

Building a Natural Language to SQL Code Generator

DataHour Meetup

Rami Krispin, Nov 27, 2023

About Me

- Senior Manager
- Forecasting
- MLOps
- Open Source
- Author



@RAMIKRISPINDS

Agenda

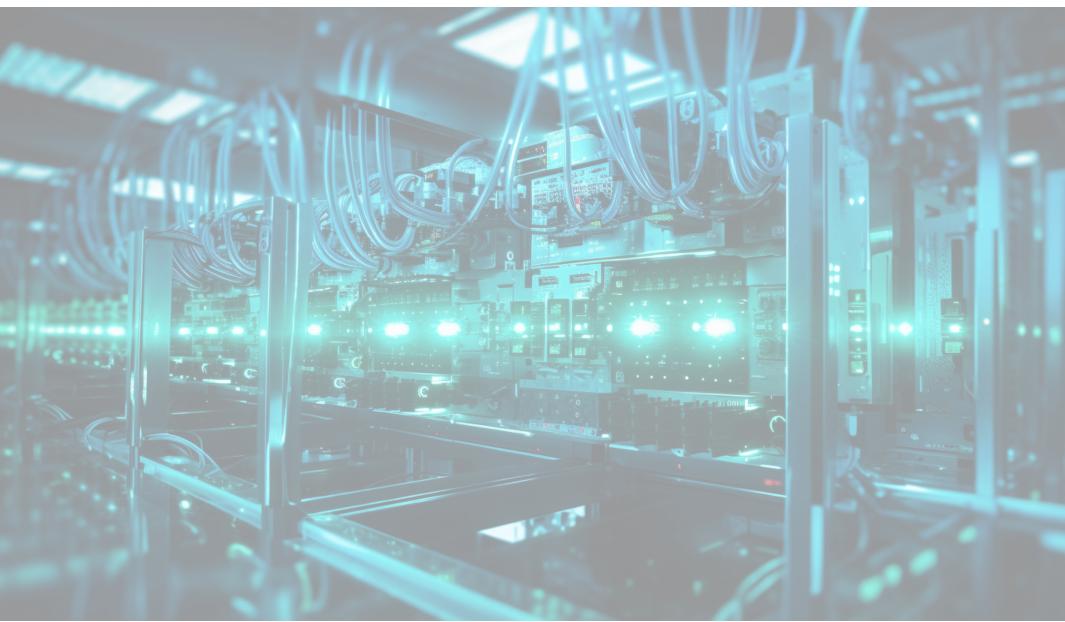
- Introduction
- Motivation
- Architecture
- OpenAI POC
- Next Steps

