Prediction of the Running Time for Multiplying Two 2048x2048 Matrices using a GPU OpenCL SGEMM Kernel

Mariam Mourad
Computer Engineering Department
Princess Sumaya University for Technology
Amman, Jordan
mar20210798@std.psut.edu.jo

Qabas Ahmad
Computer Engineering Department
Princess Sumaya University for Technology
Amman, Jordan
qab20210786@std.psut.edu.jo

Mahmoud Abu-Qtiesh
Computer Engineering Department
Princess Sumaya University for Technology
Amman, Jordan
mah20210383@psut.edu.jo

***Abstract*—This paper presents the implementation of various models that predict the computation time of a system consisting of a CPU (Central Processing Unit) as well as a GPU (Graphical Processing Unit) as it performs the multiplication of two 2048 x 2048 matrices. The prediction is based on GPU specifications, which consist of 14 different configurations. The models were able to properly learn the data and generalize exceptionally on new unseen data with a prediction accuracy of 99.94% exceeding the results of previous literature.**

***Keywords*—Machine Learning; Matrix Multiplication; Prediction Tasks; GPU; Kernel Performance; KNN Regression, Random Forest Regression.**

I. INTRODUCTION

In recent years, the demand for high-performance computing has surged across numerous fields, from scientific research to machine learning. As computational tasks become exceedingly complex, traditional CPU-based processors often fall short in terms of efficiency and speed. This shortcoming resulted in the emergence of Graphical Processing Units (GPUs) as an alternative such that large datasets are processed in parallel.

Matrix multiplication, a fundamental operation in linear algebra, lies at the core of numerous applications in mathematics, physics, chemistry, engineering and computer science. It plays a critical role in tasks such as PageRank [1] and Web Search algorithms, where eigenvector computations are central. However, the traditional matrix multiplication algorithms with their $O(n^3)$ time complexity become infeasible for large matrices. For instance, multiplying two $10^5 \times 10^5$ matrices, a typical size of web graphs, using a CPU, where each operation takes one microsecond, would require approximately 63.4 years to complete. This underscores the need for faster, more efficient computational approaches.

GPUs, particularly those designed for General Purpose Graphics Processing Units (GPGPUs) are highly effective for matrix multiplication as they parallelize tasks by dividing large matrices into smaller chunks for simultaneous processing. This capability is essential in machine learning, especially when handling large datasets. The SGEMM (Single-precision General Matrix Multiply) kernel, in particular, is a cornerstone of GPU computing as it provides efficient single-precision matrix multiplication. For example, a typical GPU like the NVIDIA GTX 1660 [2], with a processing capability of 5 TFLOPS, can perform the same $10^5 \times 10^5$ matrix multiplication in approximately 6.67 minutes, a stark contrast to the years required by a CPU.

In this paper, various machine-learning algorithms were implemented to predict the computational performance of an SGEMM GPU kernel for the multiplication of two arrays each of size 2048 x 2048 leveraging a dataset of 241,600 instances characterized by 14 independent variables spanning various GPU configurations. It not only contributes to the precise calculation of GPU performance but also aids engineers in selecting appropriate hardware configurations tailored to specific computational needs, bridging the gap between theoretical analysis and practical applications.

To provide a comprehensive framework for this study, the subsequent sections of the paper are structured as follows. The related works section explores prior research that has attempted to approach the same problem, highlighting the results and challenges. The dataset section delves into the dataset used for this study, explaining the features and the underlying GPU concepts necessary for this study. Additionally, that section outlines the feature engineering techniques employed to enhance the quality of model training in later sections. Following it is the section covering the models implemented and the results obtained, utilized for this study, including linear regression, polynomial regression, decision trees, random forests and k-nearest neighbors. And at the end, the conclusion section is added to outline the results and potential future work.

II. RELATED WORKS

Agrawal et al. proposed a machine learning-based approach to predict the computation time of an SGEMM GPU kernel for matrix multiplication using a dataset with 14 independent features, including local workgroup size, memory shape, and vector widths. They utilized PCA and backward elimination for feature reduction and employed KNN and Random Forest regressor for performance prediction. Their results showed that removing two features (VWM and MDIMA) using backward elimination enhanced prediction accuracy, with Random Forest achieving the highest test accuracy of 98.46%. The study demonstrates the effectiveness of combining feature reduction techniques with machine learning models for predicting GPU kernel performance [3].

