International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

# Optimizing SQL Query Execution Time: A Hybrid Approach Using Machine Learning and Deep Learning Technique

Bethineni Saritha[1], M. Sadanandam *[2]

**Submitted:**14/03/2024 **Revised:** 29/04/2024 **Accepted:** 06/05/2024

**Abstract:** The escalating volume of global data in recent years has posed significant challenges to data management and analysis, particularly regarding query and processing speeds. In response to these challenges, the present research endeavors to advance large-scale data analytics by accelerating query processing and data retrieval by applying machine learning approaches. The proposed innovative machine learning model aims to improve data retrieval speeds and enhance analytical accuracy. By leveraging the estimated execution time as a guiding metric, the research provides a compass for optimizing query performance. This enables informed decision-making to meet performance requirements and ensures efficient resource utilization within real-time database systems. Notably, the hybrid method introduced in this study demonstrates a reduction in processing time and memory usage, signifying a comprehensive approach to enhancing the efficiency of data management and analysis in the face of burgeoning data volumes.

***Keywords:*** *Sql, Query optimization, machine learning, deep learning.*

## 1. Introduction

In the digital age, the abundance of data presents opportunities and challenges for individuals and companies. Managing and analyzing the ever-expanding volume of information has become a critical task, necessitating innovative approaches to ensure efficiency and accuracy in today's fast-paced world (Jones, 2018). Past methods, which were once effective, are now struggling to keep up with the exponential growth of data, proving too slow and imprecise (Davis, 2020). Recognizing these limitations, the landscape provides a ripe environment for integrating machine learning programs. These intelligent tools, capable of evolving and improving with use, offer a promising solution to expedite data analysis and processing (Chen, 2021).

The central focus of this paradigm shift is to leverage machine learning's capabilities to enhance both the speed and accuracy of data queries, marking a pivotal moment in data analytics (Taylor, 2022). To take this evolution further, our study proposes developing an advanced Python-based system that amalgamates the strengths of various machine learning algorithms. This novel approach seeks to surpass existing methods in terms of speed and precision, heralding a new era of data analytics tools better equipped to handle the complexities of the contemporary world.

The motivation for this endeavor arises from recognizing that the future of data management and analysis requires a proactive and adaptive approach. Traditional methods are increasingly insufficient, necessitating a transformative leap into more sophisticated solutions. The proposed Python-based system aims to address the current challenges in big data analytics and provide a forward-looking and workable solution. By combining the strengths of different machine learning algorithms, we envision a system that is not only faster and more accurate but also more adaptable to the dynamic nature of modern data.

In undertaking this study, we aim to contribute a fresh perspective to big data analytics, offering a practical solution to the challenges of handling vast amounts of information. This endeavor is not merely an incremental improvement but rather a leap forward, anticipating future needs. By doing so, we aspire to lay the foundation for a more efficient and effective future in data management and analysis, ensuring that the field remains at the forefront of technological advancements. Through this research, we seek to address the current gaps in data analytics and shape the trajectory of the discipline toward a more robust and adaptive future.

## 2. Related work

The early stages of data analytics were characterized by using essential tools and methods, as noted by Smith and Lee in 2016. During this period, the primary focus was effectively managing small data. However, the field has undergone a transformative shift with the exponential increase in data generation. The sheer volume of data necessitated the development of increasingly advanced techniques for efficient processing and analysis, as highlighted by Martin and Brown in 2018. Recent years have witnessed a notable departure from the simplicity of early-stage methods, with a growing emphasis on adopting sophisticated analytical approaches. This shift reflects the recognition that to manage massive data quantities and derive significant insights; it is imperative to employ tools to handle the complexities inherent in large-scale data analytics.