LLM in a Nutshell

LLM in a Nutshell



Data



Model Training



Inference



Fine Tuning

LLM in a Nutshell



The busy person's
intro to LLMs

Andrey Karpathy

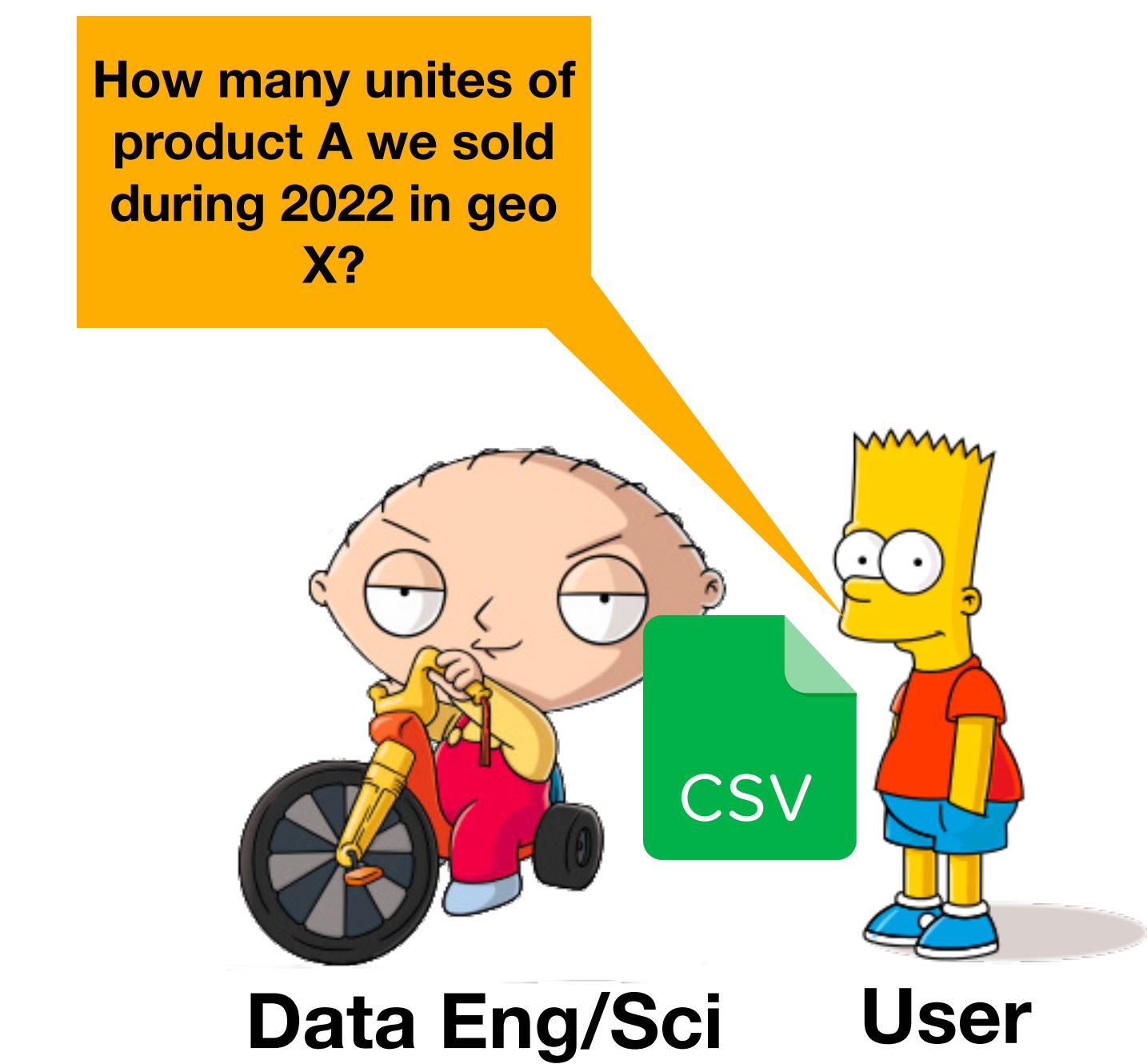


Motivation

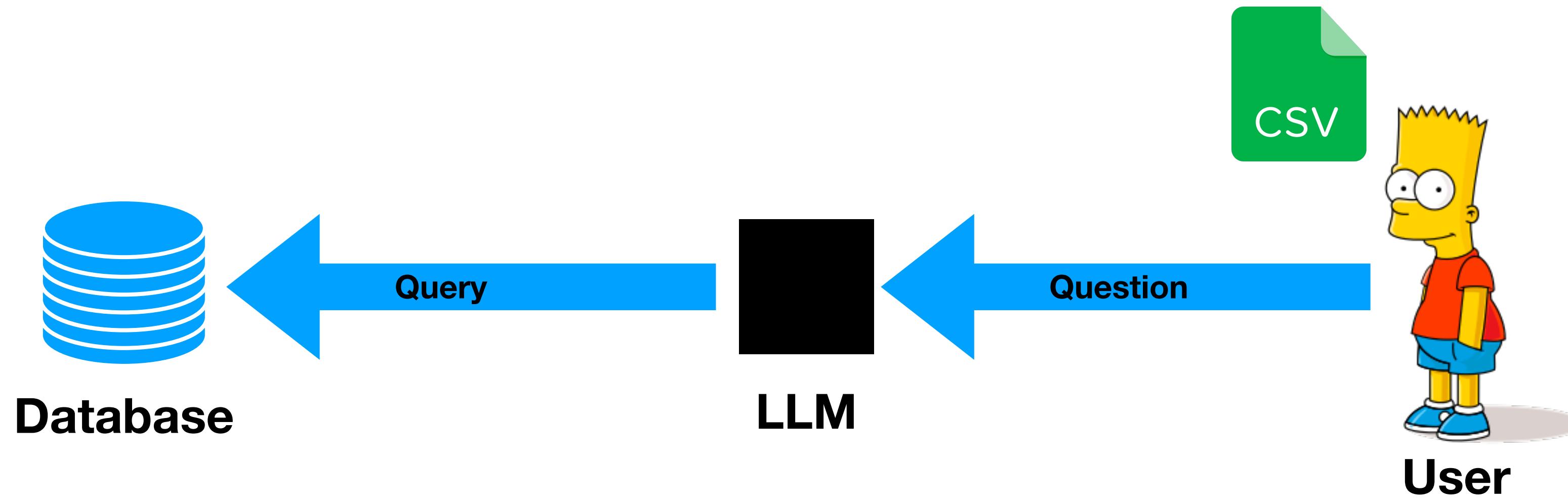
Motivation



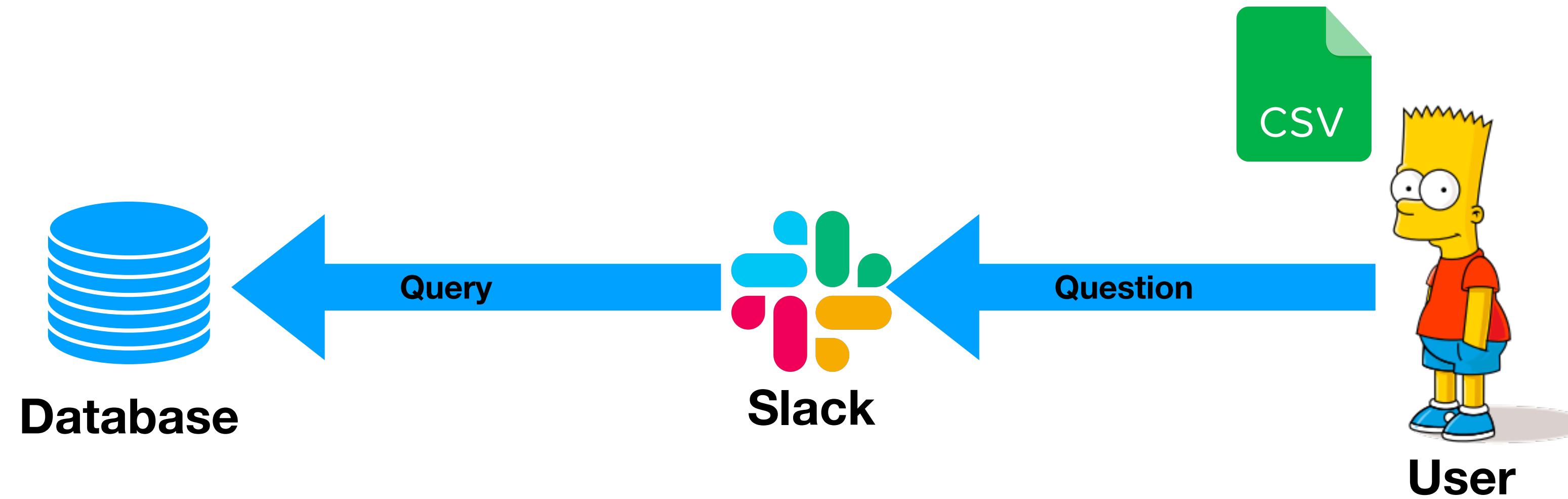
Database



Motivation



Motivation



Motivation



A Hacker's Guide to Language Models

Jeremy Howard, fast.ai

Motivation

```
import torch
from peft import PeftModel
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

ax_model = '/home/jhoward/git/ext/axolotl/glora-out'