The work presented in [4] discusses how certain programming structures can limit parallel execution which aligns closely to the optimization of SGEMM GPU kernel performance. This paper addresses these limitations by using machine learning techniques to predict running time of matrix multiplication based on varying GPU configurations, the limitations posed by sequential programming structures are addressed. Key parameters such as MWG and NWG play

a pivotal role in balancing parallelism and efficiency. It aligns with Bernstein's emphasis on identifying independent tasks that enhance computational efficiency.

Konstantinidis and Cotronis [5] introduce a quadrant-split visual model to identify the limitations in memory and computational bounds for GPU kernels. While effective, this paper's machine learning framework automates performance prediction, achieving higher scalability for various applications.

The authors of the paper "Predicting GPU Performance from CPU Runs Using Machine Learning" show a way of using machine learning to predict GPU performance through running the applications only on CPUs. This technique is used to estimate GPU metrics such as execution time and energy consumption by considering features like memory and instruction pattern during CPU runs. Such techniques allow significant optimization of the applications, as they reduce the amount of GPU runs needed for the application development. However, the approach, may not be adequate for applications with heavy parallelism other than what CPU profiling allows, as this one appears to be rather coarse. [6]

III. Dataset

The dataset for evaluation was created by Paredes et al. in October through real-world experiments. It measures the running time of matrix multiplication for 2048 x 2048 single-precision floating-point matrices A, B, and C using a CUDA SGEMM kernel. The dataset comprises 241,600 feasible parameter combinations, representing around 20% of the total possibilities due to kernel constraints. It includes 14 parameters that capture key aspects of GPU kernel execution.

A. Dataset attribute Information

MWG and NWG define the workgroup sizes, which determine how the workload is divided among threads in the M and N dimensions, ensuring efficient distribution of computations. KWG specifies the tiling size for the inner dimension, which optimizes data reuse and reduces memory overhead.

MDIMC and NDIMC define the local workgroup sizes for the M and N dimensions, controlling how threads collaborate to compute results in parallel. MDIMA and NDIMB determine the local memory allocation shapes for matrices A and B, helping to reduce global memory access overhead. KWI is a kernel shaping parameter that adjusts how the kernel execution is structured in combination with KWG, balancing memory and computational efficiency. VWM and VMN set the vector widths for processing elements of matrices A/C and B, enhancing memory access efficiency through coalescing.

STRM and STRN control strided access patterns in the M and N dimensions, reducing memory conflicts, while SA and SB decide whether portions of the workload are cached in high-speed shared memory to improve performance. These parameters collectively capture essential characteristics of GPU parallel processing and enable fine-tuning for optimal performance.

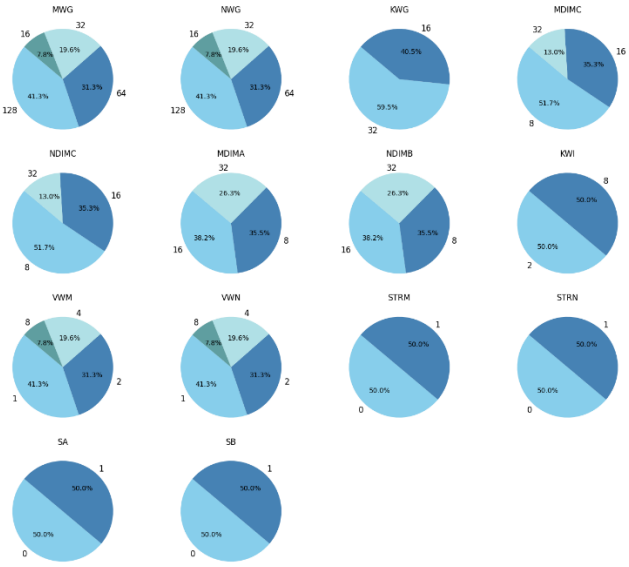

Figure 1 Input Features Distribution

Table (1) below summarizes the input features and lists their possible values.