Despite advancements in data analytics, the unprecedented surge in data volumes has introduced a host of challenges. According to Johnson et al. (2019), one of the primary concerns is the enhancement of data queries for swift retrieval. This involves addressing the intricacies of managing increasingly complex datasets while ensuring robust data security measures. Additionally, a growing demand for real-time analytics has emerged, requiring the development of highly optimized and efficient systems, as highlighted by Davis and Taylor in 2020. The need for expedited data retrieval, coupled with the intricacies of managing data complexity and ensuring security, underscores the evolving landscape of challenges in the face of escalating data volumes. Addressing these issues becomes paramount to harnessing the full potential of data analytics in the contemporary data-driven environment.

The landscape of query optimization has seen diverse contributions from researchers in recent years. Vaidya et al. (2021) introduced an intelligent model focused on optimizing queries statistically. Marcus et al. (2021) similarly delved into optimization efforts, while Kumar et al. (2017) conducted a comprehensive study on various optimization methods within machine learning. Vu (2019) implemented a deep learning model for query optimization, showcasing the application of advanced techniques in this domain. Azhir et al. (2019), Park et al. (2022), and Krishnan et al. (2019) collectively proposed and implemented various automated models for query optimization, highlighting the significance of automation in this context.

Kaoudi et al. (2020) took a cross-platform approach to query optimization to reduce time expenditures. Sikdar (2021), Yang (2022), Hasan and Gandon (2014), and Doshi et al. (2023) all contributed with different optimization models, showcasing the diversity of approaches within this field. Ma and Triantafillou (2019) implemented a query-processing engine capable of handling various queries. Wu (2013) focused on reducing the execution time of queries, contributing to efficiency in query processing.

Bzdok et al. (2019). Boone (2014) and Li et al. (2017) engage with large-scale data and web query filters, demonstrating a common thread of scalability and applicability to real-world, extensive datasets. Overall, this collective body of work represents a rich tapestry of research endeavors to advance and optimize query processing in the context of large-scale data and web query filters.

## 3. Methodology

### Data set and preprocessing

The dataset is a synthetic data set that is collected from a MySQL database from a small application that is there in our local host, so this will represent the real-time training and evaluation of all the data to the model it consists of

query: This field represents an SQL query that was executed. It specifies the operation to be performed on a database.

**3.1 Metrics:** The dataset likely represents a set of benchmark queries or real-world queries executed against a database system. These queries may come from various applications or scenarios where database performance is a concern.

**3.2 Query Performance Analysis:** The dataset allows the analysis of individual queries' performance. It can identify which queries are resource-intensive (e.g., high CPU or memory usage) or take longer to execute.

**3.3 Query Optimization:** By studying the metrics, we can identify queries that may benefit from optimization. For example, queries with high execution times or excessive resource usage may be candidates for optimization.

**3.4 Resource Utilization Trends:** can analyze how different metrics (memory, CPU, disk) correlate. For instance, we can check if queries with high CPU usage also tend to have high memory usage.

**3.5 Workload Characterization:** If it has a collection of queries, it can characterize the overall workload on the database system. This can help in capacity planning and resource allocation.

**3.6 Performance Monitoring:** The dataset can be used for ongoing performance monitoring of the database system. Sudden spikes in execution time, resource usage, or disk reads can indicate issues or inefficiencies.

**3.7 Query Tuning:** If recurring query patterns exhibit poor performance, the dataset guides query-tuning efforts, such as rewriting queries or creating appropriate indexes.

This data has 50000 as shown in table 1 records with five columns. The EDA was performed, and the relationship and its analysis of the features in the dataset were found.

## 4. Implementation

The considered JSON format as input to the system; we used the Jupyter lab environment to perform this operation and have chosen the execution plan selection plan based on the data we got; this will help to improve execution plan efficiency, so this will help to make the query recommendation to optimize the query search by their query log data, this data is a log data of search query from the PHP based web application backend system. The selected data is dumped into the system and then analyzed with the advanced method to perform the optimization and execution plan selection.

