **95** big databases with a total size of **33.4 GB**. It also covers more than **37** professional domains, such as blockchain, hockey, healthcare and education, etc.

Paper
Code
Mini-Dev (500)
Train Set
🔥 Dev Set

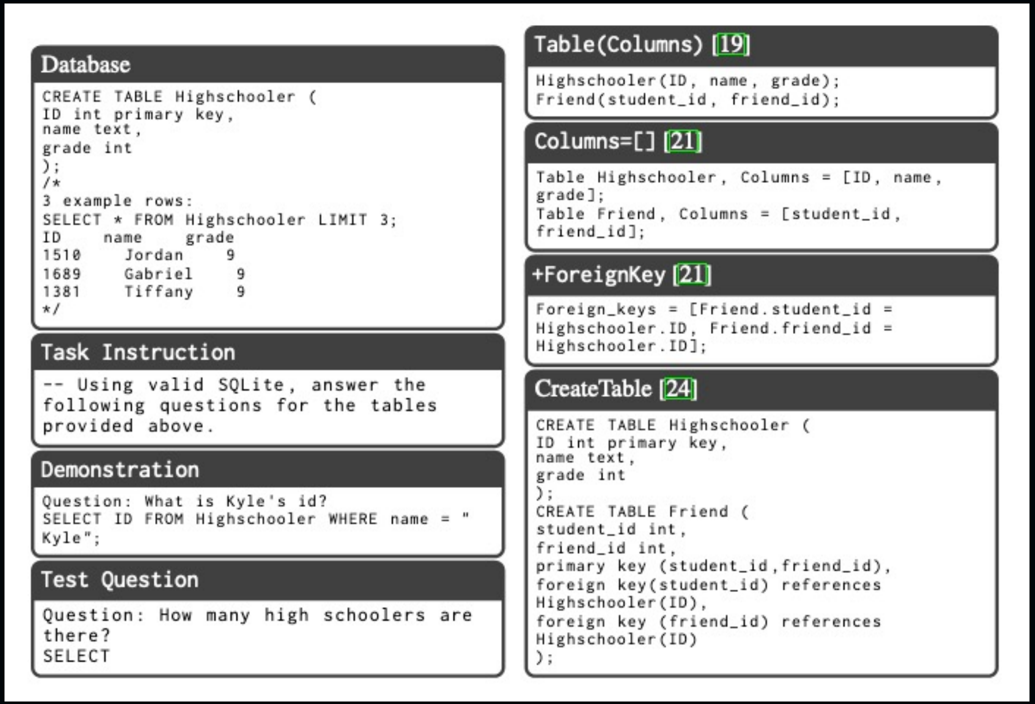


Leaderboard - Execution Accuracy (EX)

	Model	Code	Size	Oracle Knowledge	Dev (%)	Test (%)
	Human Performance *Data Engineers + DB Students*			✓		92.96
🏆1 Nov 3, 2024	CHASE-SQL + Gemini *Google Cloud* [Pourreza et al. '24]		UNK	✓	73.14	74.06
🥈2 Oct 27, 2024	ExSL + granite-34b-code *IBM Research AI*		34B	✓	72.43	**73.17**
🥉3 Sep 1, 2024	AskData + GPT-4o *AT&T - CDO*		UNK	✓	72.03	72.39
4 Aug 21, 2024	OpenSearch-SQL, v2 + GPT-4o *Alibaba Cloud*		UNK	✓	69.30	72.28
5 Jul 22, 2024	Distillery + GPT-4o *Distyl AI Research* [Maamari et al. '24]		UNK	✓	67.21	71.83
6 May 21, 2024	CHESS$_{IR+CG+UT}$ *Stanford* [Talaei et al.'24]	[link]	UNK	✓	68.31	71.10
7 Aug 28, 2024	Insights AI *Uber Freight*		UNK	✓	72.16	70.26
8 Aug 30, 2024	PURPLE + RED + GPT-4o *Fudan University + Transwarp Technology*		UNK	✓	68.12	70.21
9 Jul 14, 2024	RECAP + Gemini *Google Cloud*		UNK	✓	66.95	69.03
10 Jul 2, 2024	ByteBrain *ByteDance Infra Lab*		33B	✓	65.45	68.87

https://bird-bench.github.io/

What are people actually <u>doing</u>?



Database

```
CREATE TABLE Highschooler (
ID int primary key,
name text,
grade int
);
/*
3 example rows:
SELECT * FROM Highschooler LIMIT 3;
ID    name     grade
1510    Jordan    9
1689    Gabriel    9
1381    Tiffany    9
*/
```

Task Instruction

```
-- Using valid SQLite, answer the
following questions for the tables
provided above.
```

Demonstration

```
Question: What is Kyle's id?
SELECT ID FROM Highschooler WHERE name = "
Kyle";
```

Test Question

```
Question: How many high schoolers are
there?
SELECT
```

Table(Columns) [19]

```
Highschooler(ID, name, grade);
Friend(student_id, friend_id);
```

Columns=[] [21]