tokr = AutoTokenizer.from_pretrained('meta-llama/Llama-2-7b-hf')

model = AutoModelForCausalLM.from_pretrained('meta-llama/Llama-2-7b-hf',
                                              torch_dtype=torch.bfloat16, device_map=0)
model = PeftModel.from_pretrained(model, ax_model)
model = model.merge_and_unload()
model.save_pretrained('sql-model')

toks = tokr(sql_prompt(tst), return_tensors="pt")

res = model.generate(**toks.to("cuda"), max_new_tokens=250).to('cpu')

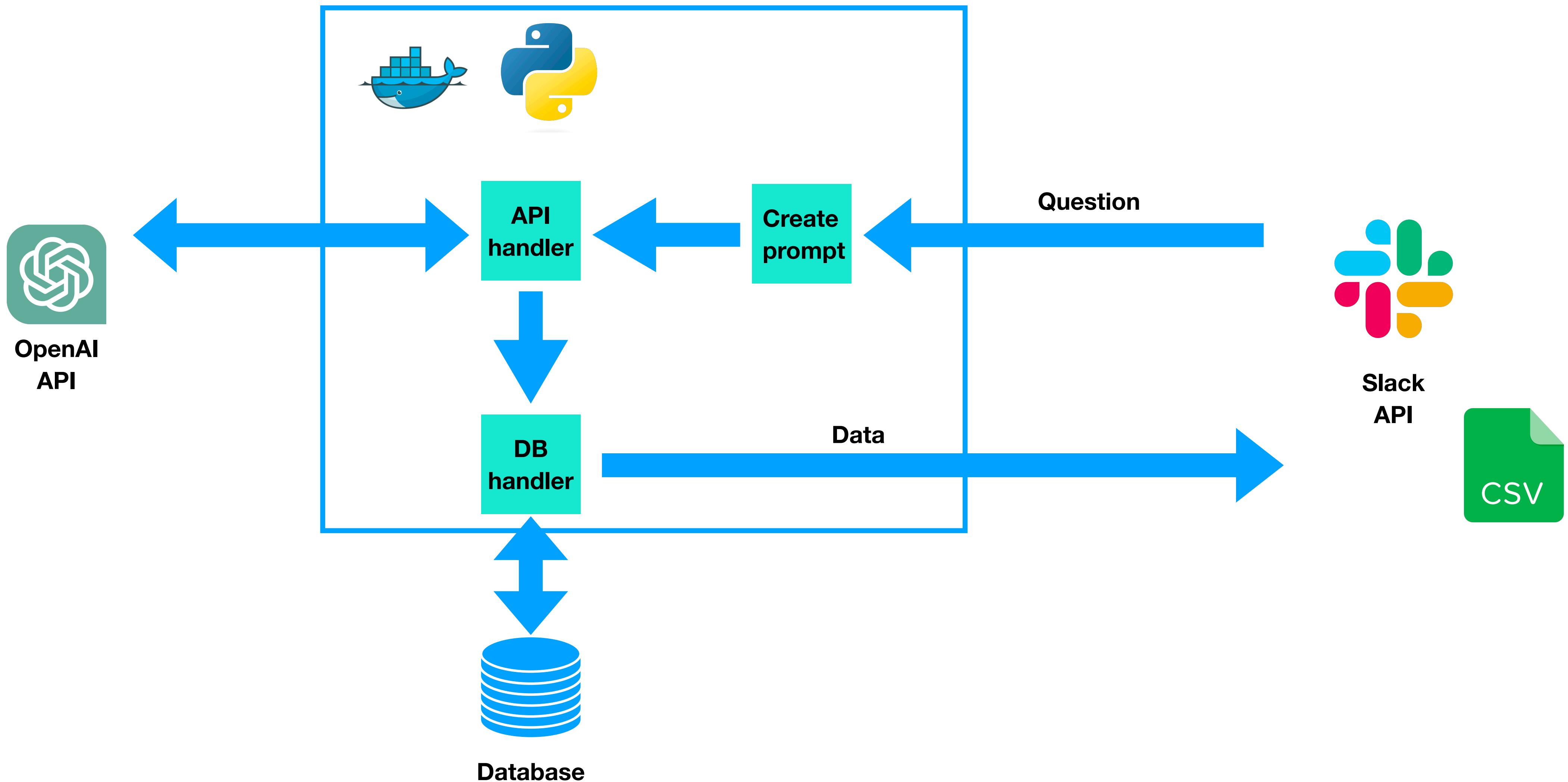
print(tokr.batch_decode(res)[0])
```

<s> SYSTEM: Use the following contextual information to concisely answer the question.