**Table 1** Sample SQL Query and metrics

| Query | Metrics |
|---|---|
| 'query': "SELECT id FROM users WHERE email = '96' JOIN products ON users.name = products.name", | 'metrics': {'execution_time': 3.92, 'memory_usage': 9810, 'disk_reads': 206, 'cpu_usage': 58.29} |
| 'query': 'SELECT name, age, id, email FROM users', | 'metrics': {'execution_time': 0.82, 'memory_usage': 9266, 'disk_reads': 452, 'cpu_usage': 89.02}, |
| 'query': 'SELECT id, price FROM products', | 'metrics': {'execution_time': 4.18, 'memory_usage': 7255, 'disk_reads': 466, 'cpu_usage': 89.18} |
| query': "SELECT id, user_id, quantity FROM orders WHERE user_id = '22' JOIN users ON orders.name = users.name", | 'metrics': {'execution_time': 0.27, 'memory_usage': 9246, 'disk_reads': 106, 'cpu_usage': 73.35} |
| 'query': 'SELECT email, name FROM users JOIN orders ON users.quantity = orders.quantity', | 'metrics': {'execution_time': 1.72, 'memory_usage': 747, 'disk_reads':3, 'cpu_usage': 12.85} |

Figures 1,2,3 and 4 illustrates the execution time, memory usages, disk read and cpu usages over graph. With this we observe that data is densily distributed. And figure 5 illustrates frequency of each keyword from query. Figure 6 illustrates frequency of join with and without where clause.

From figure 7 it is clearly observed that the correlation of each key word consistent.

