MATLAB-Based Lightweight Workload Prediction via Machine Learning Models in Distributed Systems

Ghadeer Al Jufout
Computer Engineering and Computer Science
California State University
Long Beach, CA, USA
ghadeer.aljufout01@student.csulb.edu

Simon Zhang
Computer Engineering and Computer Science
California State University
Long Beach, CA, USA
simon.zhang01@student.csulb.edu

Hailu Xu
Computer Engineering and Computer Science
California State University,
Long Beach, CA, USA
hailu.xu@csulb.edu

Saleh Al Jufout
Electrical Engineering
California State University, Long Beach
Long Beach, CA, USA
saleh.aljufout@csulb.edu

Abstract— **This paper presents a dynamic selector model for data workload prediction. A main function that is responsible for selecting the most accurate Machine Learning Algorithm (e.g., Linear Regression, Support Vector Regression, and Random Forest Regression) has been developed. The selection of the algorithm is based on the runtime workload of the given data application ensuring an efficient and accurate development of the application. The selector model is parallelized and deployed in Docker containers which can speed up the runtime and improve the integrated performance of the system on a large scale. Evaluated results show significant improvements in scalability, versatility, and prediction accuracy by analyzing various data sets in different distributed environments.**

Keywords—machine learning, MATLAB, parallel computing, workload, docker container

I. Introduction

With big data growing at a very high rate, the need for Machine Learning tools has also increased. Such industries as finance, healthcare, and marketing are making great use of these tools in data analysis and predictions, turning out very valuable insights and outputs. In finance, ML models detect fraud and advise investors on financial matters [1]. In Healthcare, such models are used to monitor patient applications with large data sets and enable clinics to make better decisions [2]. In marketing, ML models optimize digital marketing campaigns better [3]. Despite the broad diffusion of ML models in a variety of applications, their adoption into distributed systems is challenging: scalability, latency, or performance efficiency.

In a recent study [4], researchers explored big data analytics and ML for their transformative potential in healthcare. The research mentioned how the two technologies are increasingly changing the nature of healthcare concerning patient health tracking, disease prediction, and treatment personalization. Big data analytics enables the early identification of health issues to allow proactive interventions. The study further accentuated the contributions these technologies have made toward the management of epidemics by corroborating data from disparate sources to predict and manage outbreaks like COVID-19. However, it's noted that some important concerns still need to be taken into consideration in terms of data privacy and rigorous validation before its use in clinical practice. Shetu *et al.* investigated the use of ML models in predicting CPU temperature and usage for small-scale distributed systems. Workload-based prediction models for CPU temperature and usage are beneficial for cooling management and workload distribution. The predicted values from their models are compared with real measurements for accuracy verification. Future work will focus on refining such models by considering additional parameters such as the external temperature, number of cooling fans, dynamic voltage and frequency scaling per core, and processor [5]. Dong *et al.*, in [6], examined some of the computational and technical advantages of the Random Forest algorithm run over a large data set. This study availed 167 million smart meter energy consumption observations in London and observed quite considerable computational efficiencies for AWS s3, EMR, MongoDB, and Apache Spark. It also emphasized that the scale of data and architecture is critical in optimizing ML models' performance, and small data sets may not obtain full benefits from distributed systems.

In this paper, we have implemented a dynamic selector model in MATLAB, used to select the most accurate machine learning algorithm depending on the workload characteristics of a dataset, whether it be Linear Regression, Support Vector Regression, or Random Forest. Moreover, we have integrated this parallel execution inside Docker containers for better performance of the system to reduce runtime overhead. Hence, lightweight and dynamic migration in various distributed environments can be supported. The parallel machine-learning processes running in large-scale data applications are more efficient and accurate as shown in the experimental evaluated results.

II. System Overview

A. System Workflow

The proposed system provides the overall framework of data prediction using ML algorithms in a distributed system as shown in Fig. 1. It dynamically chooses a suitable ML algorithm, either Linear Regression, Support Vector

Regression, or Random Forest Regression, based on the workload of the input data. This results in high efficiency and accuracy for the predicted data with the aid provided by distributed environments. The main components of the proposed system ensure high levels of accuracy and efficiency in prediction using data within distributed environments.