Table 1 Input feature summary

Parameter	Type	Values
MWG	Ordinal	{16, 32, 64, 128}
NWG	Ordinal	{16, 32, 64, 128}
KWG	Ordinal	{16, 32}
MDIMC	Ordinal	{8, 16, 32}
NDIMC	Ordinal	{8, 16, 32}
MDIMA	Ordinal	{8, 16, 32}
NDIMB	Ordinal	{8, 16, 32}
KWI	Ordinal	{2, 8}
VWM	Ordinal	{1, 2, 4, 8}
VWN	Ordinal	{1, 2, 4, 8}
STRM	Binary	{0, 1}
STRN	Binary	{0, 1}
SA	Binary	{0, 1}
SB	Binary	{0, 1}

After configuring the GPU based on the feasible parameter combinations, four independent runs of matrix multiplications took place to capture how long it took to multiply matrix A and B, these results were measured in milliseconds.

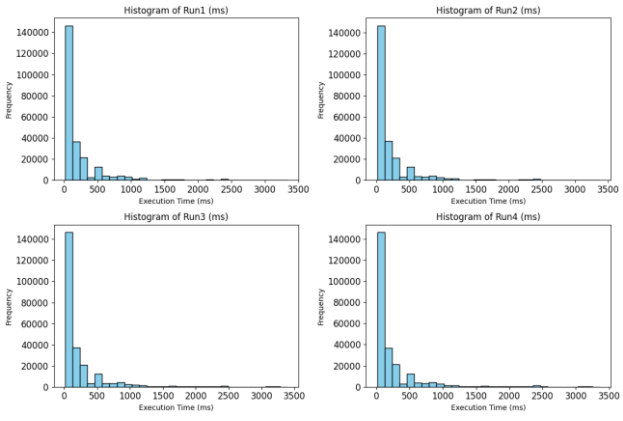

Figure 2 Independent Run Distribution

B. Data Preprocessing

As demonstrated in the work by Falch et al [7], taking the average of multiple independent runs simplifies the task by avoiding the complexity of multi-label classification, particularly when the runs exhibit nearly identical distributions. Similarly, in this study, the average of four independent runs was used as a new label since their distributions were nearly identical. That being said, the resulting distribution was heavily left-skewed as most results finished relatively fast, which could hinder model training. To address this, a logarithmic transformation was applied to normalize the data and improve its suitability for machine learning models.

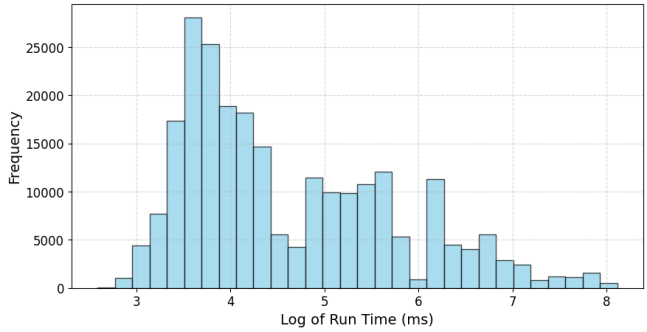

Figure 3 Logarithmic Runtime Distribution

The newly created label, "log_runs," demonstrates a significantly improved distribution compared to "average_run." This is evident from its lower variance (1.27), which indicates that the values are tightly clustered around the central tendency. Additionally, the difference between the mean (4.62) and the median (4.26) is minimal, further supporting its consistent and well-behaved distribution.

In contrast, the "average_run" label exhibits a much higher variance of 135,976.68, suggesting a greater spread of values. Moreover, the difference between its mean (217.57) and median (38.76) is substantial, highlighting a skewed distribution with values spread far from the central tendency. These metrics underscore that the "log_runs" label has values more consistently distributed around the median, whereas "average_run" shows significant variability and a less stable distribution.

C. Data Exploration

To examine the relationships between numerical variables, a correlation matrix is used to visualize the strength and direction of linear correlations between the features. Values closer to (1) indicate a positive correlation, values closer to (-1) indicate a negative correlation. The strength of this correlation is dictated by the absolute value of the correlation, the higher it is the stronger the correlation between the two features.

