

An Overhaul of Traditional Web Application Architecture using Data-Less LLM

Proposal at a Glance

Since the past few years, Machine Learning has greatly contributed in overhauling the Business landscape. Right from Data Analysis using simple regression & classification models, to neural networks, transformers, Large Language Models, AI Agents etc, the use of technology in business has been to maximize final output, both on monetary and non-monetary aspects. In this paper, we propose a novel construction and business use-case of a LLM which has the potential to redefine how applications work

Objectives

We want to achieve two broad objectives –

- Minimize & Simplify Web Applications through LLM's
- Minimize LLM and operational costs

Solution

A simple user interface consisting of a text box where the user inputs a request. The request is passed to a LLM which processes & generates a database query (assumed SQL henceforth) which is sent to the database and the data is sent to the user without passing through the LLM. While it may sound like a chatbot, the LLM significantly differs in the following aspects –

- The LLM only generates the SQL query and posts it to the database. It does NOT access the database data, neither it accesses the output produced from the database
- The LLM is trained only on the SQL schema of the database tables, and NOT on any other data, be it the front-end & back-end codebase, stored procedures, or actual data
- The LLM does not interact with the user, it only receives requests, it doesn't respond to it.
- Since the LLM only generates SQL query, it can be trained only using SQL data, thus minimizing costs, & the post-training for tailoring to that specific business-case is only dependent on the company's Database SQL Schema, ensuring a Data-Less LLM from business data usage perspective.
- The LLM is flexible, easily adaptable to most business cases without major modifications.

In short, we want to build a LLM which is not trained on company-specific business or codebase data but performs the same as a LLM or chatbot would have done if trained on company-specific data.

Model Construction

We chose the standard GPT architecture for creating a text-to-SQL LLM, consisting of the Bigram Language Model. However, the real trick in helping the model outperform is the training methodology adopted for this LLM.

We chose a SQL dataset consisting of 100k SQL queries for training of various domains and complexities. However, instead of choosing all column inputs, only three important were chosen – User Prompt, SQL Schema, SQL query. With this we achieved training dataset of 49 million characters. The smaller data size enabled us to choose character-level encoding to ensure deeper context embedding instead of using tokenizers.

Why train on SQL Schema & not the actual data? –

- The actual data is huge, and computationally intensive and expensive both for pre-training and post-training stages. Using SQL schema needs negligible computation, still producing the correct output.
- The actual database or codebase can change massively, making it important to re-train the model on newer data, however, the SQL Schema of databases rarely changes, and even if they do, the additional post-training needed is absolutely minimal compared to the actual database change. This advantage makes this model stable, less computational.
- Companies value data privacy & security above everything. Companies are apprehensive about using LLM's that uses company data to train on and reject this option. Hence, by not using nor accessing company data, and neither by intercepting outputs, we provide a credible, safe approach to using LLM's to solve business problems.

The model consists of 1.75 M parameters, (the maximum I can run on my laptop) and also consists of an additional post training front-end option wherein additional data can be uploaded from frontend to further train the model on-the-go. Checkpoints have been implemented in the model to save model parameters, which are replaced with the existing parameters every time the backend is started to avoid re-computation of all parameters. The model also features more updations from the standard architecture, such as using cosine learning decay and weight decay for losses optimization, complete CUDA compatibility, a customized output query termination mechanism to ensure no unnecessary outputs or descriptions.

To ensure the model functions correctly, slight modifications have to be done. When the user inputs prompt, it must be appended with the database schema before the prompt is consumed by the LLM. This can be done in multiple ways. If schema rarely changes, it can be saved in the backend and appended directly, if not, it can be called using DESCRIBE function, stringified in prescribed way and appended to the prompt.

However, the unique training methodology also can be utilized in a different way – to generate SQL schemas for creating tables. Since the model is trained on a “Prompt + Schema + Query” framework, providing a Prompt but without appending Schema as proposed previously, will result in the model itself creating a Schema for the Prompt, as well as generating Query which will produce data as required, thus resulting in an end-to-end SQL model capable of write (CUD) operations and read (SELECT) operations.