USER: CREATE TABLE farm_competition (Hosts VARCHAR, Theme VARCHAR)
====
Get the count of competition hosts by theme.
ASSISTANT: SELECT COUNT(Hosts), Theme FROM farm_competition GROUP BY Theme</s>

Architecture

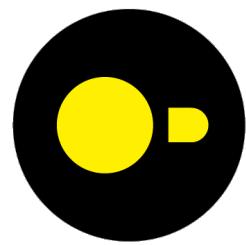
Architecture



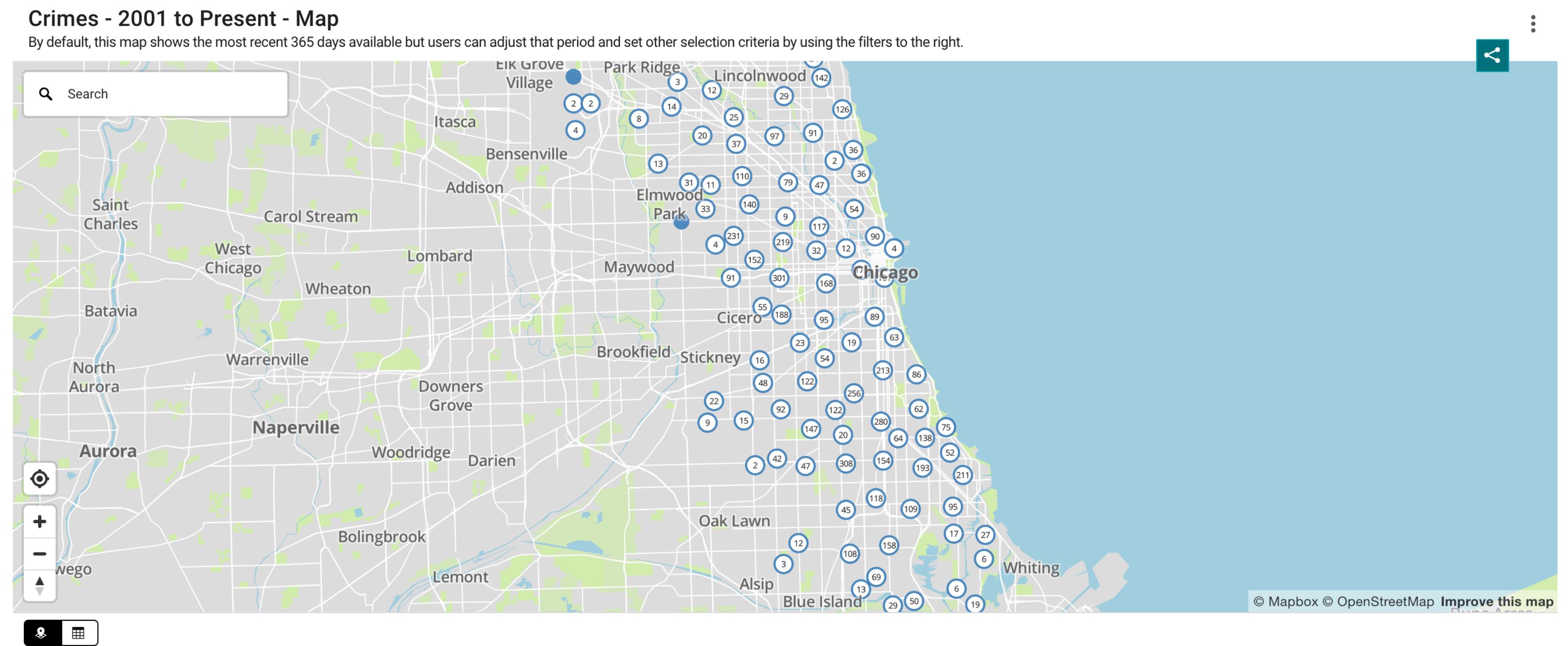
OpenAI POC

OpenAI POC

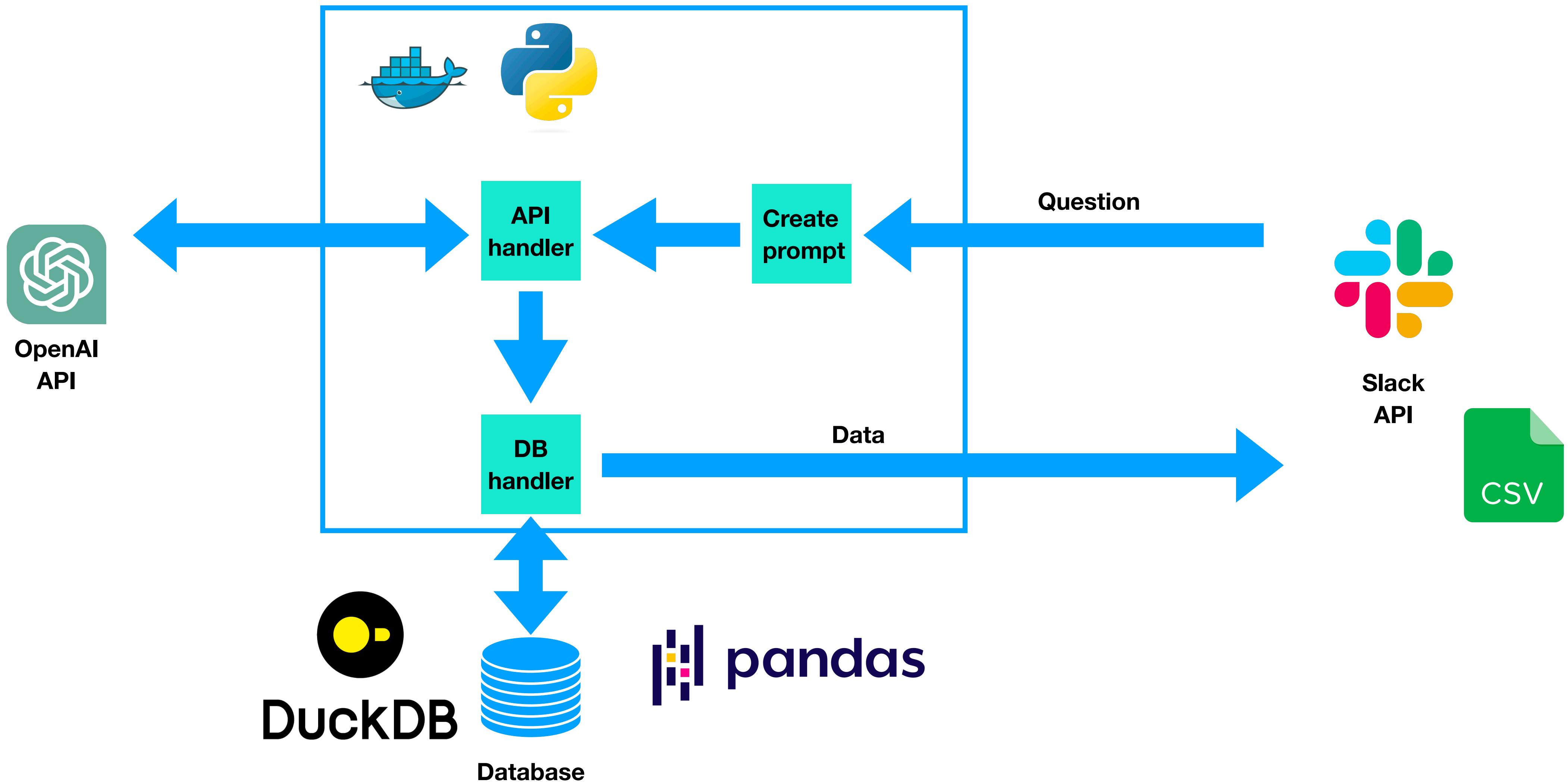
Tools and Data



DuckDB



Architecture



Prompt Engineering 101

It is All About the Context

**How many cases ended up with
arrest?**

OpenAI POC

Prompt

 You
Create an SQL query that counts the number of records that ended up with an arrest

 ChatGPT
To create an SQL query that counts the number of records where an arrest occurred, you need to have a column in your database that indicates whether an arrest happened or not. For the sake of this example, let's assume you have a table called `crime_records` with a column named `arrested` that is a boolean column indicating whether an arrest occurred (1 for true, 0 for false).

Here's a sample SQL query:

sql

```
SELECT COUNT(*) AS arrest_count
FROM crime_records
WHERE arrested = 1;
```

 Copy code

In this query:

- `SELECT COUNT(*)`: This selects the count of records.
- `AS arrest_count`: This aliases the result column as `arrest_count` for easier reference.
- `FROM crime_records`: Specifies the table from which to retrieve the data.
- `WHERE arrested = 1`: Filters the records to include only those where the `arrested` column is equal to 1 (indicating an arrest).

Please adjust the table and column names based on your actual database schema. If your schema is different, provide more details, and I can help you tailor the query accordingly.

OpenAI POC

Prompt

 You
Create an SQL query that counts the number of records that ended up with an arrest

 ChatGPT
To create an SQL query that counts the number of records where an arrest occurred, you need to have a column in your database that indicates whether an arrest happened or not. For the sake of this example, let's assume you have a table called `crime_records` with a column named `arrested` that is a boolean column indicating whether an arrest occurred (1 for true, 0 for false).
Here's a sample SQL query:

sql

```
SELECT COUNT(*) AS arrest_count
FROM crime_records
WHERE arrested = 1;
```

 Copy code

In this query:

- `SELECT COUNT(*)`: This selects the count of records.
- `AS arrest_count`: This aliases the result column as `arrest_count` for easier reference.