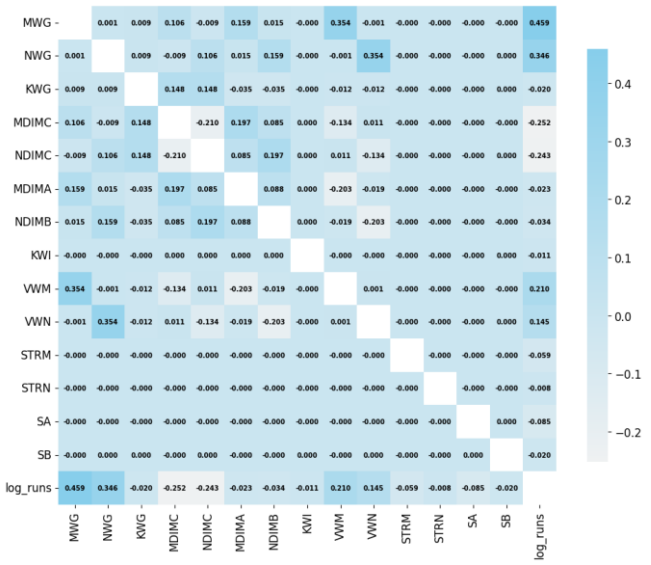

Figure 4 Heatmap of Correlation Matrix

The Heatmap visualizes the linear correlations between numerical features in the dataset. Darker shades represent stronger correlations, with positive correlations in blue and negative correlations in lighter shades. The diagonal is cleared for clarity, emphasizing inter-variable relationships.

IV. Model Training and Prediction

This section explores the training, evaluation, and results of five classical machine-learning models employed to predict the computational performance of the SGEMM GPU kernel. The models include Linear Regression, Polynomial Regression, Decision Trees, Random Forest, and k-Nearest Neighbors, representing a spectrum of approaches from simple linear relationships to advanced non-linear and ensemble-based methods. By comparing their performance, this study aims to identify the model with the highest R^2 score. As a baseline, the results are compared with the work of Agrawal et al., who achieved an impressive R^2 score of 99.86%. This benchmark sets a high standard for the models evaluated in this study, pushing for accuracy and reliability in the predictions.

Before delving any deeper, it is essential to explain the metrics used to evaluate the models: Mean Absolute Error (MAE), Mean Squared Error (MSE) and Coefficient of Determination (R^2). These metrics provide complementary insights into the models' performance and predictive accuracy

MAE [8] measures the sum of the average of the absolute difference between the predicted and the actual values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad (1)$$

MSE measures the average squared difference between predicted and actual values:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (2)$$

Lower MSE indicated more accurate predictions, but it is measured in squared units, which can make it more difficult to interpret. For this reason, we will primarily focus on R^2.

R^2 measures [9] how well the model explains the variance in the target variables compared to the mean:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \quad (3)$$

An R^2 of value one indicated a perfect fit, whilst value zero means the model is no better than the mean, negative values are rate but occur when the model performs worse than the baseline. Its use metric to understand the model's overall effectiveness.

A. Linear Regression (LR)

For any regression problem, it is wise to start with the most basic and popular model; Linear Regression [10]. It is a fundamental statistical method used to model relationship between one or more independent predictors and a dependent outcome. Based on the model's results, one can tell if the system is linear or not. The model achieved an MSE of 0.5576 and an R^2 score of 0.5578, explaining about 55.8% of the variance, indicating the system's behavior may not be linear.

Table 2 Performance of Linear Regression

MSE Value	R^2 Score
0.557585389196344	0.5577962484740586

B. Polynomial Regression (PR)

The next step is to increase the degree of the polynomial and test again. Table (3) shows the results of both training and testing data. Better than LR however not the best.

Table 3 PR Results for a Polynomial of Degree 3

	MSE	RMSE	R^2
Training Data	0.149411398	0.386537706	0.8815064347
Testing Data	0.151238631	0.388894113	0.8815428506

Using cross-validation did not improve the results of the test data, indicating an issue with the model used.

Table 4 5-fold CV on PR of Degree 3

Mean MSE Value	Mean R^2 Score
0.15390382666975774	0.877895752802546

C. Decision Tree (DT)

Decision Trees are intuitive machine learning models that split data into branches based on decision rules [11]. Each split is made to minimize errors and improve prediction accuracy. The maximum depth of a tree defines the number of splits or levels, controlling the model's complexity. A deeper tree captures more intricate patterns but risks overfitting [12], while a shallow tree may underfit the data. In this study, a tree with a maximum depth of 10 was initially tested, yielding promising results, as shown in Table (5).

Table 5 DR Results with Max Depth of 10

	MSE Value	Mean R^2 Score
Training Data	0.0306	0.9759
Testing Data	0.0314	0.9772

Tree of depth 10 appears to perform reasonably well, see Figure (5). The red dashed line indicates true values, and the closer the points are to this line, the greater the accuracy of the model. Most of the blue points are grouped about the red line, which suggests an acceptable overall performance of the model with some small degree of inaccuracy regarding the scattered points.