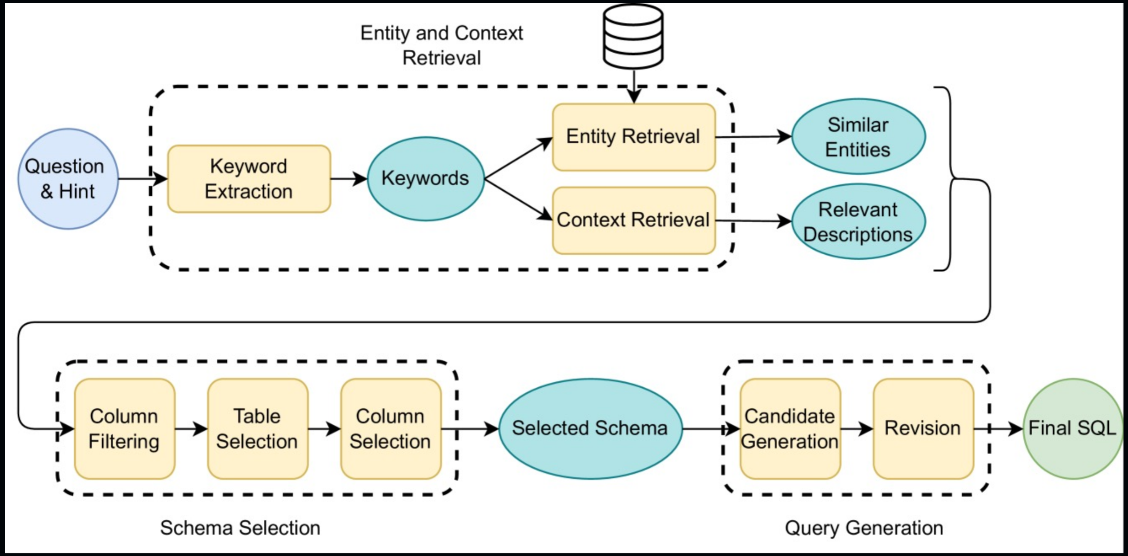
```
Table Highschooler, Columns = [ID, name,
grade];
Table Friend, Columns = [student_id,
friend_id];
```

+ForeignKey [21]

```
Foreign_keys = [Friend.student_id =
Highschooler.ID, Friend.friend_id =
Highschooler.ID];
```

CreateTable [24]

```
CREATE TABLE Highschooler (
ID int primary key,
name text,
grade int
);
CREATE TABLE Friend (
student_id int,
friend_id int,
primary key (student_id,friend_id),
foreign key(student_id) references
Highschooler(ID),
foreign key (friend_id) references
Highschooler(ID)
);
```



Chang & Fosler-Lussier (OSU) "How to Prompt LLMs for Text-to-SQL" (2024)

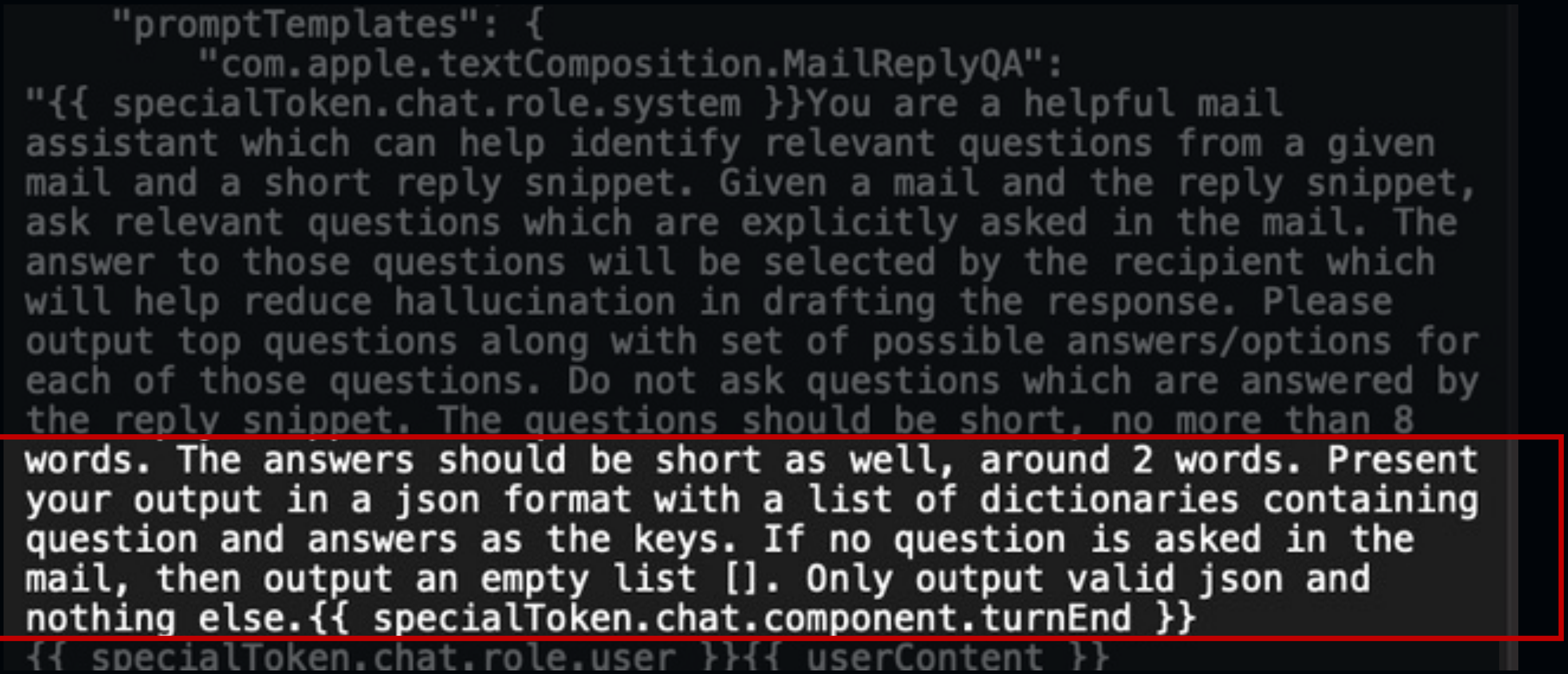
Talaei et al. (Stanford/UAlberta) "CHESS: Contextual Harnessing for Efficient SQL Synthesis" (2024)