Results

Example of a training result focused on write operations –

Prompt – Store properties in property table

Output - CREATE TABLE properties (id INT, property_id INT, property_name VARCHAR(50), property_id INT);

Example of a training result focused on read operations –

Prompt - How many accidents have been recorded for SpaceX and Blue Origin rocket launches?# CREATE TABLE Accidents (id INT, launch_provider VARCHAR(255), year INT, description TEXT); INSERT INTO Accidents (id, launch_provider, year, description) VALUES (1, 'SpaceX', 2015, 'Falcon 9 explosion'), (2, 'Blue Origin', 2011, 'Propulsion system failure'), (3, 'SpaceX', 2016, 'Falcon 9 explosion');\$

Output - SELECT COUNT(*) FROM Accidents WHERE launch_provider = 'SpaceX' AND year = 2015;

For more results, please scroll to the end.

The model showed a close $\pm 3\%$ error deviation between training and validation datasets at all intervals, indicating proper fitting. While the model still has room for improvements, with just 1.75M parameters, the model for now has showed promising results. Since the model is trained on a diverse but structured, consistent data, the model encodes the nuances and logic on a domain-independent level, ensuring that with business-specific requirements post training, the model can generate outputs consistent with user expectations, as shown in training examples where random domain related questions were asked.

n'th iteration	Training loss	Validation loss
400	1.86	1.86
800	0.97	0.97
1200	0.67	0.67
1600	0.57	0.57
2000	0.526	0.535
2400	0.494	0.499
2800	0.463	0.470
3200	0.440	0.441
3600	0.418	0.429
4000	0.405	0.408
4400	0.398	0.405
4800	0.395	0.399
5000	0.393	0.399

What does a data-less LLM mean? The LLM is trained on data of a .txt file of 48 MB, which is negligible compared to hundreds of GB's of data used by advanced LLM's but still showing promising results. Also, the LLM does not use any company-specific business data or codebase data for reinforcement learning, which literally makes it a Data-less LLM.

Post-training methodologies

For company-specific use-cases, it consists of two stages –

- Using the Database schema to prompt a general-use case LLM to generate dataset for training our LLM in the prescribed format.
- Using the Database schema to generate human-labeled data in the prescribed format.

However, a certain deviation can also be implemented with respect to the Database schema in the post-training methodology. While it is optional but if the database schema also consists of unique records from columns serving as primary keys for normal stored procedures, the model can learn better the context between the selection of columns for capturing the conditional data provided in the prompt. Example –

```
SELECT 'product_code', product_code FROM investor_transactions GROUP BY product_code
UNION ALL
SELECT 'tran_type', tran_type FROM investor_transactions GROUP BY tran_type
UNION ALL
SELECT 'investor_code', investor_code FROM investor_transactions GROUP BY investor_code
```

Challenges with our model

- Unwanted self-attention mechanism – Since the model learns with a fixed block size which is almost always higher than the length of an average SQL training query, the model encodes context from unrelated previous SQL queries in order to process next SQL query. While this is somewhat acceptable in understanding the schema and structuring a query, it creates unwanted mixing of business domain-specific aspects of the query.

Example – Assume the yellow highlighted part is used as a block for iteration through the model, clearly it is mixing the context as they are two different queries

List all the unique equipment types and their corresponding total maintenance frequency from the equipment_maintenance table.##

```
CREATE TABLE equipment_maintenance (equipment_type VARCHAR(255), maintenance_frequency
INT);$
SELECT equipment_type, SUM(maintenance_frequency) AS total_maintenance_frequency FROM
equipment_maintenance GROUP BY equipment_type;;
```

How many marine species are found in the Southern Ocean?##

```
CREATE TABLE marine_species (name VARCHAR(50), common_name VARCHAR(50), location
VARCHAR(50));$
SELECT COUNT(*) FROM marine_species WHERE location = 'Southern Ocean';;
```