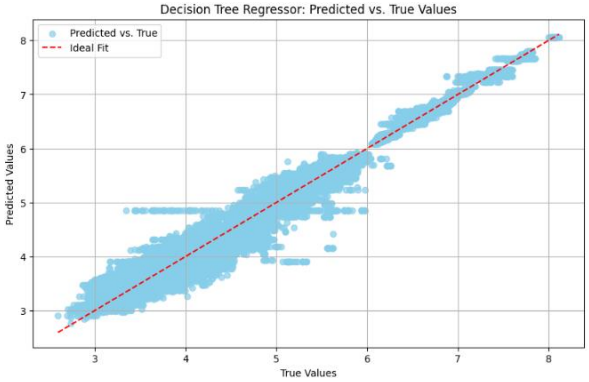

Figure 5 scatter plot for DT (max depth = 10) predicted vs true values

As a way to tune the "max depth" hyperparameter, Grid Search (GR) was performed to figure out the best tree depth. The best results so far, it is also an enhancement over baseline performance. Fitting 5 folds for every 20 candidates, a total of 100 fits having a maximum depth of 20 results in the following values in Table (6).

Table 6 GS result for best depth

MSE Value	R^2 Score
0.0011	0.9991

Absorbing the plot in Figure (6), blue points are closer to the red dotted line (actual values), and because the results are on test data, the model is not learning the data's pattern and overfitting.

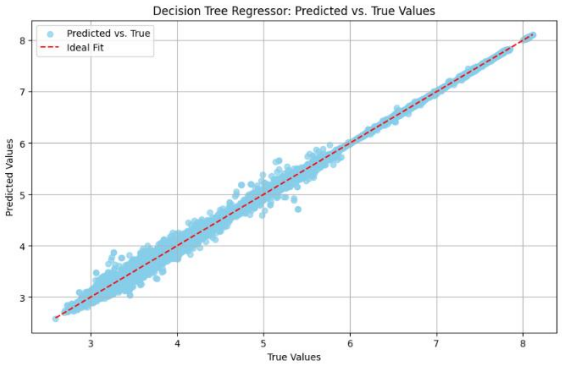

Figure 6 DT scatter plot after applying hyperparameter tuning

D. *Random Forest (RF)*

Random Forest model fits multiple decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting and bad generalization. The random forest regressor was trained on the training data and later evaluated using various metrics on the testing data to give the following results shown in Table (7):

Table 7 Performance of RF

MAE Value	RMSE Value	R2 Value
0.01493839	0.02777873	0.999388

Then 5-fold cross-validation was performed to ensure that there was no overfitting. It gave consistent values across each fold with an average score of 0.9991764193.

To find the best parameters for the regressor, a randomized search from Scikit-learn was used and it showed that the best parameters are 300 estimators, a minimum sample split of two, and a minimum sample leaf of one and a maximum depth of fifty. Using these parameters, Table (8) shows the results for various performance metrics:

Table 8 Performance of RF with Best Parameters

MAE Value	RMSE Value	R2 Value
0.0148376	0.0276029	0.999388

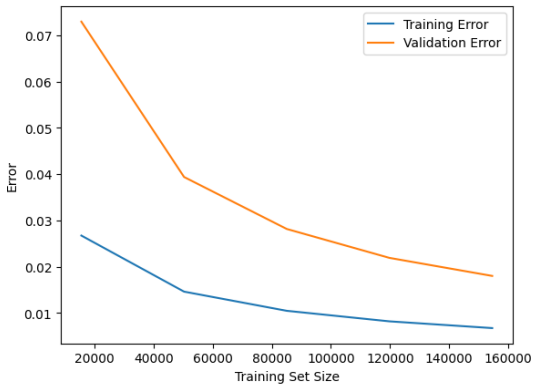

Figure 7 Effect of Training Set Size of Error

Furthermore, residual error was plotted to ensure the difference between the observed value and the predicted value is within an acceptable range, as shown in the figure.

