Project Report

On

LLM Based Apache Spark Optimization

Submitted
In partial fulfilment
For the award of the Degree of

PG-Diploma in Big Data Technology (PG-DBDA)

C-DAC, ACTS (Pune)

Guided By:	Submitted By:
Mr Shivakarthik S.	Darshan Mhatre– PRN 240840125011
Mr Pankaj Patil	Pavan Patil- PRN 240840125033
	Rajwardhan Shinde– PRN 240840125050
	Soham Kulkarni– PRN 240840125052

Centre for Development of Advanced Computing(C-DAC), ACTS

(Pune- 411008)

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Mr Shivakarthik S. and Mr. Pankaj Patil for their guidance and support throughout this project. I would also like to thank my peers and family for their encouragement. Special appreciation goes to the providers of the dataset used in this study. Additionally, I extend my gratitude to CDAC Pune for their training and staff support, which significantly contributed to the completion of this project.

ABSTRACT

The increasing use of big data analytics demands efficient query execution for large datasets. Apache Spark, a widely used big data processing framework, can be further optimized using AI-driven techniques. This project introduces an LLM-based SQL query optimization system, where a large language model (LLM) translates user prompts into SQL queries. These queries are then optimized and executed on Spark tables to improve performance. The system is built using Flask for backend processing and provides a user-friendly front-end interface for seamless interaction. By integrating AI-powered query generation and execution, this project enhances efficiency, scalability, and accessibility, making data analysis faster and more effective.

Table of Contents

Contents

Chapter 1

Introduction

Introduction

This project presents an efficient and intelligent approach to optimizing SQL query execution using a Large Language Model (LLM) and Apache Spark. By integrating the Ollama model to convert natural language prompts into SQL queries, the system enables users to interact seamlessly with large datasets without requiring advanced database knowledge. The Flask web application facilitates smooth communication between the user, the model, and the Spark engine, ensuring an end-to-end streamlined workflow.

The use of Apache Spark ensures high-performance query execution, making the system scalable and suitable for handling big data workloads. This project highlights the potential of AI-driven query generation in enhancing data retrieval efficiency and usability.

In the future, the system can be improved by incorporating more advanced SQL optimization techniques, expanding support for different database engines, and enhancing the user interface with better visualization tools. This project lays the foundation for intelligent data access, making complex data analytics more accessible and efficient.

In recent years, the integration of Artificial Intelligence (AI) with data processing systems has revolutionized how users interact with databases. Traditional database query languages like SQL require users to have technical expertise, making data retrieval challenging for non-experts. Large Language Models (LLMs) have emerged as a powerful solution to bridge this gap by automatically converting natural language queries into optimized SQL statements.

This project focuses on developing an LLM-based SQL query generation and optimization system using **Flask and Apache Spark**. The system enables users to input queries in plain language, which are processed by an **Ollama model** to generate SQL statements. These queries are then executed efficiently using **Apache Spark**, ensuring faster processing of large-scale data.

The proposed system enhances accessibility, reduces manual effort in query writing, and optimizes SQL execution for better performance. By leveraging **Flask** for web-based interaction and **Spark** for distributed query execution, this project aims to create a seamless and intelligent data retrieval system for users across various domains.

Objective and Specifications

- Optimize SQL Query Generation – Utilize an LLM model to convert user prompts into efficient SQL queries.
- Enhance Spark Performance – Implement query optimizations for better execution on Apache Spark.
- Seamless Integration – Develop a Flask-based system to process and manage queries efficiently.
- User-Friendly Interface – Create an intuitive front-end for easy interaction with the system.
- Scalability & Efficiency – Ensure the system handles large datasets effectively.

Specifications:

- OS: Windows/Linux
- Languages & Tools: Python, Flask, Apache Spark, Ollama
- Database: Spark SQL Tables
- IDE: VS Code / Jupyter notebook

Chapter 2

LITERATURE REVIEW

Literature Review

The integration of Large Language Models (LLMs) with SQL query generation and optimization has been an area of active research in recent years. Several studies highlight the significance of AI-driven approaches for improving database accessibility and performance.

