Building a Language to SQL Translator

Illuminate AI Bay Area Chapter Meetup

Rami Krispin, Nov 2nd, 2023

Agenda

- Motivation
- Architecture
- OpenAI POC
- Next Steps

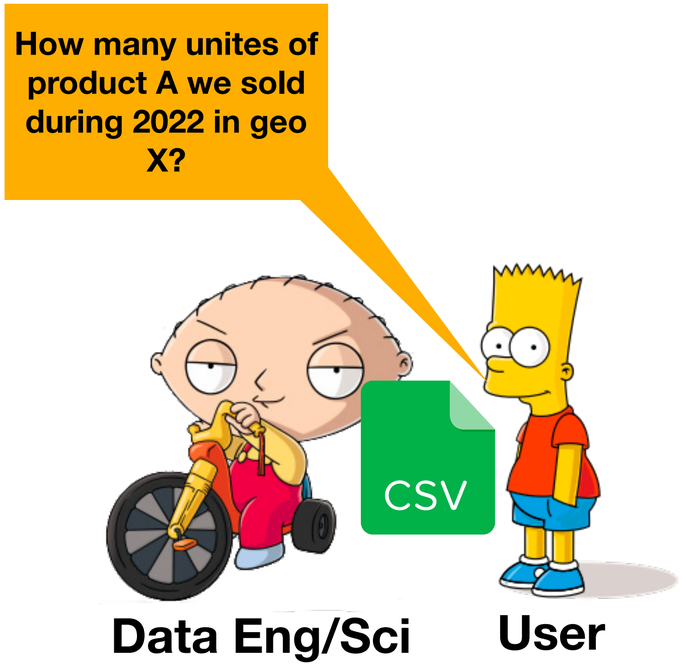
About Me

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

Motivation

Motivation

Database



Motivation

Motivation

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

```
ax_model = '/home/jhoward/git/ext/axolotl/qlora-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")
```

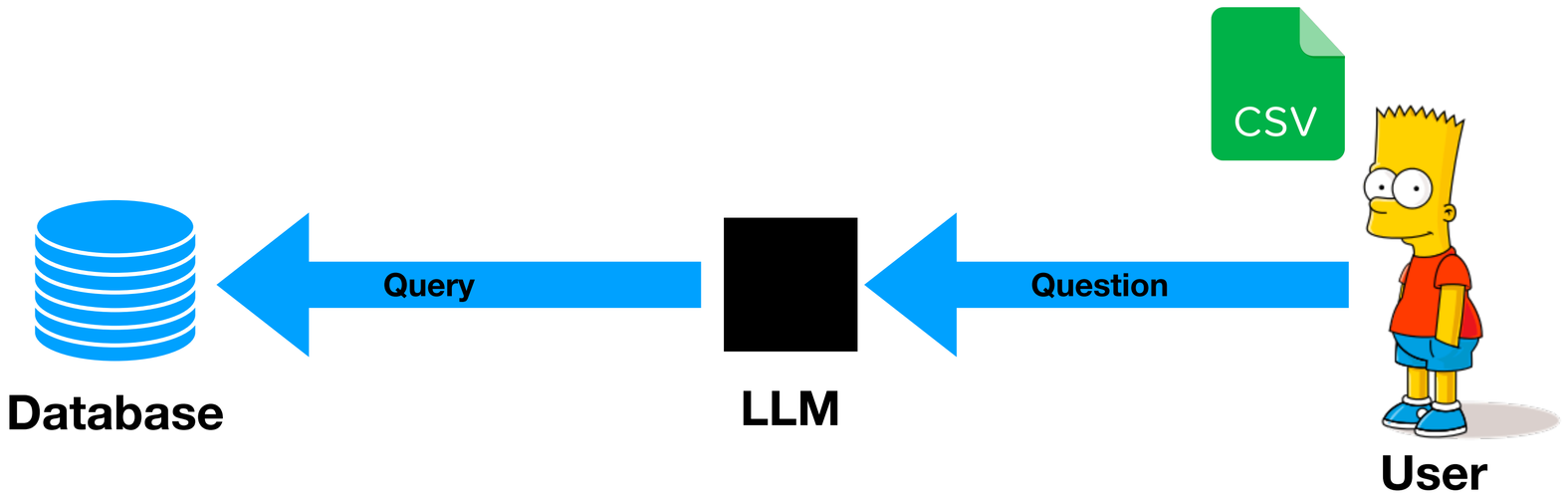
```
res = model.generate(**toks.to("cuda"), max_new_tokens=250).to('cpu')
```

