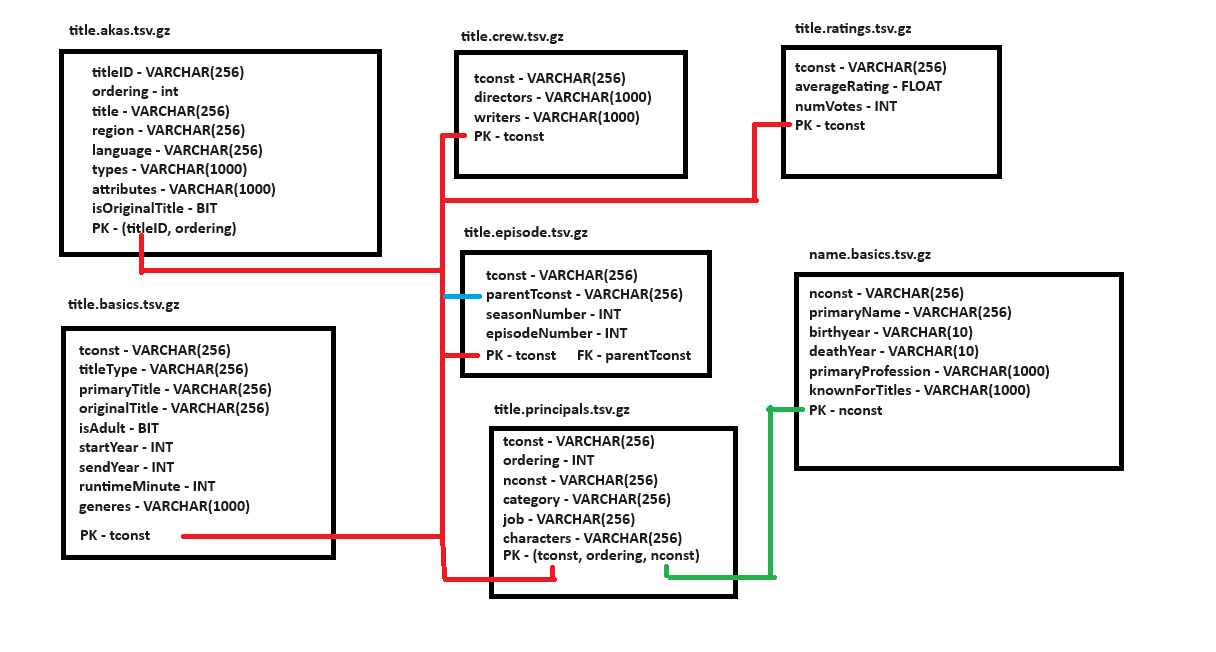
**Part 1**

**Frameworks**: Pytorch, colab, tensorflow, huggingface (transformers), Python

**Algorithms & models**: Defog/sqlcoder – 7bil parameters transformer model that can perform text-to-sql tasks

**Databases**: IMDb Non-Commercial Datasets

Architecture:



**Description of any preprocessing stage:**

I needed to provide the model with the code to start the DB, and description of each attribute and constraint.

for that I’ve draw the image of the DB schema and wrote a code to generate this DB according to it.

fine tuning the model requires payment.

text preprocessing operations are not required for this task.

**the code to generate the schema: (the algorithm assume that the DB was generated from this code)**

CREATE TABLE title\_ratings(

tconst varchar(256), -- alphanumeric unique identifier of the title

averageRating float, -– weighted average of all the individual user ratings

numVotes int, -- number of votes the title has received

primary key (tconst)

);

CREATE TABLE title\_principals(

tconst varchar(256), -- alphanumeric unique identifier of the title

ordering int, -- a number to uniquely identify rows for a given titleId

nconst varchar(256), -- alphanumeric unique identifier of the name/person

category varchar(256), -- the category of job that person was in

job varchar(256), -- the specific job title if applicable, else '\N'

characters varchar(256), -- the name of the character played if applicable, else '\N'

primary key (tconst, oredring, nconst)

);

CREATE TABLE title\_episode(

tconst varchar(256), -- alphanumeric identifier of episode

parentTconst varchar(256), -- alphanumeric identifier of the parent TV Series

seasonNumber int, -– season number the episode belongs to

episodeNumber int, -– episode number of the tconst in the TV series

primary key (tconst)

);

CREATE TABLE title\_crew(

tconst varchr(256), -- alphanumeric unique identifier of the title

directors varchar(1000), -- director(s) of the given title

writers varchar(1000), -– writer(s) of the given title

primary key (tconst)

);

CREATE TABLE title\_basics(

tconst varchar(256), -- alphanumeric unique identifier of the title

titleType varchar(256), -– the type/format of the title

primaryTitle varchar(256), -– the more popular title / the title used by the filmmakers on promotional materials at the point of release

originalTitle varchar(256), -- original title, in the original language

isAdult - BIT, -- 0: non-adult title; 1: adult title

startYear int, -– represents the release year of a title. In the case of TV Series, it is the series start year

endYear int, -– TV Series end year. ‘\N’ for all other title types

runtimeMinutes int, -– primary runtime of the title, in minutes

genres varchar(1000), -- includes up to three genres associated with the title

primary key (tconst)