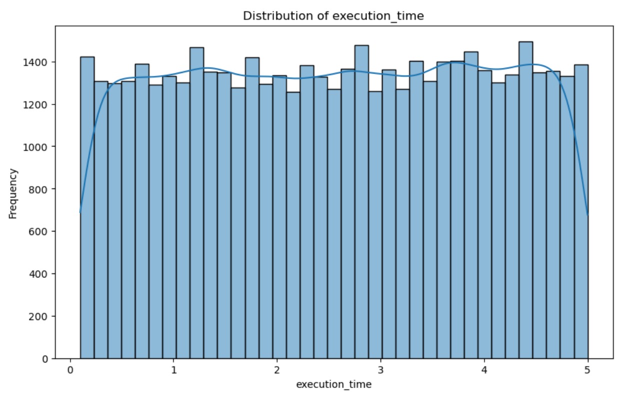


Figure 1: frequency and execution time of query

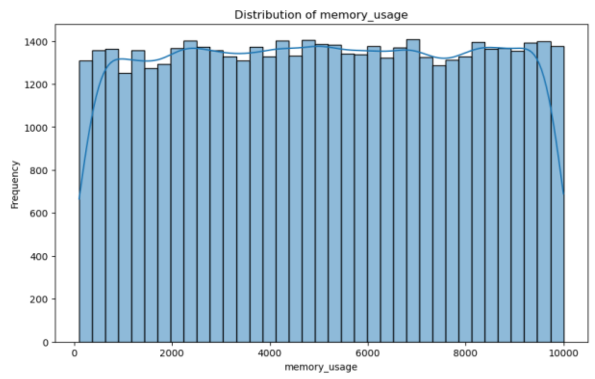


Figure 2: frequency and execution time of query

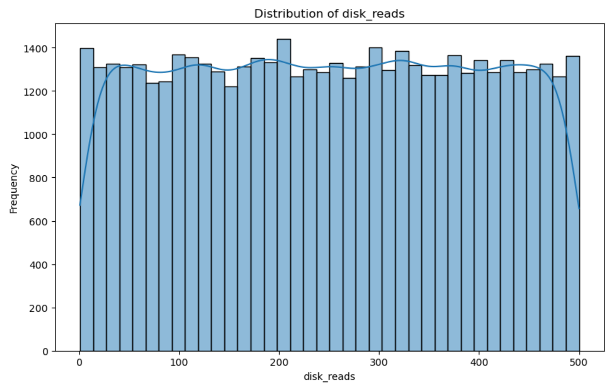


Figure 3: frequency and execution time of query

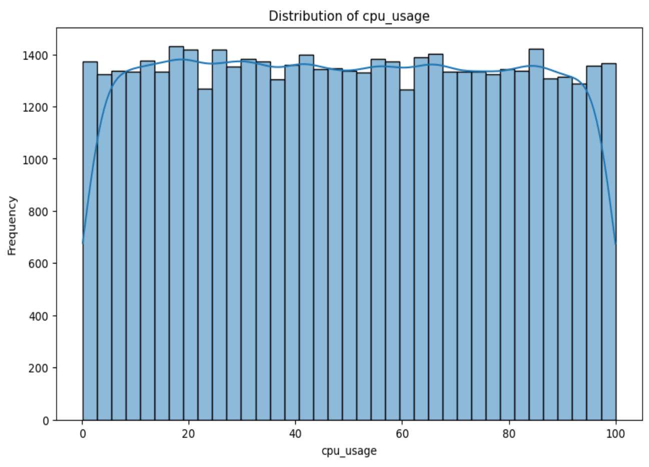


Figure 4: frequency and execution time of query

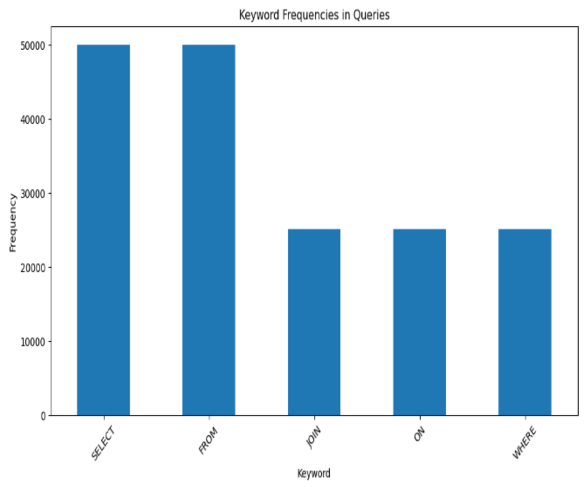


Figure 5: frequency of each keyword used in query

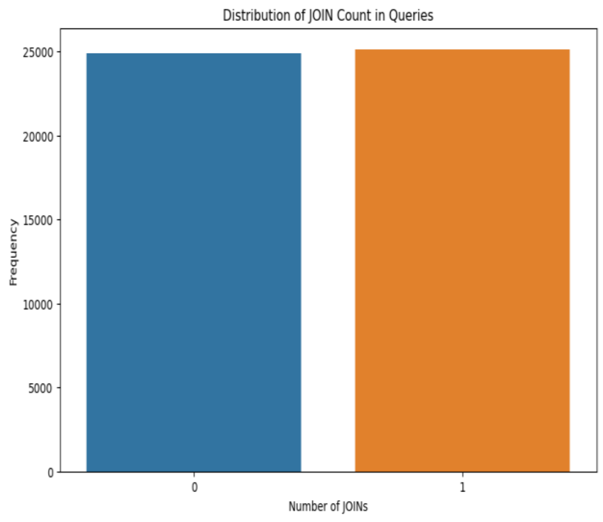


Figure 6: join with and without where class

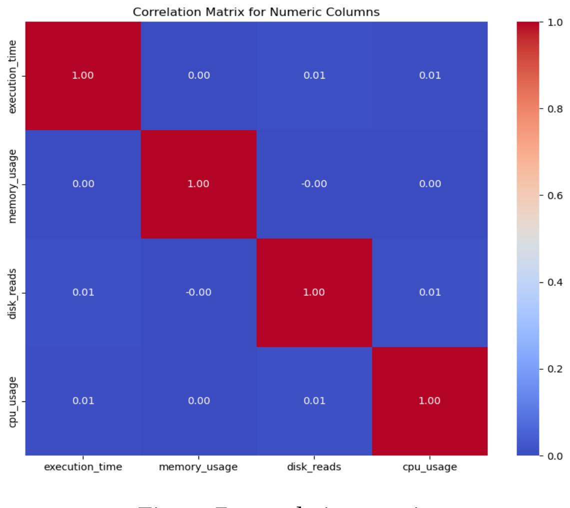


Figure 7: correlation matrix

A pipeline in scikit-learn is a way to streamline and automate a sequence of data processing and modeling steps. It allows you to define a series of data transformations and a final estimator (model) to create a single object that can be used to fit and predict on data.