To try avoid this mechanism, we created an independent GPT model, however, with dynamic block size which only takes a single training query and updates model parameters accordingly. Even though the block size changes with query, the model parameter dimensions are fixed. The model gave interesting results, with the first few output query characters being correct but with an increasing error in subsequent characters. We suspect this to be caused because of insufficient parameter weight optimizations towards the end indices of these weights caused due to a very small number of training queries which have a larger length. A dynamic block size model also means no batching of training or validation data is possible, resulting in sequential execution more or less. To mitigate this, we again tried an iteration of fixing block size but only filling in a single training query and filling the remainder of the training data and validation data block size with torch.tensor(0). However, due to the large variance in training query lengths, the sheer amount of torch.tensor(0) generate a conundrum in which the model is better off predicting the entire output query as torch.tensor(0) as it is likely to yield

lower loss, thus resulting in an inability to solve this issue. Nevertheless we also have a dynamic block-size model with us which probably can be used in future for some specialized tasks.

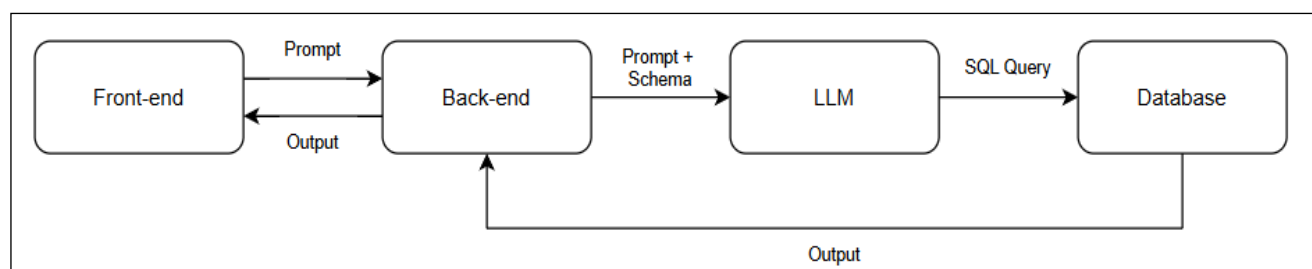
- **Risk Management** – To ensure that users access only the data intended for them, it is necessary to ensure the SQL queries contain access-level details as well. One solution for it would be to append user-level access details in the Prompt. Example – If an investor asks for their transactions, we can append the investor code so that it in the SQL query it always uses the investor code in the conditional clause or verifies the user level permissions and then generates the SQL query. While this can be mitigated upto some extent, hallucinations and data-checking still remain an unsolved puzzle. Data-checking can be validated upto a certain extent by pooling SQL queries through the same prompt through different general-purpose advanced models and computing quality score and only allowing queries above a certain threshold to be sent to the database for execution. For now, the model is not trained to handle out-of-scope random questions. However, if used for internal company work, these points minimize severity as the handler is explicitly informed not to use the model for out-of-scope tasks and failure to do so is the handler's liability. Still, a concrete solution is needed.

While scaling our model is possible, other alternatives can also be considered, such as advanced generic purpose models or specialized text-to-SQL models. They have many benefits, such as proven higher accuracy, easy implementation etc.

- Using advanced models such as ChatGPT 5, Gemini 3 etc. – They show superior accuracy, however they face following challenges in implementation –
 - Base models produce unnecessary explanations, resulting in excessive spend per output token, as high as 60%
 - Re-training them is computationally intensive and expensive
 - High context-size means model produces results based on previous prompts as well, leading to wrong data generation due to unwanted prompt mixing
- Specialized text-to-SQL models –
 - Reinforcement learning needs to be carried out in a specific way compatible with the original model training dataset
 - Relatively less but still computationally intensive & expensive for simpler operations
 - Unknown risk-mitigation frameworks

Integration in Web Applications

While the model used can be flexible, the intention is to devise a logic of functioning which is compatible and maximizes our dual objectives. Finally, a simple web application architecture looks like this –