- `FROM crime_records`: Specifies the table from which to retrieve the data.
- `WHERE arrested = 1`: Filters the records to include only those where the `arrested` column is equal to 1 (indicating an arrest).

Please adjust the table and column names based on your actual database schema. If your schema is different, provide more details, and I can help you tailor the query accordingly.

OpenAI POC

Prompt

The screenshot shows a dark-themed chat interface. A user message from 'You' is at the top, followed by a response from 'ChatGPT'. The user message includes a small profile icon with 'RA' and the text: 'I have a table named chicago_crime with the crime records in Chicago City since 2021. The Arrest field defines if the case ended up with arrest or not, and it is a boolean (true or false). I want to create an SQL query that counts the number of records that ended up with an arrest.' The AI response from 'ChatGPT' includes a small profile icon with a green swirl and the text: 'Certainly! You can use the `COUNT` function along with a `WHERE` clause to filter records where the Arrest field is true. Here's an example SQL query:' Below this, there is a code block labeled 'sql' containing the following SQL query:

```
sql
SELECT COUNT(*) AS ArrestedCases
FROM chicago_crime
WHERE Arrest = true;
```

A 'Copy code' button is located to the right of the code block. Below the code block, a note explains the query: 'This query selects the count of records from the `chicago_crime` table where the `Arrest` field is true and aliases the result as `ArrestedCases`.'

At the bottom of the AI response, there are four small icons: a copy icon, a share icon, a reply icon, and a refresh/circular arrow icon.

OpenAI POC

Prompt

A screenshot of a ChatGPT interface. The user (RA) asks about a table named 'chicago_crime' containing crime records from Chicago City since 2021. The 'Arrest' field is a boolean indicating if the case ended with an arrest. The user wants to create an SQL query to count the number of records where 'Arrest' is true. ChatGPT responds by providing an example SQL query:

```
sql
SELECT COUNT(*) AS ArrestedCases
FROM chicago_crime
WHERE Arrest = true;
```

The AI's response is highlighted with a blue border. Below the code, a note explains its purpose: "This query selects the count of records from the `chicago_crime` table where the `Arrest` field is true and aliases the result as `ArrestedCases`."