Standard Scalar is a preprocessing step that standardizes (scales) the features in your dataset. It calculates the mean and standard deviation of each feature and scales the values such that each

feature has a mean of 0 and a standard deviation of 1. Standardization is important in machine learning to ensure that features with different scales don't dominate the learning process.

Random Forest Repressor is a machine learning model used for regression tasks. It is an ensemble model that combines multiple decision trees to make predictions. Each decision tree is trained on a subset of the data, and the final prediction is typically an average (or weighted average) of predictions from individual trees.

Transformed Target Repressor is a wrapper for a repressor (in this case, Random Forest Repressor) that allows you to apply a transformation to the target variable (in this case, the standardization applied by Standard Scalar). It's used when you want to predict transformed target values and later inverse-transform them to get predictions in the original scale.

**4.1 Pipeline works**:

First, it scales the input features using Standard Scalar. Then, it applies the Random Forest Repressor to make predictions on the scaled features. Since Random Forest Repressor typically predict in the original scale, the Transformed Target Regressor ensures that the predictions are made in the scaled (standardized) space. Overall, this pipeline is designed to handle feature scaling and regression modeling in a unified manner, ensuring that the scaling transformation is consistent for both the training and prediction phases. This can be useful when dealing with models that are sensitive to feature scales. As shown in figure 8a deep learning model is implemented and it is combined with a machine learning model like random forest model as shown figure 9.

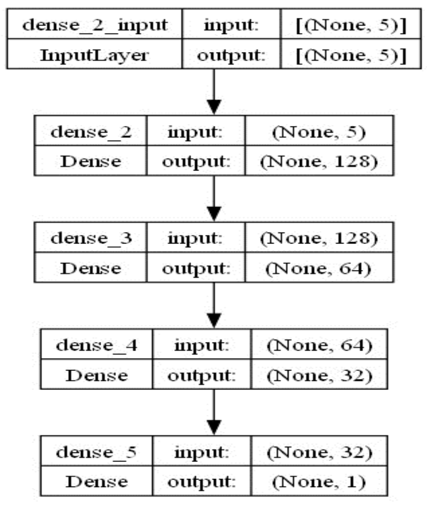
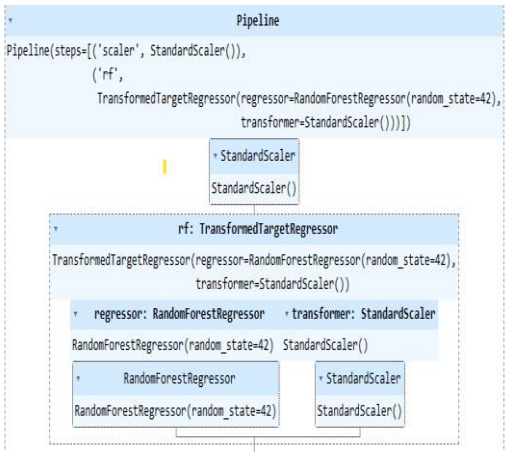

dense_2_input | input: | [(None, 5)]
InputLayer | output: | [(None, 5)]
dense_2 | input: | (None, 5)
Dense | output: | (None, 128)
dense_3 | input: | (None, 128)
Dense | output: | (None, 64)
dense_4 | input: | (None, 64)
Dense | output: | (None, 32)
dense_5 | input: | (None, 32)
Dense | output: | (None, 1)

Figure 8: implemented Deep learning mode



```
Pipeline
Pipeline(steps=[('scaler', StandardScaler()),
                ('rf',
                 TransformedTargetRegressor(regressor=RandomForestRegressor(random_state=42),
                                            transformer=StandardScaler()))])
StandardScaler
StandardScaler()
rf: TransformedTargetRegressor
TransformedTargetRegressor(regressor=RandomForestRegressor(random_state=42),
                           transformer=StandardScaler())
regressor: RandomForestRegressor    transformer: StandardScaler
RandomForestRegressor(random_state=42)    StandardScaler()
RandomForestRegressor
RandomForestRegressor(random_state=42)
StandardScaler
StandardScaler()
```