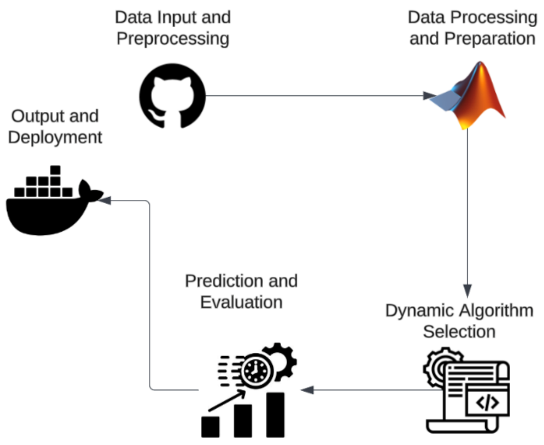
Data Input and Preprocessing
Data Processing and Preparation
Output and Deployment
Prediction and Evaluation
Dynamic Algorithm Selection

Fig. 1. Overview of the proposed system using MATLAB.

First, Data preprocessing refers to the stage where data is prepared and cleaned for use by machine learning algorithms. This stage plays a very crucial role in ensuring that there is high-quality input data fed into the ML model, hence minimizing the chances of errors and improving the general accuracy of the prediction. After the data preprocessing stage, the algorithm selector dynamically selects the most appropriate ML algorithm depending on data characteristics and workload. The reason for selecting the algorithm is that otherwise, it will be adjusted to the computational approach applied in issues for which it was prepared; Linear Regression, Support Vector Regression, or Random Forest Regression will then be applied to the input data set to achieve optimal results. The system then proceeds to model training. This is the stage where the algorithm previously picked as the best gets trained using the processed data. Effective training is central to developing a robust model with the ability to precisely interpret and, hence, predict data trends. In the last stage, the trained model is used for the prediction of new data. This stage involves taking the trained model and applying it against fresh data sets to derive predictions.

The system ensures the fact that the predictions are accurate and reliable according to the set high standards during training. Parallel computing is used to enhance the performance of the proposed system so that large data sets can be processed. The computational load is distributed across many processors with the aid of the MATLAB Parallel Computing Toolbox. This approach reduces runtime for both the model training and prediction phases.

In the data preprocessing step, parallel computing ensured that cleaning and preparation went through fast. The algorithm selector dynamically selected the most appropriate algorithm based on data characteristics, and at the model training stage, parallel processes were run to train the eligible algorithm on multiple data partitions simultaneously. This improved not only the speed of training but also that affecting the accuracy of the predictions to be made, with the robust development of a model. In the end, distributed processing was used to perform performance evaluation and validation to ensure that the system remained efficient and accurate with various workloads.

B. Algorithms

Linear Regression: This algorithm is used when predicting the value of a dependent variable based on an independent variable. It fits a straight line that minimizes dependencies between actual and predicted values. This model is simple and provides an easy way to interpret mathematical formulas to generate predictions [7].

Support Vector Regression: This algorithm uses the same principles as the Support Vector Machine but focuses more on predicting continuous data. Its main goal is to find the function representing the relationship between input and continuous target data while maintaining a low mean squared error (MSE) [8]. Random Forest Regression: This algorithm is based on decision trees. The more decision trees, the more accurate the prediction is. The trees run in parallel, meaning there is no interaction between any of these trees during their build [9].

C. Experimental Output

Base codes from the world's leading AI-powered developer platform, GitHub, were used which made it easier to build and modify the algorithm [10]. After reading and loading the data, it was split, so that 20% of the data was used for testing, while the remaining 80% was used for training the ML models. Based on this training and testing, the ML models made predictions of the continuous dependent variable. Then the mean squared error was calculated and a plot representing the accuracy and efficiency of the models in predicting the data was produced.

The datasets consisted of multiple variables related to the temperature prediction of the system based on various independent variables. The models predicted the system temperature versus the CPU total. Three different sizes of the same datasets were considered. A selector model that selects the most accurate algorithm for the given dataset based on its workload is integrated. For more efficient use of resources and less run time, parallel computing to run the three files simultaneously is integrated as well.

1) Data Loading: The data loading used is the same for all the algorithms as shown in Fig. 2.

```
% Load dataset
    opts = detectImportOptions(file_path);
    opts.VariableNamingRule = 'preserve';
    data = readtable(file_path, opts);
```

Fig. 2. Data loading using MATLAB.