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

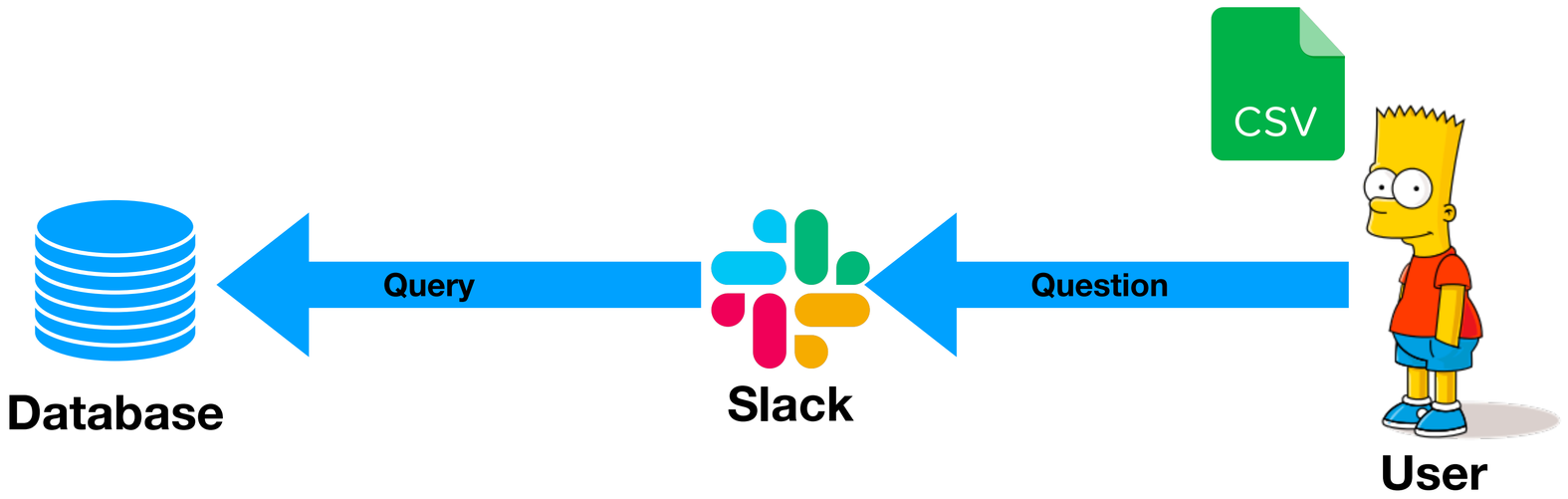
```
<s> SYSTEM: Use the following contextual information to concisely answer the question.

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>
```