1. **Natural Language to SQL Conversion**
   Research on text-to-SQL models, such as OpenAI's Codex and Google's BERT-based approaches, demonstrates how deep learning models can effectively convert natural language queries into structured SQL queries. These models improve user experience by allowing non-experts to interact with databases without requiring knowledge of SQL syntax.
2. **Query Optimization in Big Data Processing**
   Apache Spark has been widely adopted for large-scale data processing due to its distributed computing capabilities. Studies show that SQL query optimization techniques, such as query plan caching, adaptive execution, and cost-based optimization, significantly improve query performance in Spark-based systems.
3. **LLM-based Query Generation**
   Recent advancements in LLMs, such as GPT-based models, have enabled automated code generation, including SQL queries. Research indicates that these models improve accuracy through fine-tuning on domain-specific data, reducing errors and improving query efficiency.
4. **Flask for Web-based Query Execution**
   Several studies discuss the use of Flask as a lightweight framework for integrating AI models with web-based applications. Flask facilitates seamless communication between users, AI models, and databases, making it a suitable choice for building intelligent data access systems.

Research Gap

Despite significant progress, existing LLM-based SQL systems often struggle with query optimization and execution efficiency. Many models generate syntactically correct SQL queries but do not account for performance bottlenecks in big data environments. This project addresses this gap by integrating an LLM with Apache Spark for efficient query processing.

Compartion Report

Model	Exact Match	Edit Distance (Lower is Better)	Latency (Lower is Better)	Issues
Mistral	0	456	**53.73 sec (slowest)**	Extra text, unnecessary explanation
Llama3.2	0	**589 (worst)**	**31.09 sec (best)**	Syntax errors (`V VendortID` issue)
DuckDB-NSQL	0	**57 (best)**	33.65 sec	Closest to expected output

- None of the models produced an exact match.
- DuckDB-NSQL had the lowest edit distance (57), making it the most structurally accurate model.

Chapter 3

Methodology and Techniques

Methodology:

1. User Query Processing:
    - The user enters a natural language query through a Flask-based web interface.
2. LLM-Based SQL Generation:
    - The system utilizes a locally downloaded Ollama model to convert the user's prompt into an optimized SQL query.
3. Query Execution via Apache Spark:
    - The generated SQL query is sent to Apache Spark, which efficiently processes large datasets stored in Spark tables.
4. Result Retrieval & Display:
    - The processed results are retrieved from Spark and sent back to the Flask frontend for visualization.
5. Optimization & Refinement:
    - Query execution performance is analyzed, and SQL queries are optimized based on execution time and resource usage.

Techniques Used:

- **Large Language Model (LLM):** For natural language to SQL conversion.
- **Flask Framework:** Backend and frontend development.
- **Apache Spark:** Distributed query processing for handling large-scale data.
- **SQL Optimization:** Improving query execution efficiency using indexing, partitioning, and caching.
- **REST API Integration:** Connecting the Flask application with Spark SQL execution.
- **Frontend (HTML, CSS, JS):** For a user-friendly interface.

Chapter 4

Implementation

Frontend (User Interface)

- Built using **Flask with Jinja2 templates, HTML, and CSS**.
- Provides a simple input for users to enter queries.
- Displays query results in a tabular format.

Backend (Flask Server)

- Processes user queries and interacts with the **LLM and Apache Spark**.
- Converts natural language queries into **optimized SQL queries** using the **Ollama model**.

SQL Query Execution with Apache Spark

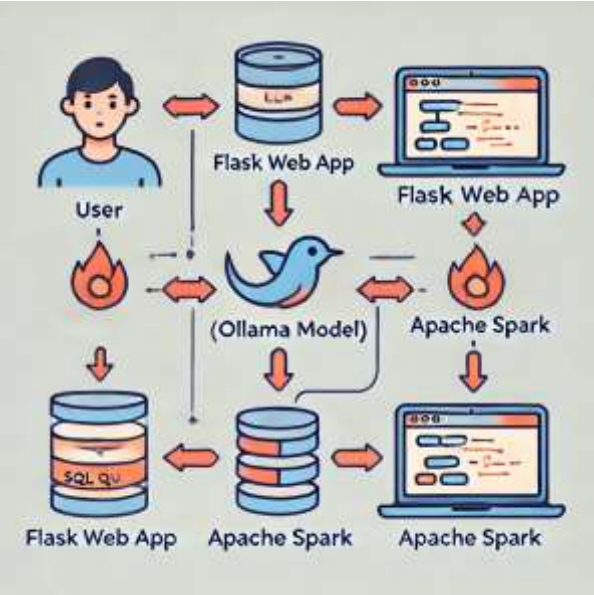
- The generated SQL query is executed on large datasets in **Spark tables**.
- Uses **indexing, partitioning, and caching** for faster query execution.