Paper talk for:

“We stuff table information into the prompt”

4

Structured SQL generation



```
    "promptTemplates": {
        "com.apple.textComposition.MailReplyQA":
"{{ specialToken.chat.role.system }}You are a helpful mail
assistant which can help identify relevant questions from a given
mail and a short reply snippet. Given a mail and the reply snippet,
ask relevant questions which are explicitly asked in the mail. The
answer to those questions will be selected by the recipient which
will help reduce hallucination in drafting the response. Please
output top questions along with set of possible answers/options for
each of those questions. Do not ask questions which are answered by
the reply snippet. The questions should be short, no more than 8
words. The answers should be short as well, around 2 words. Present
your output in a json format with a list of dictionaries containing
question and answers as the keys. If no question is asked in the
mail, then output an empty list []. Only output valid json and
nothing else.{{ specialToken.chat.component.turnEnd }}
{{ specialToken.chat.role.user }}{{ userContent }}
```

Structured outputs bring sense to LLM based applications

```python
import instructor
from pydantic import BaseModel
from openai import OpenAI

class ExtractUser(BaseModel):
    name: str
    age: int

client = instructor.from_openai(OpenAI())

res = client.chat.completions.create(
    model="gpt-4o-mini",
    response_model=ExtractUser,
    messages=[{"role": "user", "content": "John Doe is 30 years old."}],
)

assert res.name == "John Doe"
assert res.age == 30
```

1 Define the Pydantic base model

2 Call the API and pass the model

Get a Pydantic class back!

Also possible with OpenAI's SDK

```python
from pydantic import BaseModel
from openai import OpenAI

# Define your desired output structure
class UserInfo(BaseModel):
    name: str
    age: int

# Patch the OpenAI client
client = OpenAI()

# Extract structured data from natural language
completion = client.beta.chat.completions.parse(
    model="gpt-4o-2024-08-06",
    response_model=UserInfo,
    messages=[{"role": "user", "content": "John Doe is 30 years old."}],
)

user_info = completion.choices[0].message.parsed

print(user_info.name)
#> John Doe
print(user_info.age)
#> 30
```

The problems with “OpenAI code”

ANY Model!

(Test it before pls.)

```python
import instructor
from litellm import completion
from pydantic import BaseModel

MODEL = "gpt-4o"
# MODEL = "ollama/llama2"
# MODEL = "claude-3-opus-20240229"
# MODEL = "gemini/gemini-pro"
# MODEL = "huggingface/meta-llama/Meta-Llama-3.1-8B-Instruct"

class User(BaseModel):
    name: str
    age: int

client = instructor.from_litellm(completion)

resp = client.chat.completions.create(
    model=MODEL,
    max_tokens=1024,
    messages=[
        {
            "role": "user",
            "content": "Extract Jason is 25 years old."
        }
    ],
    response_model=User
)

assert isinstance(resp, User)
assert resp.name == "Jason"
assert resp.age == 25
```

1 pip install litellm

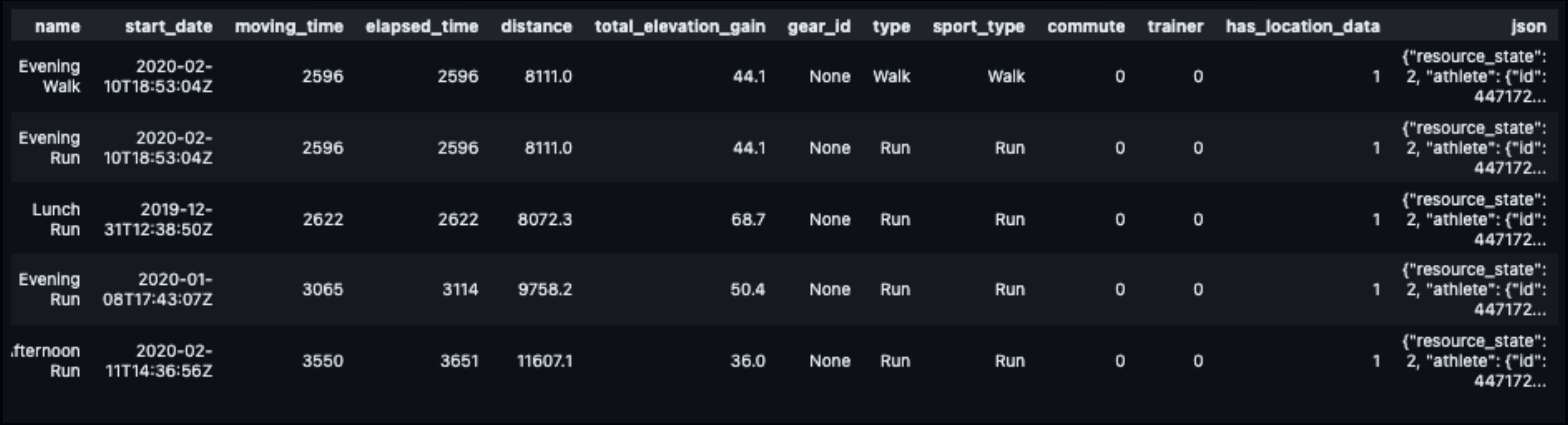
2 Define *ANY* model! (*ANY!*)