As shown in the code, the first line determines how data should be imported from the file placed in the function. The second line preserves the variables' names to be shown on the visuals of the models. The last line reads the CSV file into a table.

2) Data Splitting: After loading the data, it is split into training and testing sets, specifically, 80% for training and 20% for testing, as shown in Fig. 3.

```
% Split the data into training and testing sets
    cv  =  cvpartition(size(X,  1),  'HoldOut',
0.2);
    X_train = X(training(cv), :);
    X_test = X(test(cv), :);
    Y_train = Y(training(cv), :);
    Y_test = Y(test(cv), :);
```

Fig. 3. Data splitting using MATLAB.

The same methodology was used for all the algorithms. The goal behind splitting the data in such a way is to make sure the model has been trained with enough data to be able to produce expected data outputs when testing data is used.

3) Mean Squared Error: The computation of this error served as an indication of how much the models are accurate and efficient in predicting data as shown in Fig. 4. The lower the MSE, the more accurate the predictions are.

```
% Calculate Mean Squared Error
    mse = mean((Y_pred - Y_test).^2);
    disp(['Mean      Squared      Error:      ',
num2str(mse)]);
```

Fig. 4. Error calculation using MATLAB.

4) Visuals: Each of the algorithms has its plots depending on their representation of the relationship between the data. The quantile-quantile plot (Q-Q plot), as shown in Fig. 5, determines if the data is following the normal distribution. The blue plus sign represents the difference between the actual data and the predicted data while the red line represents the ideal fit the data is supposed to lie on for its assumptions to hold. In the support vector regression plot shown in Fig. 6, the blue dots represent the actual system temperate data, on the other hand, the red dots represent the predicted data. The black circles are the support vectors that are responsible for determining the regression line and margin. The solid black line is the regression line, and the dashed line is the margin within the model that is expected to fit the data. The blue dots, in Fig. 7, represent the actual system temperature data whereas the red zigzag lines represent the predicted data. The closer the predicted data is to the actual data, the more accurate the model is.

D. Parallel Computing Model

Integrating parallel computing in the proposed application speeds up the run time and makes efficient use of resources. Previously, the selector model only handled one file at a time, redirecting it to the most accurate algorithm for data prediction. Using parallel computing makes the proposed application more robust by allowing the ability to run all the files at the same time. MATLAB's Parallel Computing Toolbox has been used to create four MATLAB processors, allowing for different jobs to run simultaneously as shown in Fig. 8. The first line is responsible for initializing a parallel pool of four workers, in other words, four MATLAB processors on the local machine to execute parallel computing. The for-loop is responsible for going through each file, each time calling the parallel computing comparison function.

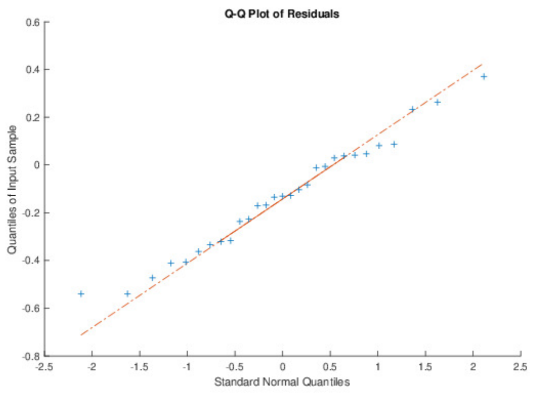

Fig. 5. Q-Q plot for linear regression.

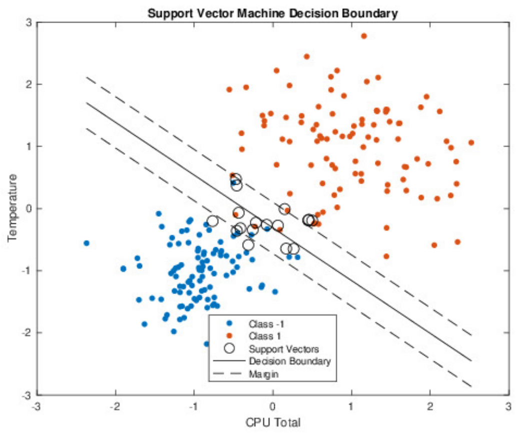

Fig. 6. Support vector regression plot. Please change the title into one sentence to explain the figure, not just the SVM plot.