Advantages –

- **Front-end:** Super-simple UI/UX, eliminating tens of modules, sections, designs, functionalities etc. Negligible front-end code, hardly more than 1-2 API's. Negligible Front-end dev(s) needed.
- **Back-end:** Simple one-way connection to Database, and another to the LLM, both independent and API's to GET and POST data back to front-end.
- **Model** – A LLM which needs no code or bug debugging, just training optimizations as and when needed
- **User** – Get customized data with a single prompt, eliminating the need for navigation or further processing in Excel, Word etc. due to fixed way of data generation in traditional applications

Business Use-Case Example

Assume an Investment Management company that has a back-office application for maintaining all investor records and performing day-to-day operations which usually consist of CRUD operations most of the times. The back-office usually consists of operations team and an IT team. If we implement the LLM, we can help optimize the back-office in the following ways –

- Operations team can generate any report, using any custom filters they want and eliminating the need to further process the data in Excel. They won't have the pain of navigating multiple features, making it easy for new-hires to learn fast. Headcount too can be optimized as a result of this. Any data bugs detected can be fixed directly through prompts. Datachecks can be executed via prompts. This functionality can be extended straight to the client as well, albeit with rigorous testing.
- IT Team will be heavily optimized. No UI/UX designers needed, no additional front-end coding required except to implement features not covered through LLM if any. Simple, clean front-end logic ensures no time or resources spent on code debugging. Same for backend. For database, a database administrator might be needed to ensure the CUD operations are appropriate and fix database-level bugs. Once implemented, minimal QA would be needed, except for when new functionalities are added.

While complete optimization may take time, it still presents a good and low-cost starting point approach for companies to start experimenting with this idea, considering it's unique architecture posing low risk for data mishaps.

Additional Results from the model –

Prompt: What is the number of medical supplies distributed by each organization, in East Africa, for the last 3 years, and the total cost of the supplies?
CREATE TABLE medical_supplies (supply_id INT, organization_id INT, location VARCHAR(255), supply_type VARCHAR(255), supply_cost DECIMAL(10,2), distribution_date DATE); INSERT INTO medical_supplies VALUES (1, 1, 'Country A', 'Medicine', 5000, '2020-01-01'); INSERT INTO medical_supplies VALUES (2, 1, 'Country A', 'Medical Equipment', 7000, '2021-01-01');\$

Output: SELECT organization_id, supply_type, SUM(supply_cost) as total_supplies FROM medical_supplies WHERE supply_type = 'Medical Equipment' GROUP BY organization_id, supply_type;

Prompt: What is the total budget allocated to public services in the state of New York?
CREATE TABLE budget_allocation (state VARCHAR(20), category VARCHAR(20), budget FLOAT); INSERT INTO budget_allocation (state, category, budget) VALUES ('New York', 'Education', 15000000), ('New York', 'Healthcare', 20000000), ('New York', 'Transportation', 10000000), ('New York', 'Infrastructure', 12000000); CREATE TABLE total_budget (state VARCHAR(20), total_budget FLOAT); INSERT INTO total_budget (state, total_budget) VALUES ('New York', 200000000);\$

Output: SELECT SUM(budget) FROM budget_allocation WHERE state = 'New York';

Prompt: What is the maximum energy consumption by a green building project in Asia?
CREATE TABLE green_buildings (id INT, name VARCHAR(50), country VARCHAR(50), energy_consumption INT); INSERT INTO green_buildings (id, name, country, energy_consumption) VALUES (1, 'GreenHub', 'India', 1200), (2, 'EcoTower', 'China', 1500), (3, 'SolarVista', 'Japan', 1800);\$

Output: SELECT MAX(energy_consumption) FROM green_buildings WHERE country = 'Asia';

Prompt: Which athletes have the most gold medals in the Winter Olympics?
CREATE TABLE winter_olympics (athlete TEXT, gold INT, silver INT, bronze INT, year INT);\$

Output: SELECT athlete, COUNT(*) as num_olympics FROM winter_olympics WHERE year = 2021 GROUP BY athlete;