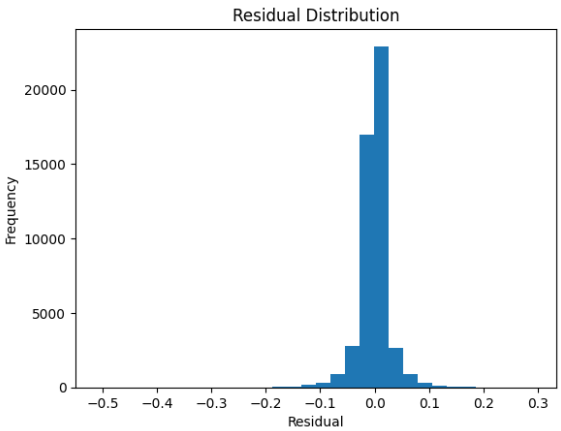

Figure 8 Residual Error Distribution

Using Random Forest estimator, the feature importance was plotted to highlight the features that contribute most to predicting the target variable, as well as those with the least effect on the prediction. See Figure (9). Features with low impact could be dropped to simplify the model and improve computational efficiency as well as reduce overfitting if occurred.

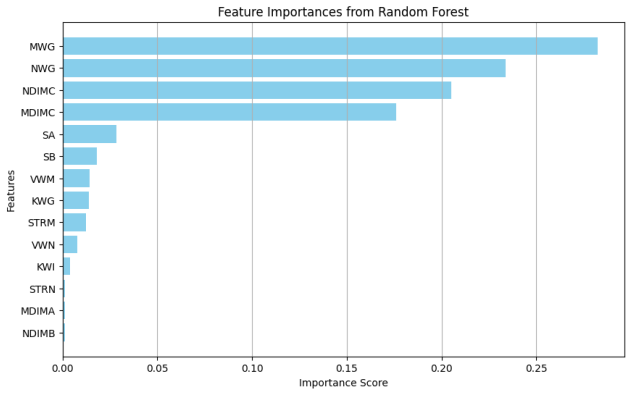

Figure 9 Feature importance horizontal bar plot

Features like MWG, NWG and KWG dominate due to their role in GPU kernel execution as they showcase the GPU's ability to parallelize and execute matrix multiplication. Larger workgroups (MWG/NWG) allow more parallelism but may lead to inefficiencies if limited by the hardware. Larger tiles (KWG) reduce memory access latency but can result in higher contention for shared memory.

Features like NDIMB, MDIMA and STRN have very low scores, indicating minimal impact on the predictions. However, dropping features does not always improve the results; Table (9) shows the results of the same PR (degree 3) model in addition to dropping the mentioned features. In comparison with the original model, the performance has worsened.

Table 9 PR of degree 3 after feature dropping

MSE Value	RMSE Value	R2 Score
0.15459170413	0.3931815155	0.8773980939

E. *K-Nearest Neighbours Regressor (KNN) :*

KNN regressor predicts a value by averaging the target values of its k nearest neighbours based on a chosen distance metric, like Euclidean distance [13].

Before training the KNN model the dataset went through 2 stages, firstly a new feature was added which had a higher correlation to the new label, and then feature reduction using the feature's MI scores. Mutual information [14], or MI scores measure the dependency between variables, quantifying how much knowing one variable reduces uncertainty about another. Higher scores indicate stronger relationships.

Figure (10) shows the optimal value for the threshold of feature reduction. Removing too many features resulted in worse performance compared to leaving all features.

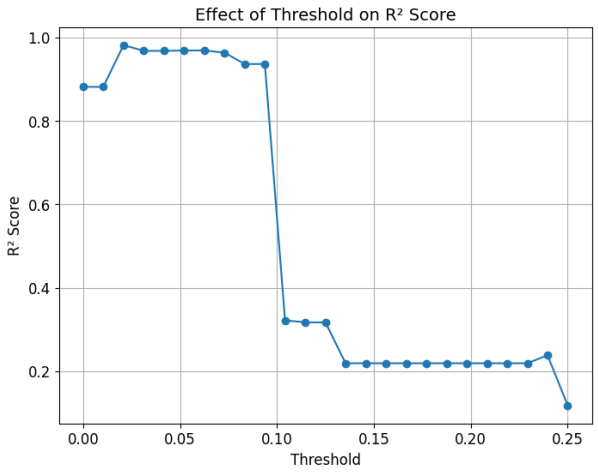

Figure 10 Effect of feature reduction on R^2 score for KNN regression

The feature reduction process selects the most relevant features. Features with MI scores above a chosen threshold are kept, while others are removed. This simplifies the dataset and improves model performance by focusing on the most predictive features.