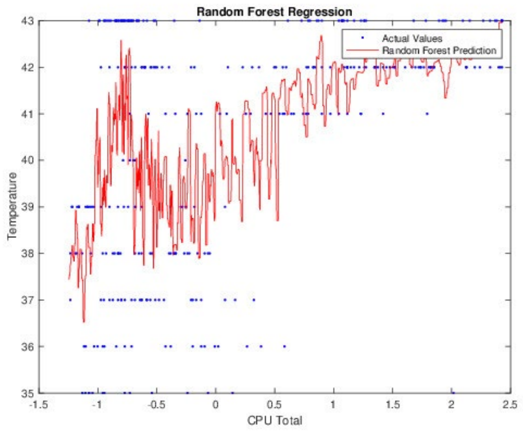

Fig. 7. Results of Random forest regression model.

The last line shuts down all the workers after the jobs are done, executing and therefore releasing the resources that were used.

```
% Initialize parallel pool once
   pool = parpool('local', 4);
% Run the function for each file
   for i = 1:length(files)
     parallel_computing_comparison(files{i},
target_column, x_feature);
   end
% Shut down parallel pool
   delete(pool);
```

Fig. 8. Parallel execution model using MATLAB.

E. Main Function

The main function that runs all the algorithms is shown in Fig. 9. The start is with command line arguments, allowing users to input the files they want to test out.

```
Function
main(file_path,target_column,x_feature)
```

Fig. 9. Main function using MATLAB.

Three parameters are used in function main, which allows users to input the file they want to test out. The correct number of arguments, three, is provided. If not, it throws an error with a usage message. The second parameter is the value of target datasets, for example, there might be a large dataset with different rows and columns, so the user can choose which row specifically to test out, and the last one is the comparison data that you want to compare with as shown in Fig. 10.

```
% Determine file size and select algorithm
   file_info = dir(file_path);
   file_size = file_info.bytes;
% This if-block selects an algorithm based on
the file size
   if file_size < 10000
       algorithm = 'Linear_Regression';
       size_category = 'small';
   elseif file_size < 50000
       algorithm = 'Support_Vector_Regression';
       size_category = 'medium';
   else
       algorithm = 'Random_Forest';
       size_category = 'large';
   end
```

Fig. 10. Model selector using MATLAB.

The second function determines the size of the different files, which are translated into bytes. Based on the file size, it selects an appropriate algorithm and categorizes the file size:

- Small (<10,000 bytes): Linear Regression
- Medium (10,000-49,999 bytes): Support Vector Regression
- Large (≥50,000 bytes): Random Forest Regression

Four MATLAB processors that can run all the files of different workloads while redirecting them to the correct algorithm are created. The selection algorithm function chooses the best algorithm to use based on the size of the files the user uploads. A switch statement is used to call the appropriate function (Linear Regression, Support Vector Regression, or Random Forest Regression) based on the selected algorithm as shown in Fig. 11.

```
% Run the selected algorithm
   switch algorithm
       case 'Linear_Regression'
           Linear_Regression(file_path,
target_column, x_feature);
       case 'Support_Vector_Regression'

Support_Vector_Regression(file_path,
target_column, x_feature);
       case 'Random_Forest'
           Random_Forest(file_path,
target_column, x_feature);
       otherwise
           error('Unknown    algorithm:    %s',
algorithm);
   end
end
```

Fig. 11. Algorithm selection using MATLAB.

III. Comparison Results of Centralized versus Parallel Computing

In this section, we present the runtime comparison results for centralized versus parallel computing.

A. Centralized Execution

The bar graph shown in Fig. 12 illustrates the centralized computing runtime based on the impact factors. For example, file size, CPU total, and memory usage. The *x*-axis shows the impact factors and the *y*-axis is the runtime in seconds.

Each part of the height of the bars shows different components, for example, the first bar shows the elapsed time of three files, the second part shows the CPU time usage of three files, and the last part shows the Memory usage of three files.

For example, The runtime of a large file is 1.75s. In this case, the CPU total is 1.49s, and the memory usage is 104.07MB. It shows that memory usage has the highest runtime, with that being said, memory usage has the most impact.

B. Parallel Execution

The graph shown in Fig. 13 compares the runtime of parallel computing to the same impact factors from different file sizes, three different sizes of files were used, but the same factors which are: file size, CPU total, and memory usage, across large, medium, and small three file sizes. The *x*-axis indicates the impact factors, and the *y*-axis measures the runtime in seconds. Each bar shows the time needed for parallel computing to handle tasks influenced by each impact factor.