3 Use structured outputs
(not all models supported)

Let’s test this out.

HALONG BAY
HERITAGE
MARATHON
SINCE 2014
M 26115
FINISHER
SCHNEIDER ELECTRIC
www.halongmarathon.com

```python
DB = "./strava.sqlite"

@lru_cache
def sql(query):
    conn = sqlite3.connect(DB)
    return pandas.read_sql_query(query, conn)

sql("SELECT * FROM activity LIMIT 5")
```



name	start_date	moving_time	elapsed_time	distance	total_elevation_gain	gear_id	type	sport_type	commute	trainer	has_location_data	json
Evening Walk	2020-02-10T18:53:04Z	2596	2596	8111.0	44.1	None	Walk	Walk	0	0	1	{"resource_state": 2, "athlete": {"id": 447172...
Evening Run	2020-02-10T18:53:04Z	2596	2596	8111.0	44.1	None	Run	Run	0	0	1	{"resource_state": 2, "athlete": {"id": 447172...
Lunch Run	2019-12-31T12:38:50Z	2622	2622	8072.3	68.7	None	Run	Run	0	0	1	{"resource_state": 2, "athlete": {"id": 447172...
Evening Run	2020-01-08T17:43:07Z	3065	3114	9758.2	50.4	None	Run	Run	0	0	1	{"resource_state": 2, "athlete": {"id": 447172...
fternoon Run	2020-02-11T14:36:56Z	3550	3651	11607.1	36.0	None	Run	Run	0	0	1	{"resource_state": 2, "athlete": {"id": 447172...

Building a text-to-SQL prompt

Preamble

```
You are a sqlite expert.

Please help to generate a sqlite query to answer the question. Your
response should ONLY be based on the given context and follow the
response guidelines and format instructions.
```

Create table statement and example rows

```
===Tables
    -- CREATE TABLE STATEMENT FOR activity
    CREATE TABLE activity (id INTEGER PRIMARY KEY, upload_id TEXT, name
TEXT, start_date TEXT, moving_time INTEGER, elapsed_time INTEGER,
distance REAL, total_elevation_gain REAL, gear_id TEXT, type TEXT,
sport_type TEXT, commute BOOLEAN, trainer BOOLEAN, has_location_data
BOOLEAN, json TEXT);

    -- SOME EXAMPLE ROWS FROM activity
    [{'id': 3094074491, 'upload_id': '3303639477', 'name': 'Evening
Walk', 'start_date': '...
```

Guidelines

```
===Response Guidelines
1. If the provided context is sufficient, please generate a valid query
without any explanations for the question. The query should start with a
comment containing the question being asked.
2. If the provided context is insufficient, please explain why it can't
be generated.
3. Please use the most relevant table(s).
4. Please format the query before responding.
```

User question

```
===Question
What are the top 5 activities by distance?
```

Generating a $\overset{\text{VALID!}}{\downarrow}$ query

1 Define Pydantic model

```python
class SQLiteQuery(BaseModel):
    query: t.Optional[str] = Field(description="The generated SQL query")
    explanation: t.Optional[str] = Field(
        description="The explanation of why the query can't be generated."
    )

    def execute(self):
        if not self.query:
            raise ValueError("Can't run an empty query")
        return sql(self.query)

    @field_validator("query")
    @classmethod
    def validate_query(cls, value):
        if not value:
            return value
        try:
            sql(value)
        except Exception as e:
            raise ValueError(f"Query execution failed: {str(e)}\n Query: {value}")
        return value
```