The new feature mentioned earlier is called execution complexity, which represents the computational workload of the kernel, combining workgroup sizes (MWG, NWG), and tiling size (KWG) to quantify the parallelism and resource demands for matrix multiplication. This new-engineered feature had a correlation of 0.513418 with the logarithm of the average of the 4 independent runs, which is higher than MWG which had a correlation of 0.45049 as can be seen in Figure (11) below. It must also be noted that the only reason this feature was introduced now and not in previous models was that upon testing the new feature with models other than the KNN regressor the performance worsened, for that reason it will only be used for the KNN regressor.

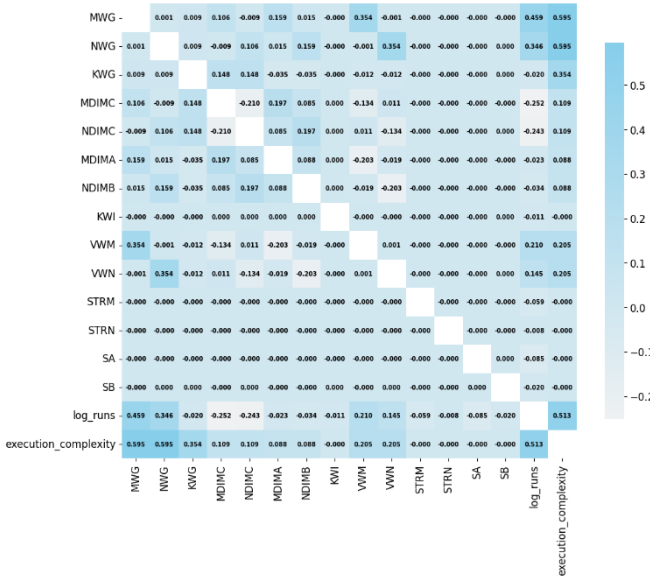

Figure 11 Correlation Matrix after adding execution complexity

The KNN model underwent two stages of cross-validation, focusing on three key parameters. The first is n_neighbors, which specifies the number of neighbours used to determine the label. The second is weights, which define how much influence each neighbor has on the prediction. Finally, the third is the metric, which determines the distance calculation method used to identify the nearest neighbours.

In the initial stage of cross-validation, the goal was to identify the best combination of weights and metric for a fixed value of n_neighbors. The optimal configuration is:

- Weights=”distance”: Neighbors closer to the point have greater influence.
- Metric=”manhattan”: The Manhattan distance provided the most effective measure for determining proximity [15].

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i| \quad (4)$$

A second stage of cross-validation involved trying out different values of k to see which had the best R^2 score.

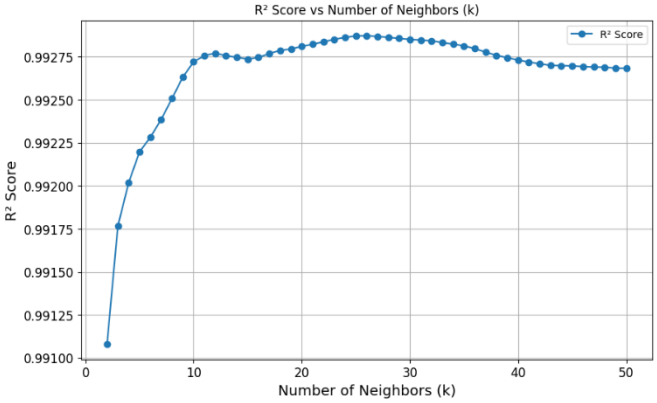

Figure 12 Effect of increasing K, on R2

As can be seen from Figure (12), the best n_neighbors have 26 neighbors, which resulted in an R^2 score of 0.99287.

V. Ensemble Model

An ensemble model of the previous Random Forest and KNN implementations was used to seek better prediction accuracy. The hyperparameters that gave the best performance previously were used in the ensemble. However, due to the already high performance of Random Forest, the performance of the ensemble did not exceed that of RF and RF still dominated, giving the results in Table (10).

Table 10 Ensemble Model Performance

	RF Model	KNN Model	Ensemble
MAE Value	0.01483677	0.1922041	0.0436525
MSE Value	0.0007621	0.06630585	0.0032732
R2 Score	0.9993956	0.94999028	0.99740413

VI. Results Discussion

The results obtained over the three main models used in this paper, Random Forest, KNN and polynomial regression, all show promising results and the overall accuracy exceeded that of the previous literature using the same dataset, the findings are summarized in the following figure (13).