Result Retrieval & Display

- Fetches query results from Spark and returns them to the **Flask frontend**.
- Displays structured data in tables with sorting and filtering options.

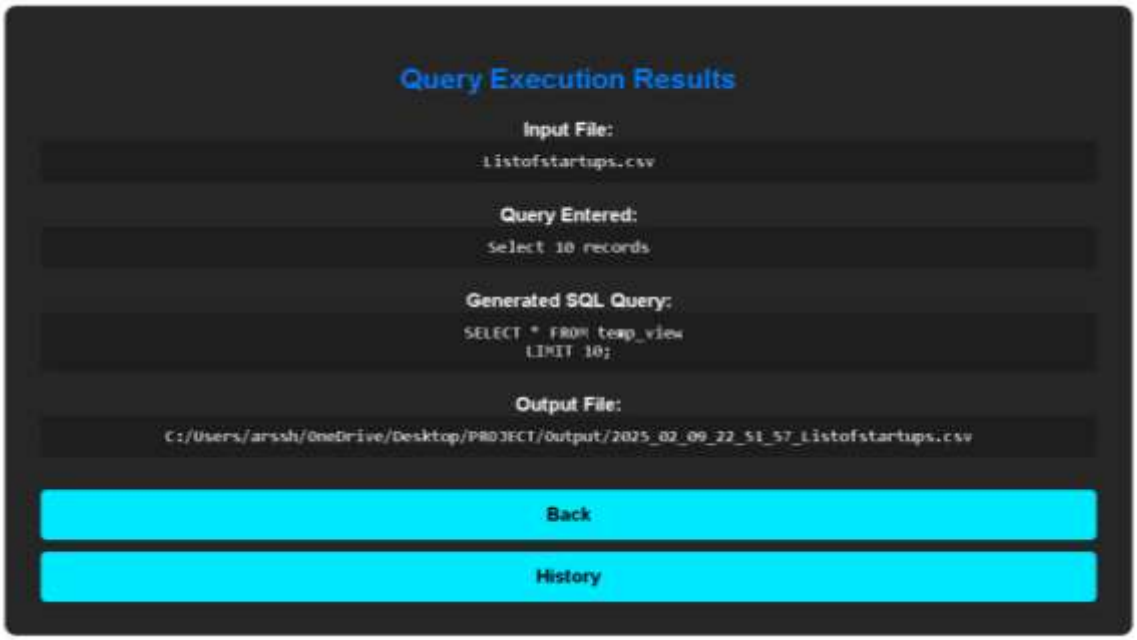
Architecture:

1. User Interface (Frontend - Flask)
    - Users input natural language queries through a Flask-based web interface.
    - The request is sent to the Flask backend for processing.
2. Backend Processing (Flask Server)
    - The Flask server receives the user query and sends it to the LLM model.
    - The Ollama model processes the query and generates an optimized SQL query.
    - The SQL query is then passed to Apache Spark for execution.
3. Apache Spark Query Execution
    - The generated SQL query is executed on large datasets stored in Spark tables.
    - Spark uses indexing, partitioning, and caching for query optimization.
    - The processed results are retrieved from Spark and sent back to Flask.
4. Result Display (Frontend)
    - The Flask frontend receives the structured query results.
    - The results are displayed in a user-friendly table format with sorting and filtering options.