2 Possibility of NOT answering

3 Execute your query (or any other attribute you want!)

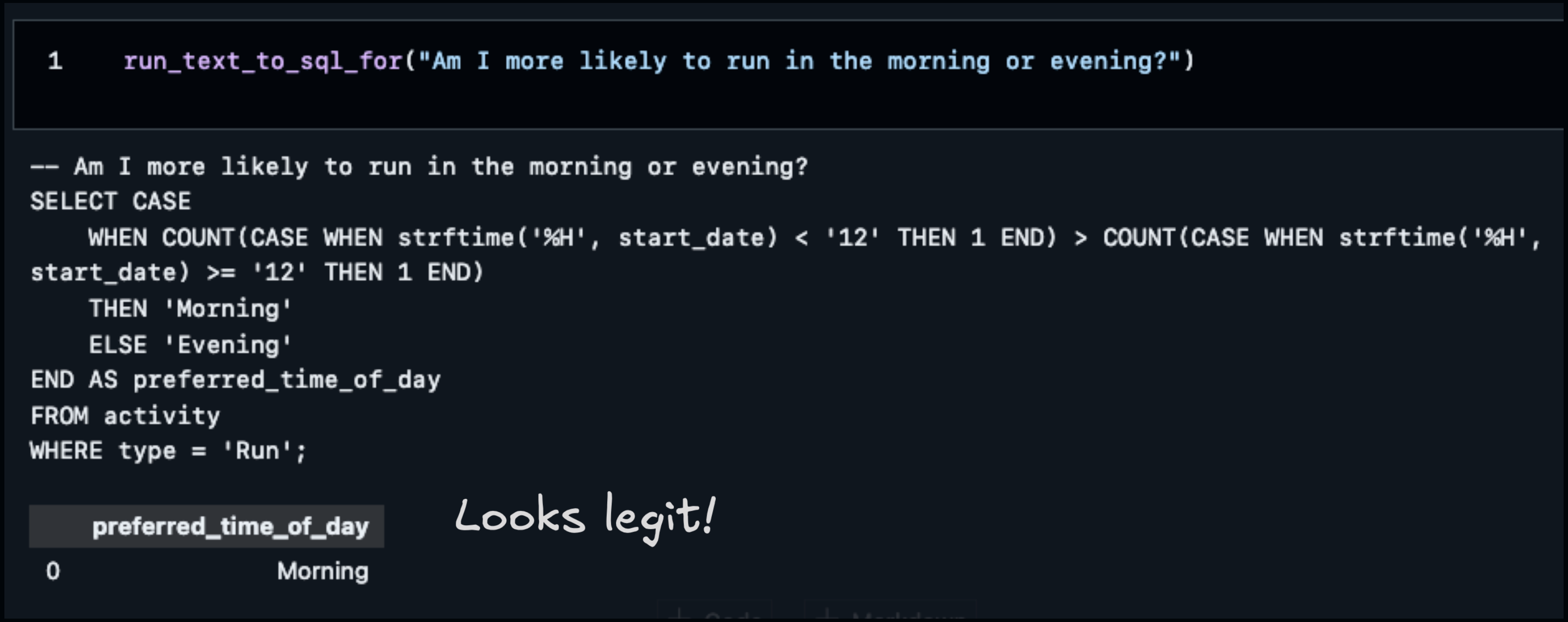
4 Query validator!

```python
def run_text_to_sql_for(question: str, **kwargs) -> SQLiteQuery:
    prompt = make_prompt_for(question)

    query = CLIENT.chat.completions.create(
        model="gpt-4o-mini",
        response_model=SQLiteQuery,
        messages=[{"role": "system", "content": prompt}],
        temperature=0.0,
        **kwargs,
    )

    if not query.query:
        print(
            f"Failed to generate query for: '{question}', explanation: {query.explanation}"
        )
        return None

    print(query.query)
    display(query.execute())
```

$\boxed{1}$ Create prompt

$\boxed{2}$ Call LLM

$\boxed{3}$ Display result of executing!

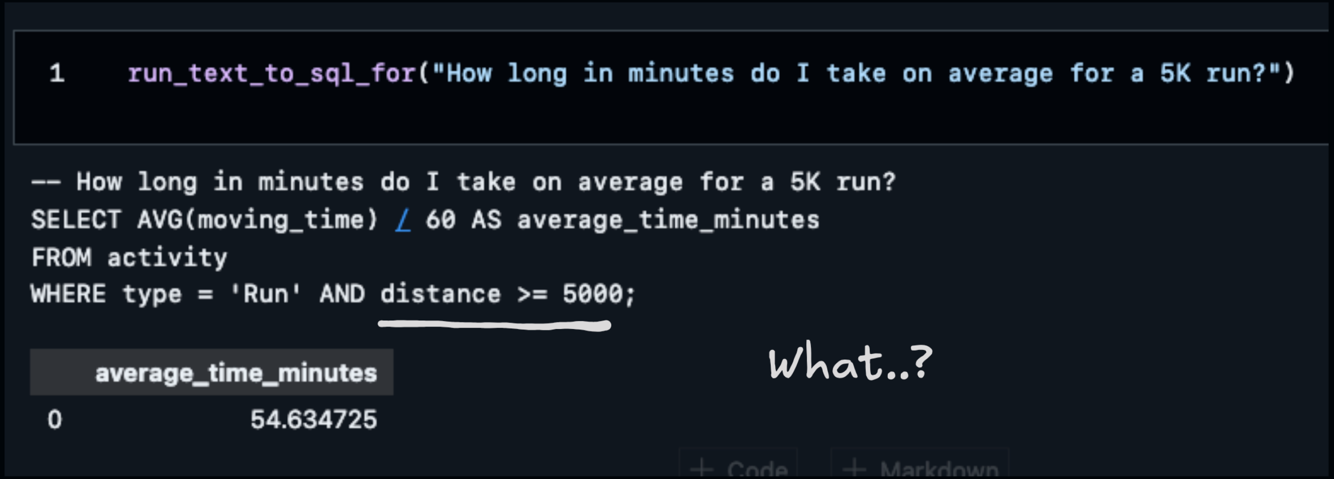


```
run_text_to_sql_for("Am I more likely to run in the morning or evening?")
```

```
-- Am I more likely to run in the morning or evening?
SELECT CASE
    WHEN COUNT(CASE WHEN strftime('%H', start_date) < '12' THEN 1 END) > COUNT(CASE WHEN strftime('%H',
start_date) >= '12' THEN 1 END)
    THEN 'Morning'
    ELSE 'Evening'
END AS preferred_time_of_day
FROM activity
WHERE type = 'Run';
```