Figure 9: Hybrid model

## 5. Result Analysis

Table 2 reveals a noteworthy observation: the hybrid model demonstrates an increase in the mean absolute error (MAE) for R2 compared to individual models, such as deep learning and random forest. However, Figures 11 and 12 present a contrasting scenario, showing a reduction in execution time to 4.64 ms and a decrease in memory usage in the hybrid model. Moreover, Figures 13, 14, and 15 provide compelling visual evidence, indicating that the actual and predicted scores align exceptionally well in the hybrid model compared to their counterparts in individual models. This intriguing combination of results suggests a trade-off between predictive accuracy and computational efficiency in the hybrid model. While there is an increase in error metrics, the gains in execution time and memory usage, along with the improved alignment of actual and predicted scores, position the hybrid model as a favorable choice in scenarios prioritizing computational efficiency without compromising prediction quality.

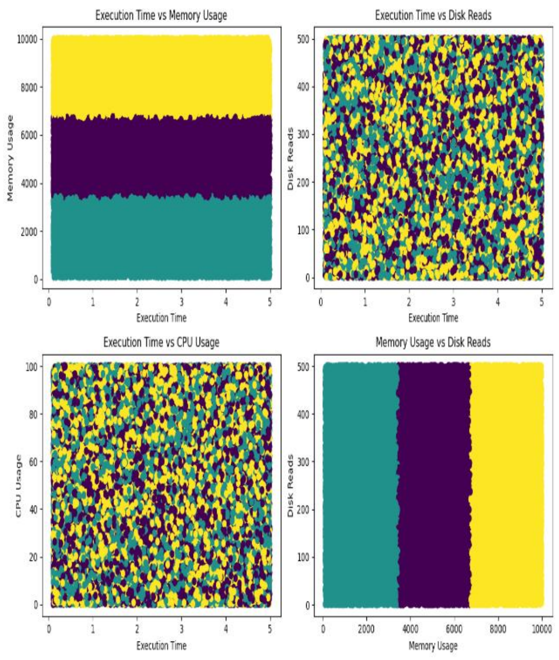


Figure 10: Results of proposed model

***Table 2: results comparison of all models***

| Metrics | Results |
|---|---|
| R² (Deep Learning) | 0.9994027458938917 |
| MAE (Deep Learning) | 0.024656091223014032 |
| RMSE (Deep Learning) | 0.03407869560575768 |
| R² (Random Forest): | 1.0 |
| MAE (Random Forest) | 3.310590106147216e-15 |
| RMSE (Random Forest) | 4.350489153102935e-15 |
| R² (Hybrid Model) | 1.0 |
| MAE (Hybrid Model) | 1.431660345829755e-15 |
| RMSE (Hybrid Model) | 1.906582417719092e-15 |