For large file sizes, the runtime is 18.23 s, the CPU total is 0.56 s, and the memory usage is 272.52 s. As it is seen memory usage again is the most impactful factor. As for small size, the runtime is 17.93 s, the CPU total is 0.63 s, and the memory usage is 272.52 MB. Consistently showing that memory usage is the dominant factor influencing the runtime performance. The small test cannot show the performance gap. As a future work, a large-scale evaluation, for example, with 1000 files as input, deployed to hundreds of servers, parallel should be much better than centralized in overall performance.

IV. Docker Integration within MATLAB

Dockers have been integrated into the proposed system for the enhancement of the application in runtime performance and latency issues concerning geographical aspects (Fig. 14). Docker is an open-source platform purposed to help developers in building, deploying, and managing their applications.

A customized MATLAB container that includes all of the necessary dependencies and configurations needed to run the application has been created. Dockers not only offer more flexibility but also make it easier to deploy applications across different locations. For the small-size file, the runtime is 17.3s, the CPU total is 0.21s, and the memory usage is 267.91MB, again memory usage has the highest runtime.

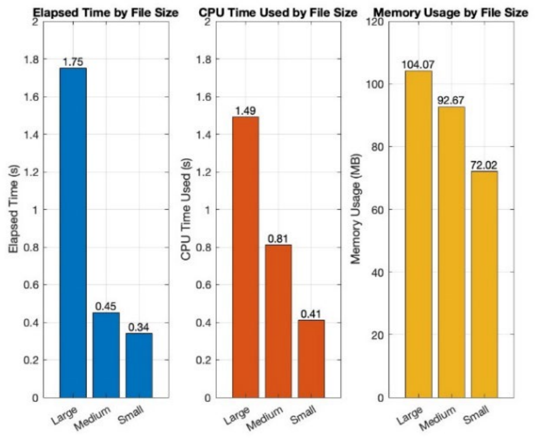

Fig. 12. Impact factors on centralized computing runtime.

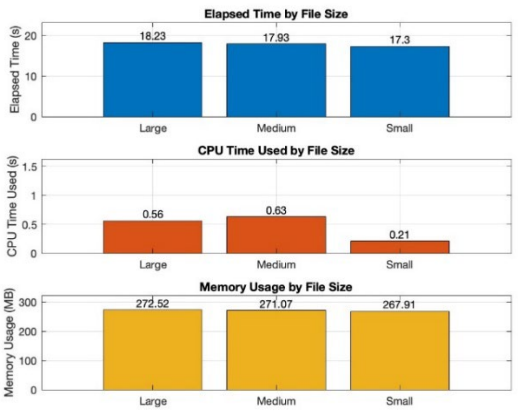

Fig. 13. Impact factor on parallel computing runtime.

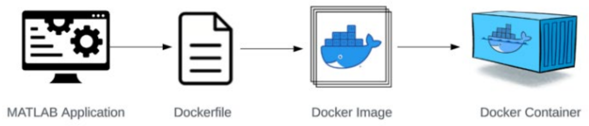

Fig. 14. Overview of the integration of dockers within MATLAB.

From that, we conclude that memory usage is the most significant factor influencing runtime. This suggests that optimizing memory usage could result in substantial improvements in computing performance, regardless of the system architecture or file size.

Based on the analysis of both centralized and parallel computing runtimes above, it indicates that memory usage is the most significant factor impacting runtime across all file size scenarios. While the exact runtime is different between centralized and parallel computing, the graph remains consistent: higher memory usage results in longer runtimes.

V. Comparison of Centralized versus Docker Deployment Performance

Fig. 15 shows the performance of deployments in centralized and Docker environments over three dataset sizes. Docker consistently shows lower runtimes for small, medium, and large datasets (0.09 s, 0.111 s, and 0.169 s, respectively). In comparison to the centralized deployment (0.144 s, 0.187 s, and 0.199 s). This demonstrates that Docker's resource management and deployment are more efficient, resulting in faster processing times regardless of the size of the dataset.

The high performance of Docker has significant implications when considering geographical deployment issues. Docker can span multiple distributed servers, thereby reducing latency and shortening response time.