	preferred_time_of_day
0	Morning

Looks legit!



```
run_text_to_sql_for("How long in minutes do I take on average for a 5K run?")
```

```
-- How long in minutes do I take on average for a 5K run?
SELECT AVG(moving_time) / 60 AS average_time_minutes
FROM activity
WHERE type = 'Run' AND distance >= 5000;
```

	average_time_minutes
0	54.634725

What..?

https://tinyurl.com/stravatexttosql

duarteocarmo.com

Fine tune a smaller model

Improve query performance

Retries

K-shot prompting

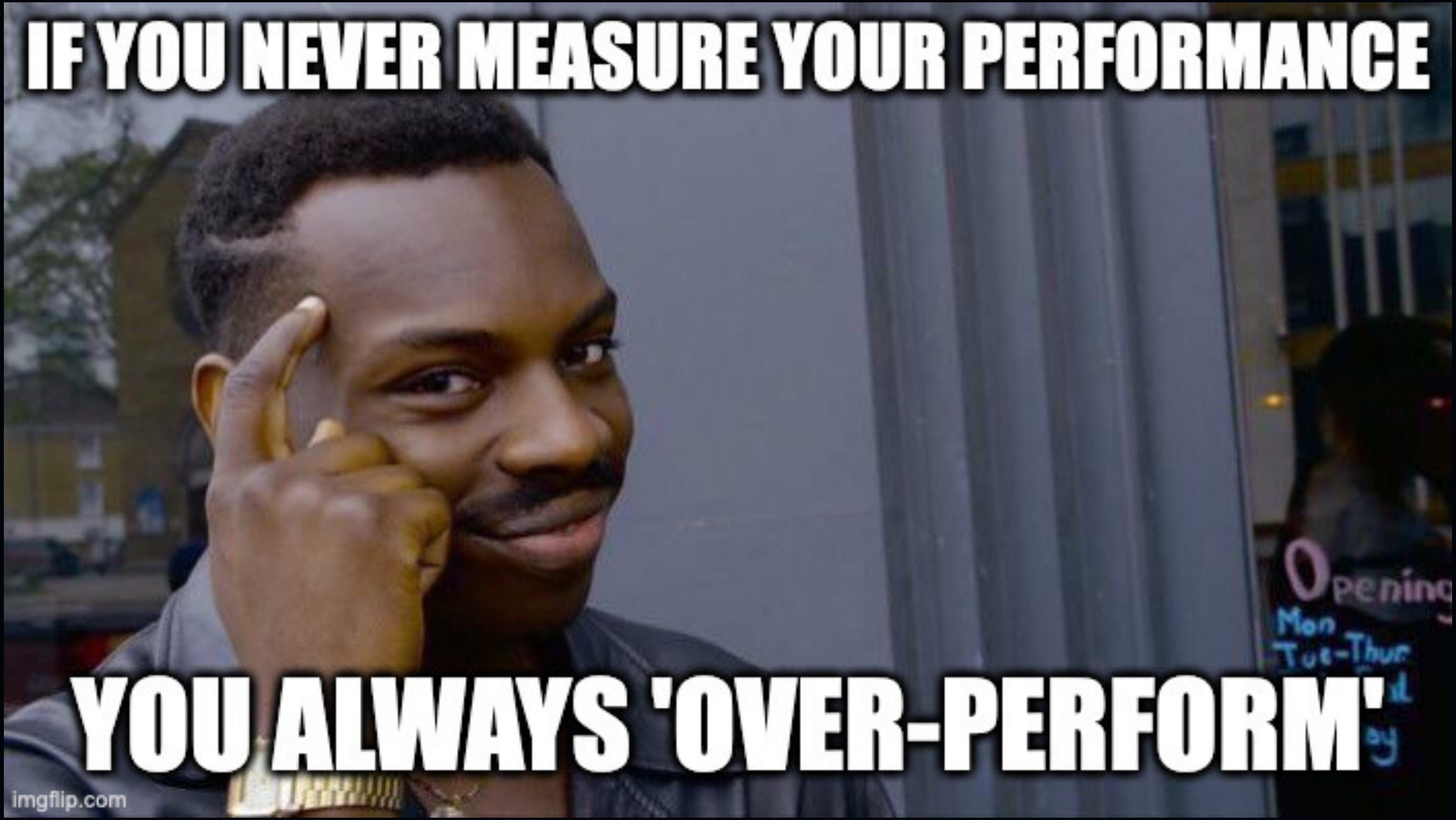
What if we return the whole table?

Query decomposition

....

There are many improvements we can make!