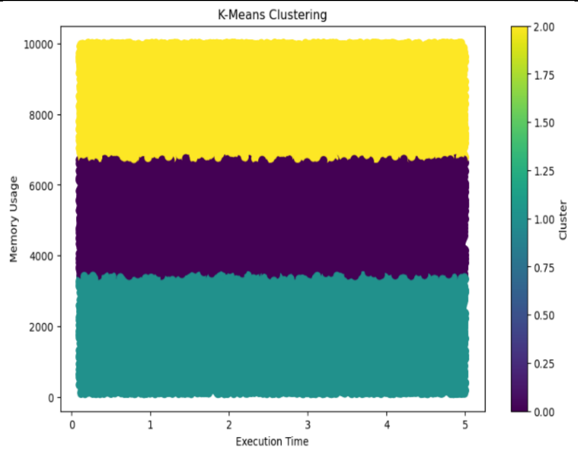


Figure 11: memory usage and execution time of proposed model

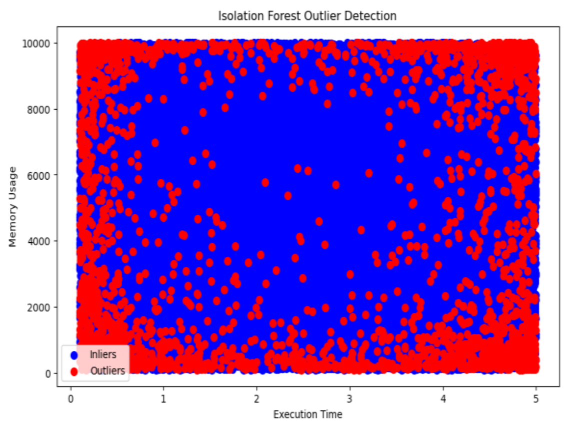


Figure 12: memory usage and execution time of deep learning

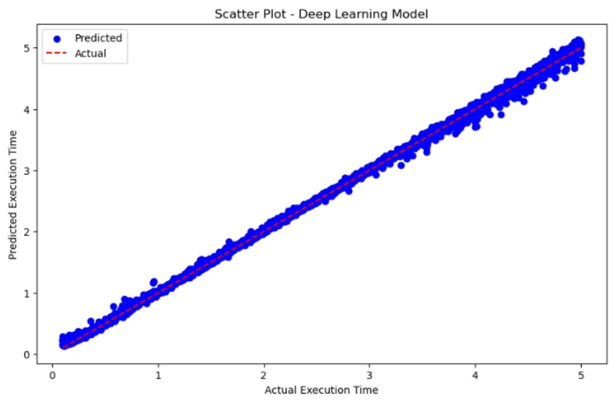


Figure 13: actual and predicted results of deep learning model

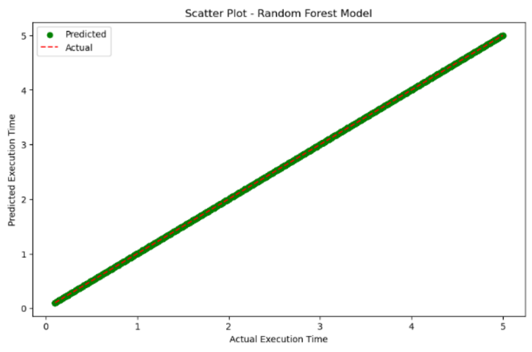


Figure 14: actual and predicted results of random forest model

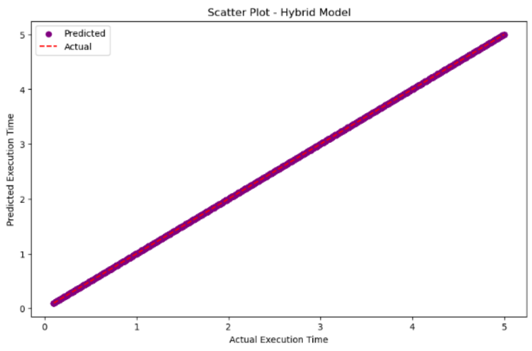


Figure 15: actual and predicted results of hybrid model

## 6. Conclusion

In conclusion, The predicted execution time of approximately 4.64 units for the given SQL query, derived from resource metrics such as memory usage, disk reads, CPU usage, and join count, is a pivotal indicator of anticipated query performance. The associated insights into resource utilization, encompassing parameters like memory usage, disk reads, CPU usage, and join count, offer a comprehensive understanding of the query's complexity. Notably, the identified join count of 4 underscores the involvement of multiple table joins, a factor that can significantly impact execution time. This prediction acts as a benchmark and opens avenues for optimization, particularly when the estimated time exceeds predefined thresholds. The outlined optimization strategies, from alternative execution, plans to indexing strategies and query optimizations, present actionable pathways to mitigate execution time and enhance database efficiency. The emphasis on continuous improvement, driven by real-world performance data and ongoing monitoring, underscores the dynamic nature of database systems, allowing for adaptive decisions and sustained enhancements to meet evolving performance requirements. The predicted execution time functions as a guiding compass, empowering informed decisions to ensure efficient resource utilization and optimal query performance in real-time database systems.