);

CREATE TABLE title\_akas(

titleID VARCHAR(256) not null, -- a tconst, an alphanumeric unique identifier of the title

ordering int, -– a number to uniquely identify rows for a given titleId

title VARCHAR(256), -– the localized title

region VARCHAR(256), -- the region for this version of the title

language VARCHAR(256), -- the language of the title

types varchar(256), -- Enumerated set of attributes for this alternative title. One or more of the following: "alternative", "dvd", "festival", "tv", "video", "working", "original", "imdbDisplay".

attributes varchar(256), -- Additional terms to describe this alternative title, not enumerated

isOriginalTitle BIT, -– 0: not original title; 1: original title

primary key (titleID, ordering)

);

CREATE TABLE name\_basics(

nconst varchar(256) NOT NULL, -- Unique ID for each person

primaryName varchar(256), -- Full name of the person

birthYear int, -- Birth year of the person

deathYear int, -- Death year of the person

primaryProfession varchar(1000), -- List of top-3 professions of the person

knownForTitles varchar(1000), -- List of Titles the person is known for

PRIMARY KEY (nconst)

);

-- title\_ratings.tconst can be joined with title\_principals.tconst

-- title\_principals.tconst can be joined with title\_episode.tconst

-- title\_principals.tconst can be joined with title\_episode.parentTconst

-- title\_principals.nconst can be joined with name\_basics.nconst

-- title\_crew.tconst can be joined with title\_principals.tconst

-- title\_basics.tconst can be joined with title\_principals.tconst

-- title\_akas.titleID can be joined with title\_principals.tconst

**Description of the entire question answering flow:**

**How can I do this task?**

In my opinion the most reliable way to get information from the tables without asking it implicitly using SQL queries is to transfer the natural language questions into SQL queries and then we can return the response of the generated query on the DB.

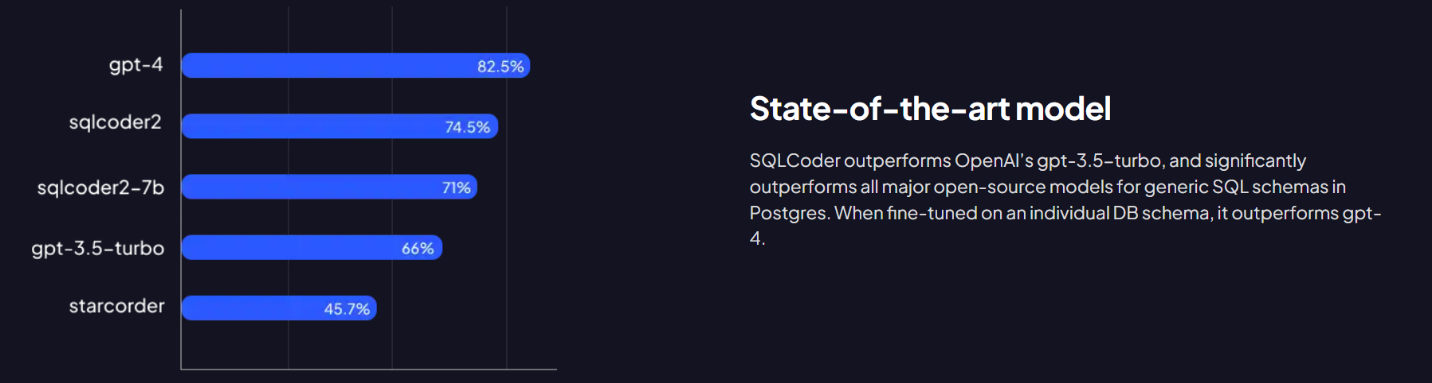
After more research I’ve found the option of embeddings / vector for indexing (like pinecone DB), I’m not going to implement it since I’ve already implemented sqlcoder.

SQL Auto Vector Query Engine - This query engine allows you to combine insights from your structured tables with your unstructured data. It first decides whether to query your structured tables for insights. Once it does, it can then infer a corresponding query to the vector store to fetch corresponding documents.

https://gpt-index.readthedocs.io/en/v0.7.6/examples/query\_engine/SQLAutoVectorQueryEngine.html

I’ve considered openAI at first, but after more search I’ve found sqlcoder’s API in huggingface, which is a SOTA solution for this specific problem (Text-To-SQL), especially when fine-tuned.

**Why defog/sqlcoder?**



**How to use defog/sqlcoder:**

Defog needs 2 arguments to create an SQL query:

1. Natural language question about the DB.
2. The DB Schema, its recommended to add description of each property in every table and to note the relationship between keys.

Given this 2 arguments, we can generate a query

**How did I create the DB schema:**

I’ve been reading the entire list of properties from the: <https://developer.imdb.com/non-commercial-datasets/>

And made assumptions based on Normalization rules, so the algorithm gets a good POV about the DB Schema.

**Why not gpt-4?**

My session with OpenAI have reached the maximum number of possible queries and I’m not interested in paying to use OpenAI’s interface.

Also, if we choose paying, we better fine tune sqlcoder2, and it should outperform gpt-4.

For booking a demo of the premium solution, which include fine tuning:

<https://calendly.com/d/2pk-z5w-xfb>