Theoretically, Docker's scalability can achieve more flexible deployment. Common challenges in centralized systems, such as high latency, scalability, and data regularity, can be effectively solved by using Docker for distributed and geographically dispersed environments.

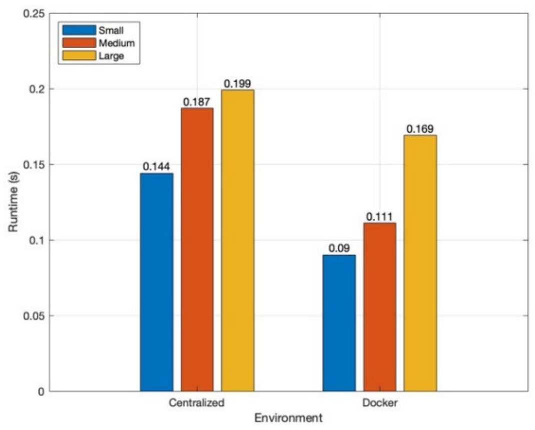

Fig. 15. Docker versus centralized deployment performance.

VI. Conclusions

In this paper, we propose a MATLAB-based dynamic selector model for data workload prediction. The model leverages machine learning models such as Linear Regression, Support Vector Regression, and Random Forest Regression to ensure both efficiency and accuracy in workload predictions. The incorporation of parallel execution in the Docker container further enhances the system's lightweight performance, resource utilization, and efficient deployment on a large scale. Experimental results demonstrate significant improvements in prediction accuracy and runtime efficiency in various distributed environments.

References

[1] Zhu, Mengran, et al. "Enhancing Credit Card Fraud Detection: A Neural Network and SMOTE Integrated Approach." Journal of Theory and Practice of Engineering Science 4.02: 2330, 2024.

[2] Kolasa, Katarzyna, Bisrat Admassu, Malwina Hołownia-Voloskova, Katarzyna J. Kędzior, Jean-Etienne Poirrier, and Stefano Perni. "Systematic reviews of machine learning in healthcare: a literature review." Expert Review of Pharmacoeconomics & Outcomes Research 24, no. 1: 63-115, 2024.

[3] Herhausen, Dennis, Stefan F. Bernritter, Eric WT Ngai, Ajay Kumar, and Dursun Delen. "Machine learning in marketing: Recent progress and future research directions." Journal of Business Research 170: 114254, 2024.

[4] Nandy, Tarak, and Angella Nyundo. "Prediction of Possible Outcomes using Big Data Analysis and Machine Learning." In 2023 International Conference on Applied Intelligence and Sustainable Computing (ICAISC), pp. 1-6. IEEE, 2023.

[5] Shetu, Raju Ahmed, Tarik Toha, Mohammad Mosiur Rahman Lunar, Novia Nurain, and ABM Alim Al Islam. "Workload-based prediction of CPU temperature and usage for small-scale distributed systems." In 2015 4th International Conference on Computer Science and Network Technology (ICCSNT), vol. 1, pp. 1090-1093. IEEE, 2015.

[6] Dong, Chris, Lingzhi Du, Feiran Ji, Zizhen Song, Yuedi Zheng, Alexander Howard, Paul Intrevado, Diane Myung-kyung Woodbridge, and Alexander J. Howard. "Forecasting smart meter energy usage using distributed systems and machine learning." In 2018 IEEE 20th International Conference on High-Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 1293-1298. IEEE, 2018.

[7] Su, Xiaogang, Xin Yan, and Chih‐Ling Tsai. "Linear regression." Wiley Interdisciplinary Reviews: Computational Statistics 4, no. 3: 275-294, 2012.

[8] Awad, Mariette, Rahul Khanna, Mariette Awad, and Rahul Khanna. "Support vector regression." Efficient learning machines: Theories, concepts, and applications for engineers and system designers: 67-80, 2015.

[9] Rodriguez-Galiano, Victor, Manuel Sanchez-Castillo, M. Chica-Olmo, and M. J. O. G. R. Chica-Rivas. "Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees, and support vector machines." Ore Geology Reviews 71: 804-818, 2015.

[10] M. Sai, "Temperature Prediction using Machine Learning," GitHub, 2019. [Online]. Available: https://github.com/Mohan-Sai/Temperature-prediction-using-Machine-Learning. [Accessed: Jun. 19, 2024].