OpenAI POC

Prompt

SYSTEM Given the following SQL tables, your job is to write queries given a user's request.

```
CREATE TABLE Orders (
    OrderID int,
    CustomerID int,
    OrderDate datetime,
    OrderTime varchar(8),
    PRIMARY KEY (OrderID)
);
```

```
CREATE TABLE OrderDetails (
    OrderDetailID int,
    OrderID int,
    ProductID int,
    Quantity int,
    PRIMARY KEY (OrderDetailID)
);
```

```
CREATE TABLE Products (
    ProductID int,
    ProductName varchar(50),
    Category varchar(50),
    UnitPrice decimal(10, 2),
    Stock int,
    PRIMARY KEY (ProductID)
);
```

```
CREATE TABLE Customers (
    CustomerID int,
    FirstName varchar(50),
    LastName varchar(50),
    Email varchar(100),
    Phone varchar(20),
    PRIMARY KEY (CustomerID)
);
```

USER Write a SQL query which computes the average total order value for all orders on 2023-04-01.

OpenAI POC

Prompt

SYSTEM

Given the following SQL tables, your job is to write queries given a user's request.

```
CREATE TABLE Orders (
    OrderID int,
    CustomerID int,
    OrderDate datetime,
    OrderTime varchar(8),
    PRIMARY KEY (OrderID)
);
```

```
CREATE TABLE OrderDetails (
    OrderDetailID int,
    OrderID int,
    ProductID int,
    Quantity int,
    PRIMARY KEY (OrderDetailID)
);
```

```
CREATE TABLE Products (
    ProductID int,
    ProductName varchar(50),
    Category varchar(50),
    UnitPrice decimal(10, 2),
    Stock int,
    PRIMARY KEY (ProductID)
);
```

```
CREATE TABLE Customers (
    CustomerID int,
    FirstName varchar(50),
    LastName varchar(50),
    Email varchar(100),
    Phone varchar(20),
    PRIMARY KEY (CustomerID)
);
```

USER

Write a SQL query which computes the average total order value for all orders on 2023-04-01.

OpenAI POC

Prompt

SYSTEM Given the following SQL tables, your job is to write queries given a user's request.

```
CREATE TABLE Orders (
    OrderID int,
    CustomerID int,
    OrderDate datetime,
    OrderTime varchar(8),
    PRIMARY KEY (OrderID)
);
```

```
CREATE TABLE OrderDetails (
    OrderDetailID int,
    OrderID int,
    ProductID int,
    Quantity int,
    PRIMARY KEY (OrderDetailID)
);
```

```
CREATE TABLE Products (
    ProductID int,
    ProductName varchar(50),
    Category varchar(50),
    UnitPrice decimal(10, 2),
    Stock int,
    PRIMARY KEY (ProductID)
);
```

```
CREATE TABLE Customers (
    CustomerID int,
    FirstName varchar(50),
    LastName varchar(50),
    Email varchar(100),
    Phone varchar(20),
    PRIMARY KEY (CustomerID)
);
```

USER Write a SQL query which computes the average total order value for all orders on 2023-04-01.

OpenAI POC

Prompt

```
def create_message(table_name, query):

    class message:
        def __init__(message, system, user, column_names, column_attr):
            message.system = system
            message.user = user
            message.column_names = column_names
            message.column_attr = column_attr

    system_template = """

        Given the following SQL table, your job is to write queries given a user's request. \n

        CREATE TABLE {} ({})

    """

    user_template = "Write a SQL query that returns - {}"

    tbl_describe = duckdb.sql("DESCRIBE SELECT * FROM " + table_name + ";")
    col_attr = tbl_describe.df()[["column_name", "column_type"]]
    col_attr["column_joint"] = col_attr["column_name"] + " " + col_attr["column_type"]
    col_names = str(list(col_attr["column_joint"].values)).replace('[', '').replace(']', '').replace('\'', '')

    system = system_template.format(table_name, col_names)
    user = user_template.format(query)

    m = message(system = system, user = user, column_names = col_attr["column_name"], column_attr = col_attr["column_type"])
    return m
```

OpenAI POC

Prompt

```
def create_message(table_name, query):

    class message:
        def __init__(message, system, user, column_names, column_attr):
            message.system = system
            message.user = user
            message.column_names = column_names
            message.column_attr = column_attr

    system_template = """

        Given the following SQL table, your job is to write queries given a user's request. \n

        CREATE TABLE {} ({})

    """

    user_template = "Write a SQL query that returns - {}"

    tbl_describe = duckdb.sql("DESCRIBE SELECT * FROM " + table_name + ";")
    col_attr = tbl_describe.df()[["column_name", "column_type"]]
    col_attr["column_joint"] = col_attr["column_name"] + " " + col_attr["column_type"]
    col_names = str(list(col_attr["column_joint"].values)).replace('[', '').replace(']', '').replace('\'', '')

    system = system_template.format(table_name, col_names)
    user = user_template.format(query)

    m = message(system = system, user = user, column_names = col_attr["column_name"], column_attr = col_attr["column_type"])
    return m
```