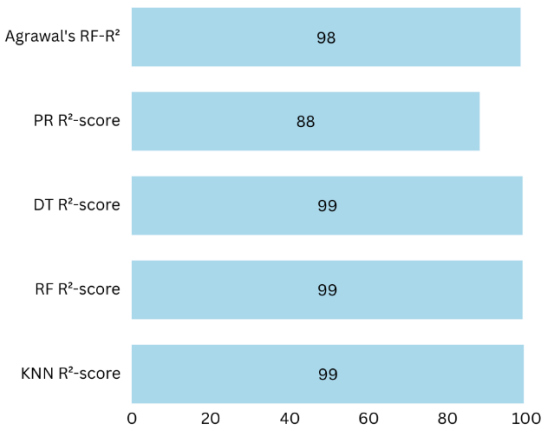

Figure 13 Comparative Evaluation

Table 11 Model Evaluation Comparison

	KNN R2	RF R2
Previous Findings	0.9838	0.9846
Current Findings	0.9927	0.9994

The findings encourage further exploration into additional features and the usage of new machine learning techniques in the prediction. A potential avenue for future research could be deep learning with more detailed CPU configurations.

VII. Conclusion

With the larger scope of problems beginning to be computerized and requiring a lot of computational resources, the accurate prediction of the running time is a necessary part of any project. Using a random forest regressor with parameters of 300 estimators, a minimum sample split of two, a minimum sample leaf of one and a maximum depth of fifty results in excellent generalization on new unseen data. This method is able to predict the running time with a test accuracy of 99.94%.

Scientists and engineers will be able to analyze the running time of computationally-demanding applications efficiently and accurately. In future studies, more detailed GPU configurations can be added to cover corner cases.

References

[1] G. M. G. A. &. R. F. Del Corso, "Fast PageRank computation via a sparse linear system," *Internet Mathematics*, vol. 2, no. 3, 2005.

[2] V. Allada, T. Benjegerdes and B. Bode, "Performance analysis of memory transfers and GEMM subroutines on NVIDIA Tesla GPU cluster," in *IEEE International Conference on Cluster Computing*, 2009.

[3] S. Agrawal, A. Bansal and S. Rathor, "Prediction of SGEMM GPU Kernel Performance using Supervised and Unsupervised Machine Learning Techniques," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Bengaluru, 2018.

[4] A. J. Bernstein, "A. J. Bernstein, "Analysis of Programs for Parallel Processing," in IEEE Transactions on Electronic Computers, vol. EC-15, no. 5, pp. 757-763, Oct. 1966, doi: 10.1109/PGEC.1966.264565.," *IEEE Transactions on Electronic Computers.*

[5] E. K. a. Y. Cotronis, "E. Konstantinidis and Y. Cotronis, "A Practical Performance Model for Compute and Memory Bound GPU Kernels," 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Turku, Finland, 2015, pp. 651-658, doi: 10.11," *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*.

[6] I. Baldini, S. J. Fink and E. Altman, "Predicting GPU Performance from CPU Runs Using Machine Learning," in *IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, Paris, France, 2014.

[7] T. L. Falch and A. C. Elster, "Machine learning-based auto-tuning for enhanced performance portability of OpenCL applications," *Concurrency and Computation: Practice and Experience*, vol. 29, 2017.

[8] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Research*, vol. 30, no. 1, pp. 79-82, 2005.

[9] R. G. D. Steel and J. H. Torrie, Principles and Procedures of Statistics with Special Reference to the Biological Sciences, McGraw Hill, 1960.

[10] Z. Li and A. Kannan, "Algorithm Switching for Multiobjective Predictions in Renewable Energy Markets.," in *Learning and Intelligent Optimization*, Cham, Springer Nature Switzerland, 2025, pp. 233-248.

[11] J. R. Quinlan, "Simplifying decision trees," *International Journal of Man-Machine Studies*, vol. 27, no. 3, pp. 221-234, 1987.

[12] M. Bramer, "Avoiding Overfitting of Decision Trees," in *Principles of Data Mining*, Springer, 2016.

[13] P. H. T. Cover, " Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, 1967.

[14] P. Baudot, M. Tapia, D. Bennequin and J. M. Goaillard, "Topological Information Data Analysis," *Entropy*, vol. 21, no. 9, 2019.

[15] P. E. Black, "Manhattan distance," Dictionary of Algorithms and Data Structures. [Online].