Making things *better*


IF YOU NEVER MEASURE YOUR PERFORMANCE
YOU ALWAYS 'OVER-PERFORM'
imgflip.com

To make things better,
we need to <u>measure</u>

Remember accuracy?

Questions that we ask our Database

Human-made queries to answer those questions

Ask the same questions to our text-to-SQL system

Generate a query using the LLM

Questions that we ask our Database

Human-made queries to answer those questions

Ask the same questions to our text-to-SQL system

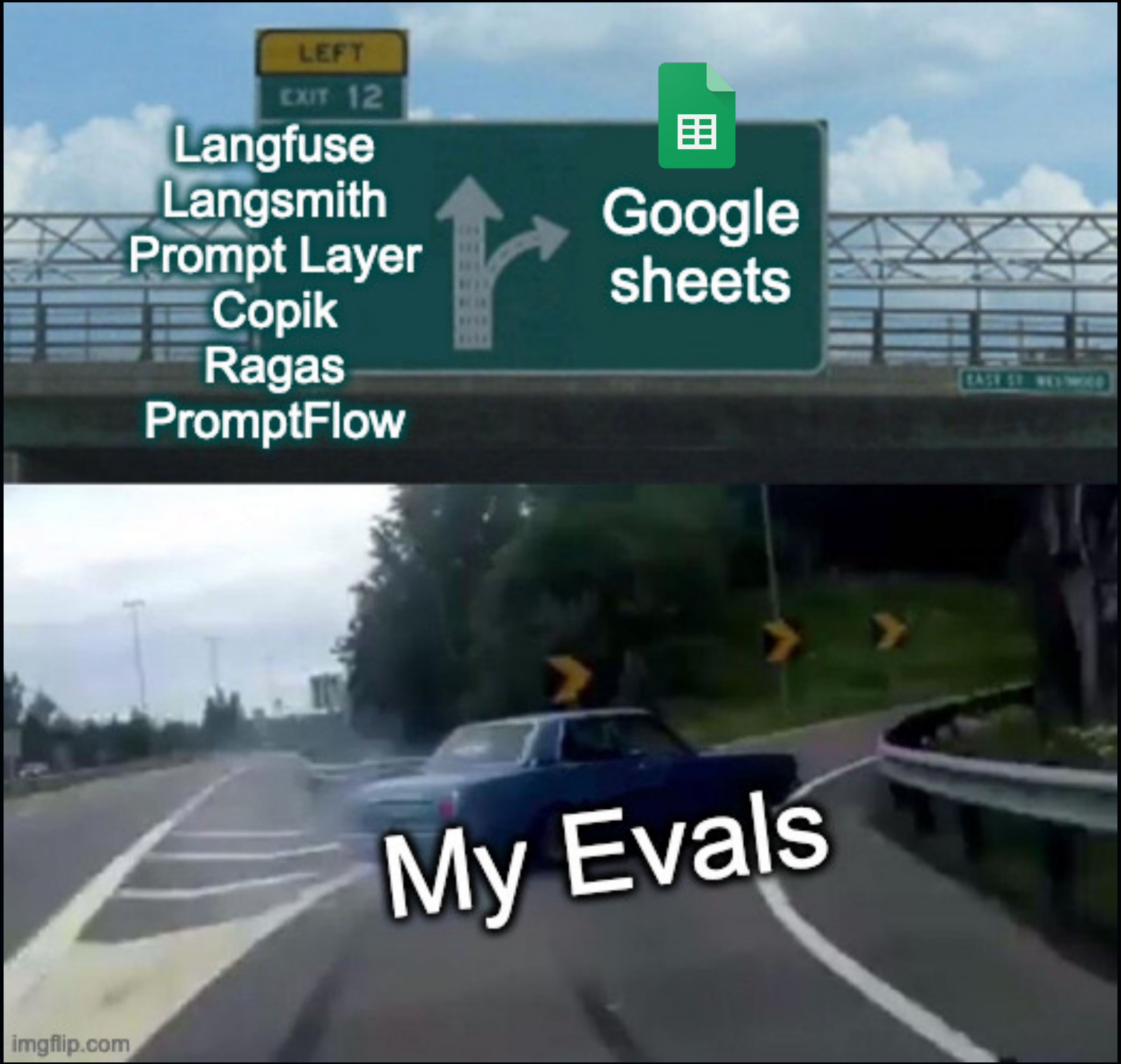
Generate a query using the LLM

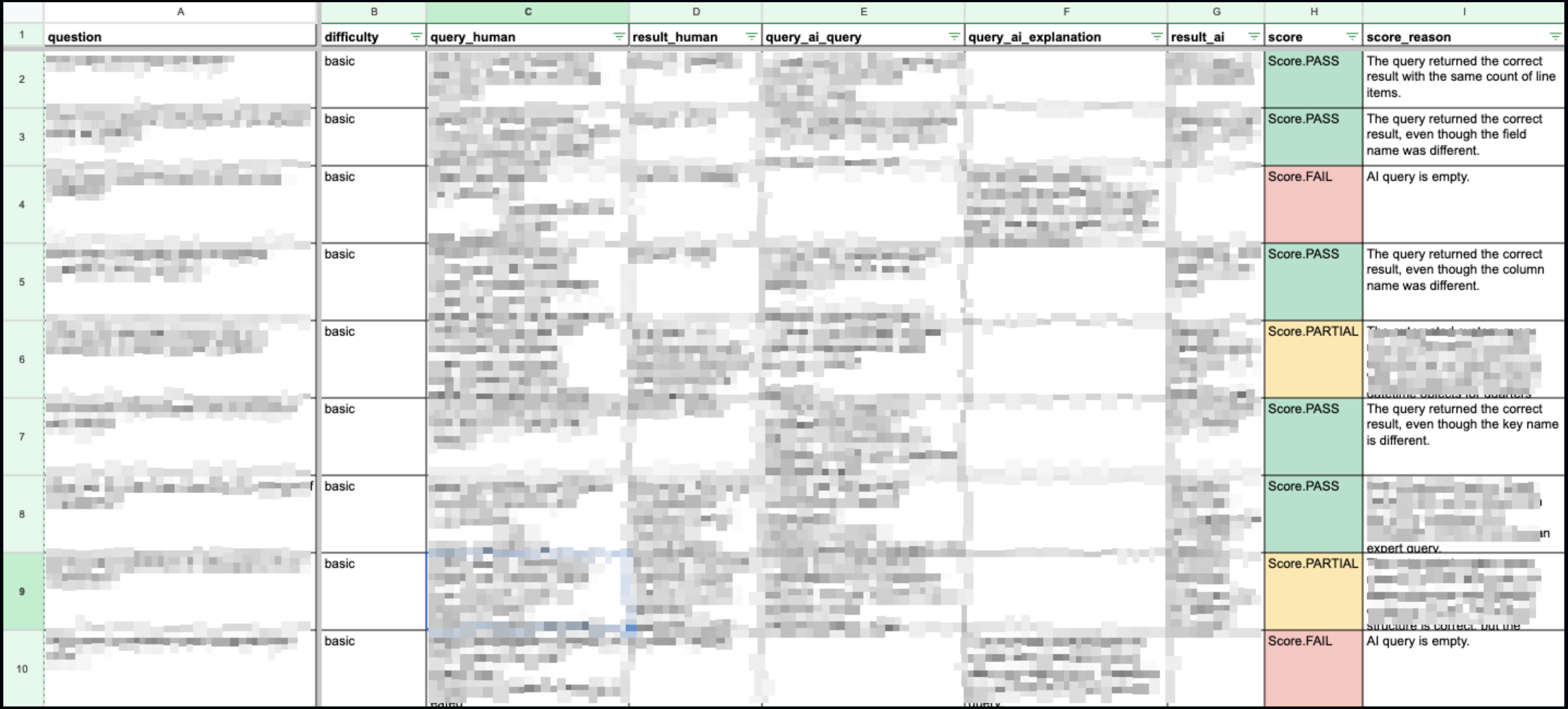
Run both queries and compare results: Are they similar?

If they are similar: PASS, if not, FAIL

How many PASS/FAIL = <u>Accuracy 🎉</u>

```python
class Score(Enum):
    PASS = "PASS"
    FAIL = "FAIL"
    PARTIAL = "PARTIAL"


class TextToSqlEvaluation(BaseModel):
    score: Score = Field(description="The score of the evaluation.")
    reason: str = Field(description="The reason for the score")
```


LEFT
EXIT 12
Langfuse
Langsmith
Prompt Layer
Copik
Ragas
PromptFlow
Google sheets
My Evals
imgflip.com



	A	B	C	D	E	F	G	H	I
1	question	difficulty	query_human	result_human	query_ai_query	query_ai_explanation	result_ai	score	score_reason
2	[illegible]	basic	[illegible]	[illegible]	[illegible]		[illegible]	Score.PASS	The query returned the correct result with the same count of line items.