OpenAI POC

Prompt

```
def create_message(table_name, query):

    class message:
        def __init__(message, system, user, column_names, column_attr):
            message.system = system
            message.user = user
            message.column_names = column_names
            message.column_attr = column_attr

    system_template = """

        Given the following SQL table, your job is to write queries given a user's request. \n

        CREATE TABLE {} ({})

    """

    user_template = "Write a SQL query that returns - {}"

    tbl_describe = duckdb.sql("DESCRIBE SELECT * FROM " + table_name + ";")
    col_attr = tbl_describe.df()[["column_name", "column_type"]]
    col_attr["column_joint"] = col_attr["column_name"] + " " + col_attr["column_type"]
    col_names = str(list(col_attr["column_joint"].values)).replace('[', '').replace(']', '').replace('\'', '')

    system = system_template.format(table_name, col_names)
    user = user_template.format(query)

    m = message(system = system, user = user, column_names = col_attr["column_name"], column_attr = col_attr["column_type"])
    return m
```

OpenAI POC

Prompt

```
def create_message(table_name, query):

    class message:
        def __init__(message, system, user, column_names, column_attr):
            message.system = system
            message.user = user
            message.column_names = column_names
            message.column_attr = column_attr

    system_template = """

        Given the following SQL table, your job is to write queries given a user's request. \n

        CREATE TABLE {} ({})

    """

    user_template = "Write a SQL query that returns - {}"

    tbl_describe = duckdb.sql("DESCRIBE SELECT * FROM " + table_name + ";")
    col_attr = tbl_describe.df()[["column_name", "column_type"]]
    col_attr["column_joint"] = col_attr["column_name"] + " " + col_attr["column_type"]
    col_names = str(list(col_attr["column_joint"].values)).replace('[', '').replace(']', '').replace('\'', '')

    system = system_template.format(table_name, col_names)
    user = user_template.format(query)

    m = message(system = system, user = user, column_names = col_attr["column_name"], column_attr = col_attr["col
    return m
```

OpenAI POC

Language to SQL

```
def lang2sql(api_key, table_name, query, model = "gpt-3.5-turbo", temperature = 0, max_tokens = 256, top_p = 1, f:
    class response:
        def __init__(output, message, response, sql):
            output.message = message
            output.response = response
            output.sql = sql

    openai.api_key = api_key

    m = create_message(table_name = table_name, query = query)

    message = [
        {
            "role": "system",
            "content": m.system
        },
        {
            "role": "user",
            "content": m.user
        }
    ]

    openai_response = openai.ChatCompletion.create(
        model = model,
        messages = message,
        temperature = temperature,
        max_tokens = max_tokens,
        top_p = top_p,
        frequency_penalty = frequency_penalty,
        presence_penalty = presence_penalty)

    sql_query = add_quotes(query = openai_response["choices"][0]["message"]["content"], col_names = m.column_names)

    output = response(message = m, response = openai_response, sql = sql_query)

return output
```

OpenAI POC

Language to SQL

```
def lang2sql(api_key, table_name, query, model = "gpt-3.5-turbo", temperature = 0, max_tokens = 256, top_p = 1, frequency_penalty = 0, presence_penalty = 0):
    class response:
        def __init__(self, message, response, sql):
            self.message = message
            self.response = response
            self.sql = sql

    openai.api_key = api_key

    m = create_message(table_name = table_name, query = query)

    message = [
        {
            "role": "system",
            "content": m.system
        },
        {
            "role": "user",
            "content": m.user
        }
    ]

    openai_response = openai.ChatCompletion.create(
        model = model,
        messages = message,
        temperature = temperature,
        max_tokens = max_tokens,
        top_p = top_p,
        frequency_penalty = frequency_penalty,
        presence_penalty = presence_penalty)

    sql_query = add_quotes(query = openai_response["choices"][0]["message"]["content"], col_names = m.column_names)

    output = response(message = m, response = openai_response, sql = sql_query)

    return output
```

OpenAI POC

Language to SQL

```
def lang2sql(api_key, table_name, query, model = "gpt-3.5-turbo", temperature = 0, max_tokens = 256, top_p = 1, f:
    class response:
        def __init__(output, message, response, sql):
            output.message = message
            output.response = response
            output.sql = sql

        openai.api_key = api_key

        m = create_message(table_name = table_name, query = query)

        message = [
            {
                "role": "system",
                "content": m.system
            },
            {
                "role": "user",
                "content": m.user
            }
        ]

        openai_response = openai.ChatCompletion.create(
            model = model,
            messages = message,
            temperature = temperature,
            max_tokens = max_tokens,
            top_p = top_p,
            frequency_penalty = frequency_penalty,
            presence_penalty = presence_penalty)

        sql_query = add_quotes(query = openai_response["choices"][0]["message"]["content"], col_names = m.column_names)

        output = response(message = m, response = openai_response, sql = sql_query)

    return output
```

OpenAI POC Example

```
query = "How many cases ended up with arrest?"
response = lang2sql(api_key = api_key, table_name = "chicago_crime", query = query)

print(response.message)
print(response.sql)
print(response.response)

<__main__.create_message.<locals>.message object at 0xfffff4035c2e0>
SELECT COUNT(*) FROM chicago_crime WHERE "Arrest" = true;
{
    "id": "chatcmpl-8Cg1DSYrYSzUyOZP25MjNGGd8VC6b",
    "object": "chat.completion",
    "created": 1698034999,
    "model": "gpt-3.5-turbo-0613",
    "choices": [
        {
            "index": 0,
            "message": {
                "role": "assistant",
                "content": "SELECT COUNT(*) FROM chicago_crime WHERE Arrest = true;"
            },
            "finish_reason": "stop"
        }
    ],
    "usage": {
        "prompt_tokens": 137,
        "completion_tokens": 12,
        "total_tokens": 149
    }
}

duckdb.sql(response.sql).show()
```