Motivation

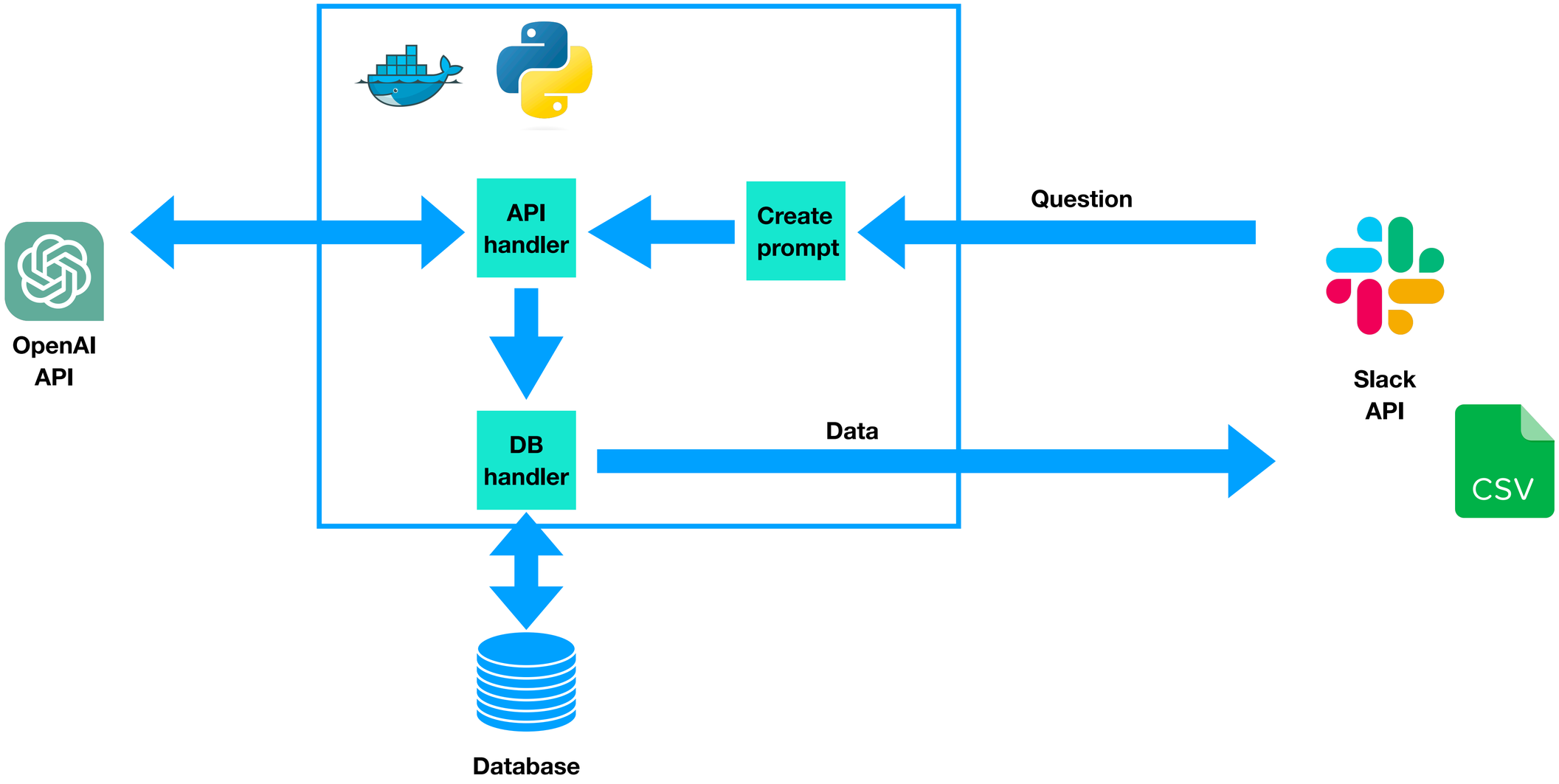


Motivation



Architecture

Architecture



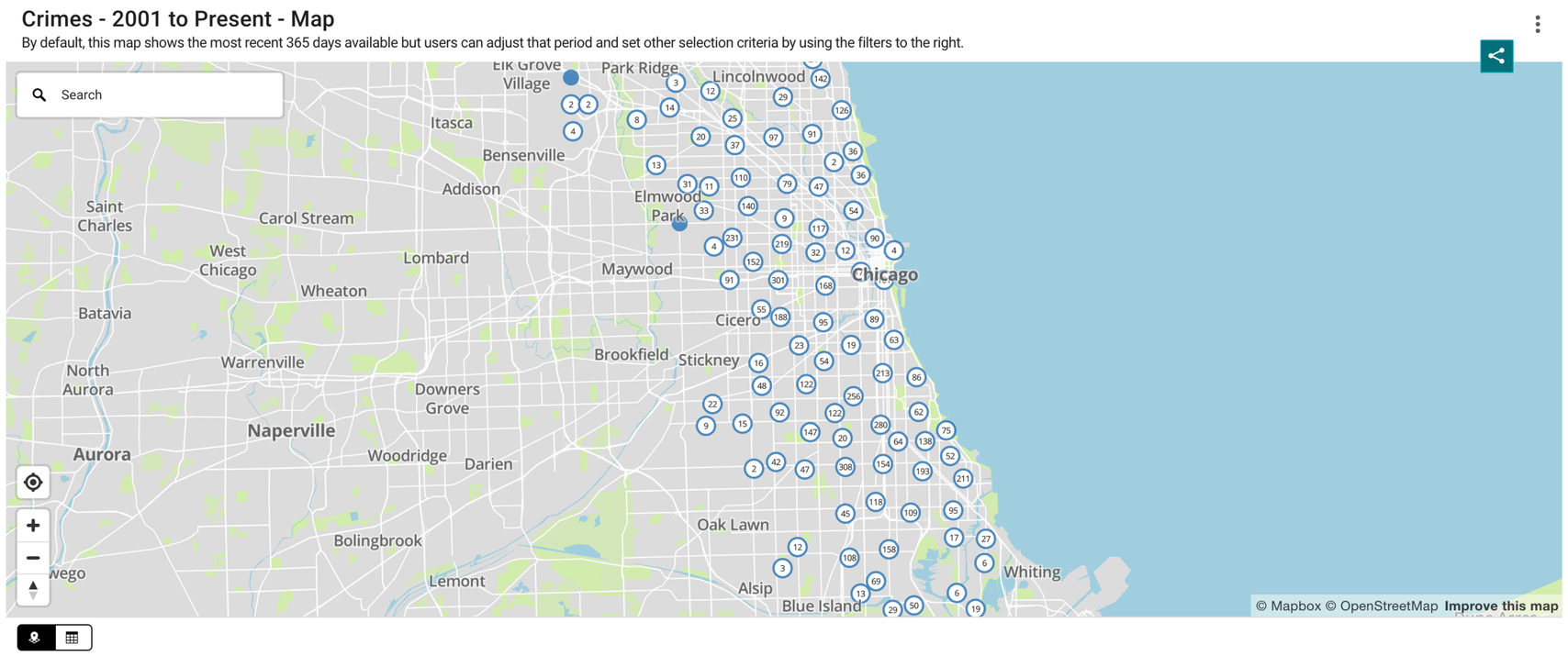
OpenAI POC

OpenAI POC

Tools and Data

pandas

DuckDB



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 {} ({}) \n
    """

    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

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 {} ({}) \n
    """

    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

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 {} ({}) \n
    """

    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

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 {} ({}) \n
    """

    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_name

    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_name

    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_name

    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 0xffff4035c2e0>
SELECT COUNT(*) FROM chicago_crime WHERE "Arrest" = true;
{
  "id": "chatcmpl-8CglDSYrYSzUyOZP25MjNGGd8VC6b",
  "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 0xffff4035c2e0>
SELECT COUNT(*) FROM chicago_crime WHERE "Arrest" = true;
{
  "id": "chatcmpl-8CglDSYrYSzUyOZP25MjNGGd8VC6b",
  "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 0xffff4035c2e0>
SELECT COUNT(*) FROM chicago_crime WHERE "Arrest" = true;
{
  "id": "chatcmpl-8CglDSYrYSzUyOZP25MjNGGd8VC6b",
  "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

- 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
- OpenAI example - https://platform.openai.com/examples/default-sql-translate

Questions?

Thank You!