Chapter 5

Results



Input File	Query Entered	Generated SQL Query	Output File
Listofstartups.csv	Select 10 records	SELECT * FROM temp_view LIMIT 10;	2025_02_09_22_51_57_Listofstartups.csv
Listofstartups.csv	Select 10 records	SELECT * FROM temp_view LIMIT 10;	2025_02_09_18_51_38_Listofstartups.csv
Listofstartups.csv	Select 2 records	SELECT * FROM temp_view LIMIT 2;	F:/Users/arvsh/OneDrive/Desktop/PROJECT/NewOutput/2025_02_09_18_43_36_Listofstartups.csv
Listofstartups.csv	Select 11 records	SELECT * FROM temp_view LIMIT 11;	F:/Users/arvsh/OneDrive/Desktop/PROJECT/NewOutput/2025_02_09_18_38_51_Listofstartups.csv
Listofstartups.csv	Select 1 records	SELECT * FROM temp_view WHERE Incubation_Center = '1'	2025_02_09_18_31_37_Listofstartups.csv
Listofstartups.csv	Select 5 records	SELECT * FROM temp_view LIMIT 5;	2025_02_08_21_23_01_Listofstartups.csv
Listofstartups.csv	select 10 records	SELECT * FROM temp_view LIMIT 10;	2025_02_08_19_41_48_Listofstartups.csv
Listofstartups.csv	Select 12 records	SELECT * FROM temp_view LIMIT 12;	2025_02_08_19_38_15_Listofstartups.csv

Next

Back

Error Details

Input File:

Listofstartups.csv

Table Schema:

```
Incubation_Center (string)
Name_of_startup (string)
Location_of_company (string)
Sector (string)
```

SQL Query:

```
SELECT * FROM temp_view WHERE Location_of_company = 'Raipur'
```

Error Message:

[UNRESOLVED_COLUMN.WITH_SUGGESTION] A column or function parameter with name `Location_of_company` cannot be resolved. Did you mean one of the following? [`Location of company`, `Incubation_Center`, `Name_of_startup`, `Sector`].; line 1 pos 31; 'Project [*] +- 'Filter ('Location_of_company = Raipur) +- SubqueryAlias temp_view +- View (`temp_view`, [Incubation_Center#17,Name_of_startup#18,Location of company#19,Sector#20]) +- Relation [Incubation_Center#17,Name_of_startup#18,Location of company#19,Sector#20] csv

Suggested Solution:

Error Analysis The error message suggests that the column `Location_of_company` cannot be resolved in the Spark query. However, there are similar columns with different names: `Incubation_Center`, `Name_of_startup`, and `Location of company`. Looking at the schema of the Relation data source, it appears to have columns with these names. **Possible Solutions** 1. **Check for typo**: Verify that the column name is indeed `Location_of_company` in the Spark query code. A simple typo might be causing this issue. 2. **Use a different alias**: If `Location_of_company` cannot be resolved, try using a different alias (e.g., `loc`) and update all occurrences of `Location_of_company` to use this new alias. ```sql Project [*] +- 'Filter ('loc = Raipur) +- SubqueryAlias temp_view +- View (`temp_view`, [Incubation_Center#17,Name_of_startup#18,loc#19,Sector#20]) +- Relation

Chapter 6

Conclusion

This project presents an efficient and intelligent approach to optimizing SQL query execution using a Large Language Model (LLM) and Apache Spark. By integrating the Ollama model to convert natural language prompts into SQL queries, the system enables users to interact seamlessly with large datasets without requiring advanced database knowledge. The Flask web application facilitates smooth communication between the user, the model, and the Spark engine, ensuring an end-to-end streamlined workflow.

The use of Apache Spark ensures high-performance query execution, making the system scalable and suitable for handling big data workloads. This project highlights the potential of AI-driven query generation in enhancing data retrieval efficiency and usability.

In the future, the system can be improved by incorporating more advanced SQL optimization techniques, expanding support for different database engines, and enhancing the user interface with better visualization tools. This project lays the foundation for intelligent data access, making complex data analytics more accessible and efficient.

Chapter 7

References

[1] https://medium.com/@suffyan.asad1/introduction-to-the-english-sdk-for-apache-spark-combining-the-power-of-apache-spark-and-llms-6a4ea74ff4f4

[2] https://aws.amazon.com/blogs/big-data/introducing-generative-ai-troubleshooting-for-apache-spark-in-aws-glue-preview/

[3] https://ollama.com/library/duckdb-nsql

[4] https://lydian-moonstone-474.notion.site/Can-an-LLM-optimize-my-Spark-query-for-me-133b61cc304880a29415f7a21dd327f8?pvs=4