count_star()
int64
77635

OpenAI POC Example

```
query = "How many cases ended up with arrest?"  
response = lang2sql(api_key = api_key, table_name = "chicago_crime", query = query)
```

```
print(response.message)  
print(response.sql)  
print(response.response)  
  
<__main__.create_message.<locals>.message object at 0xfffff4035c2e0>  
SELECT COUNT(*) FROM chicago_crime WHERE "Arrest" = true;  
{  
    "id": "chatcmpl-8Cg1DSYrYSzUyOZP25MjNGGd8VC6b",  
    "object": "chat.completion",  
    "created": 1698034999,  
    "model": "gpt-3.5-turbo-0613",  
    "choices": [  
        {  
            "index": 0,  
            "message": {  
                "role": "assistant",  
                "content": "SELECT COUNT(*) FROM chicago_crime WHERE Arrest = true;"  
            },  
            "finish_reason": "stop"  
        }  
    ],  
    "usage": {  
        "prompt_tokens": 137,  
        "completion_tokens": 12,  
        "total_tokens": 149  
    }  
}
```

```
duckdb.sql(response.sql).show()
```

count_star()
int64
77635

OpenAI POC Example

```
query = "How many cases ended up with arrest?"
response = lang2sql(api_key = api_key, table_name = "chicago_crime", query = query)

print(response.message)
print(response.sql)
print(response.response)

<__main__.create_message.<locals>.message object at 0xfffff4035c2e0>
SELECT COUNT(*) FROM chicago_crime WHERE "Arrest" = true;
{
    "id": "chatcmpl-8Cg1DSYrYSzUyOZP25MjNGGd8VC6b",
    "object": "chat.completion",
    "created": 1698034999,
    "model": "gpt-3.5-turbo-0613",
    "choices": [
        {
            "index": 0,
            "message": {
                "role": "assistant",
                "content": "SELECT COUNT(*) FROM chicago_crime WHERE Arrest = true;"
            },
            "finish_reason": "stop"
        }
    ],
    "usage": {
        "prompt_tokens": 137,
        "completion_tokens": 12,
        "total_tokens": 149
    }
}
```

```
duckdb.sql(response.sql).show()
```

count_star()
int64
77635

OpenAI POC

Example

```
query = "How many cases ended up with arrest between 2022 and 2023?"  
response = lang2sql(api_key = api_key, table_name = "chicago_crime", query = query)  
print(response.sql)
```

```
duckdb.sql(response.sql).show()
```

```
SELECT COUNT(*) FROM chicago_crime WHERE "Arrest" = TRUE AND "Year" BETWEEN 2022 AND 2023;
```

count_star()
int64
51265

```
query = "Summarize the cases by primary type"
```

```
response = lang2sql(api_key = api_key, table_name = "chicago_crime", query = query)
```

```
print(response.sql)
```

```
duckdb.sql(response.sql).show()
```

```
SELECT "Primary Type", COUNT(*) as TotalCases  
FROM chicago_crime  
GROUP BY "Primary Type"
```

Primary Type varchar	TotalCases int64
THEFT	139021
ARSON	1344
KIDNAPPING	319
HUMAN TRAFFICKING	32
OTHER NARCOTIC VIOLATION	11
GAMBLING	33
ASSAULT	58685
BURGLARY	19898
CRIMINAL DAMAGE	75611
NARCOTICS	13931
.	.
.	.
.	.
OBSCENITY	127
BATTERY	115760
CRIMINAL TRESPASS	11255
CRIMINAL SEXUAL ASSAULT	4300
PUBLIC PEACE VIOLATION	1980
STALKING	1206

OpenAI POC

Example

```
query = "How many cases ended up with arrest between 2022 and 2023?"  
response = lang2sql(api_key = api_key, table_name = "chicago_crime", query = query)  
print(response.sql)  
  
duckdb.sql(response.sql).show()  
  
SELECT COUNT(*) FROM chicago_crime WHERE "Arrest" = TRUE AND "Year" BETWEEN 2022 AND 2023;  
  


| count_star() | int64 |
|--------------|-------|
| 51265        |       |


```

```
query = "Summarize the cases by primary type"  
response = lang2sql(api_key = api_key, table_name = "chicago_crime", query = query)  
  
print(response.sql)  
  
duckdb.sql(response.sql).show()
```

```
SELECT "Primary Type", COUNT(*) as TotalCases  
FROM chicago_crime  
GROUP BY "Primary Type"
```

Primary Type varchar	TotalCases int64
THEFT	139021
ARSON	1344
KIDNAPPING	319
HUMAN TRAFFICKING	32
OTHER NARCOTIC VIOLATION	11
GAMBLING	33
ASSAULT	58685
BURGLARY	19898
CRIMINAL DAMAGE	75611
NARCOTICS	13931
.	.
.	.
.	.
OBSCENITY	127
BATTERY	115760
CRIMINAL TRESPASS	11255
CRIMINAL SEXUAL ASSAULT	4300
PUBLIC PEACE VIOLATION	1980
STALKING	1206

OpenAI POC

Conclusion

- Works well for simple queries
- Reproducibility
- Cost
- Performance

Next Steps

Next Steps

- Google PaLM 2
- Hugging Face API POC
- Fine tuning
- Other applications - visualization, forecasting, etc

Resources

- Code - <https://github.com/RamiKrispin/lang2sql>
- Jeremy Howard, [A Hackers' Guide to Language Models](#)
- Notebook: <https://github.com/fastai/lm-hackers/blob/main/lm-hackers.ipynb>
- Andrej Karpathy, [Intro to LLM](#)
- OpenAI example - <https://platform.openai.com/examples/default-sql-translate>

Questions?



Thank You!