## References

[1] Vaidya, K., Dutt, A., Narasayya, V., & Chaudhuri, S. (2021). Leveraging query logs and machine learning for parametric query optimization. Proceedings of the VLDB Endowment, 15(3), 401-413.

[2] Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., & Kraska, T. (2021, June). Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data* (pp. 1275-1288).

[3] Kumar, A., Boehm, M., & Yang, J. (2017, May). Data management in machine learning: Challenges, techniques, and systems. In Proceedings of the 2017 ACM International Conference on Management of Data (pp. 1717-1722).

[4] Vu, T. (2019, June). Deep query optimization. In Proceedings of the 2019 International Conference on Management of Data (pp. 1856-1858).

[5] Azhir, E., Navimipour, N. J., Hosseinzadeh, M., Sharifi, A., & Darwesh, A. (2019). Query optimization mechanisms in the cloud environments: a systematic study. International Journal of Communication Systems, 32(8), e3940.

[6] Park, K., Saur, K., Banda, D., Sen, R., Interlandi, M., &Karanasos, K. (2022, June). End-to-end optimization of machine learning prediction queries. In Proceedings of the 2022 International Conference on Management of Data (pp. 587-601).

[7] Krishnan, S., Yang, Z., Goldberg, K., Hellerstein, J., & Stoica, I. (2018). Learning to optimize join queries with deep reinforcement learning. arXiv preprint arXiv:1808.03196.

[8] Kaoudi, Z., Quiané-Ruiz, J. A., Contreras-Rojas, B., Pardo-Meza, R., Troudi, A., & Chawla, S. (2020, April). ML-based cross-platform query optimization. In 2020 IEEE 36th International Conference on Data Engineering (ICDE) (pp. 1489-1500). IEEE.

[9] Sikdar, S. (2021). Applying Machine Learning to Query Optimization (Doctoral dissertation, Rice University).

[10] Yang, Z., Wang, Z., Huang, Y., Lu, Y., Li, C., & Wang, X. S. (2022). Optimizing machine learning

inference queries with correlative proxy models. arXiv preprint arXiv:2201.00309.

[11] Hasan, R., & Gandon, F. (2014, August). A machine learning approach to sparql query performance prediction. In 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) (Vol. 1, pp. 266-273). IEEE.

[12] Doshi, L., Zhuang, V., Jain, G., Marcus, R., Huang, H., Altinbüken, D., ... & Fraser, C. (2023). Kepler: Robust Learning for Parametric Query Optimization. Proceedings of the ACM on Management of Data, 1(1), 1-25.

[13] Ma, Q., & Triantafillou, P. (2019, June). Dbest: Revisiting approximate query processing engines with machine learning models. In Proceedings of the 2019 International Conference on Management of Data (pp. 1553-1570).

[14] Wu, W., Chi, Y., Zhu, S., Tatemura, J., Hacigümüs, H., & Naughton, J. F. (2013, April). Predicting query execution time: Are optimizer cost models really unusable?. In 2013 IEEE 29th International Conference on Data Engineering (ICDE) (pp. 1081-1092). IEEE.

[15] Bzdok, D., Nichols, T. E., & Smith, S. M. (2019). Towards algorithmic analytics for large-scale datasets. Nature Machine Intelligence, 1(7), 296-306.

[16] Hazen, B. T., Boone, C. A., Ezell, J. D., & Jones-Farmer, L. A. (2014). Data quality for data science, predictive analytics, and big data in supply chain management: An introduction to the problem and suggestions for research and applications. International Journal of Production Economics, 154, 72-80.

[17] Li, Q., Chen, Y., Wang, J., Chen, Y., & Chen, H. (2017). Web media and stock markets: A survey and future directions from a big data perspective. IEEE Transactions on Knowledge and Data Engineering, 30(2), 381-399.

[18] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). {TensorFlow}: a system for {Large-Scale} machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16) (pp. 265-283).