3	[illegible]	basic	[illegible]	[illegible]	[illegible]		[illegible]	Score.PASS	The query returned the correct result, even though the field name was different.
4	[illegible]	basic	[illegible]	[illegible]		[illegible]		Score.FAIL	AI query is empty.
5	[illegible]	basic	[illegible]	[illegible]	[illegible]		[illegible]	Score.PASS	The query returned the correct result, even though the column name was different.
6	[illegible]	basic	[illegible]	[illegible]	[illegible]		[illegible]	Score.PARTIAL	[illegible] datetime objects for quarters
7	[illegible]	basic	[illegible]	[illegible]	[illegible]		[illegible]	Score.PASS	The query returned the correct result, even though the key name is different.
8	[illegible]	basic	[illegible]	[illegible]	[illegible]		[illegible]	Score.PASS	[illegible] expert query.
9	[illegible]	basic	[illegible]	[illegible]	[illegible]		[illegible]	Score.PARTIAL	[illegible] structure is correct, but the
10	[illegible]	basic	[illegible]	[illegible]		[illegible]		Score.FAIL	AI query is empty.

Can be expanded (latency, query length, result length, etc..)

Cool. So what?

1

Take the cat out of the bag

Set expectations with users/client

Get it out there! Fast!

Make sure interface is clean

2

Make it robust and explainable

Leverage structured generation

Validate question/query loop

Keep it provider-agnostic

3

Iterate with them, not for them

Establish a baseline to start with

Evaluate your generation process

Keep it simple, stupid!

Dziękuję!

Questions?