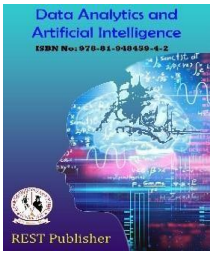
Data Analytics and Artificial Intelligence

Vol: 6(1), 2026

REST Publisher; ISBN: 978-81-948459-4-2

Website: http://restpublisher.com/book-series/daai/

DOI: https://doi.org/10.46632/daai/6/1/5



Predicting Processor Efficiency Through Regression Models: A Data-Driven Approach in Computer Architecture

Ananth Majumdar

Vice President, Wellington Management

Corresponding Author Email: thisisananth@gmail.com

Abstract: *This study investigates the application of machine learning techniques for predicting processor performance based on essential hardware configuration parameters. The study considers three major input features: clock speed, cache size, and core count, with Performance_Score serving as the target variable. Linear Regression (LR) and Random Forest Regression (RFR) are implemented to analyze the relationship between hardware specifications and processor performance. A comprehensive dataset is collected, preprocessed, and divided into training and testing sets to evaluate model performance and generalization capability. The predictive effectiveness of both models is assessed using standard evaluation metrics, including R^2, Explained Variance Score (EVS), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Maximum Error. The results indicate that Random Forest Regression achieves a training R^2 score above 0.98, demonstrating its ability to capture complex patterns. However, higher testing errors reveal overfitting. In contrast, Linear Regression provides more consistent predictions, lower test errors, and better generalization on unseen data. The study concludes that Linear Regression is more suitable for the analyzed dataset. This research highlights the potential of machine learning for processor performance estimation and supports efficient hardware design, architectural evaluation, and future optimization strategies.*

Keywords: *Machine Learning, Linear Regression, Random Forest Regression, Processor Performance Prediction, Computer Architecture, Clock Speed, Cache Size, Core Count, Regression Models, Performance Score.*

1. INTRODUCTION

In computer architecture, designers must thoroughly understand technological limits, system behavior, and user needs while exploring multiple processor configuration choices. Using their knowledge of program behavior and system-level interactions, architects develop processors optimized for area, performance, and power. Over the years, many architectural paradigms have been created to exploit parallelism and improve computational efficiency. Modern computer architectures generally include multiple interconnected processing units that work simultaneously to solve complex tasks. These architectures are often described using the stream model, which explains how instruction streams and data streams interact. The transition from basic scalar processors to highly parallel systems reflects the continual effort to increase processing speed and make better use of hardware resources. Scalar processors, categorized as Single Instruction Single Data (SISD), represent the simplest and most economical architectural form, executing only one instruction at a time [1]. Simulation is essential in processor and system design because it offers a cost-effective way to examine architectural options and evaluate performance trade-offs. In research, simulation becomes even more valuable, enabling designers to test new ideas and analyze various design factors. However, simulation codes are often not reusable across different tools, restricting researchers to platforms with available models and limiting the exploration of unconventional architectures. This dependence on simulation has also slowed the adoption of other modeling techniques, such as analytical and hybrid approaches, even though they could provide meaningful insights. Despite several attempts, analytical modeling has not gained strong acceptance in the research community [3]. To overcome these limitations, Machine Learning (ML) is increasingly used in computer architecture. ML techniques offer new ways to optimize, predict, and automate various aspects of architectural design. Beyond processor development, ML methods contribute to data center management, compiler optimization, and automatic code generation, improving scalability and efficiency. As architectures grow more complex, relying only on manual design and tuning becomes costly and inefficient. This shift has driven interest in self-optimizing systems that can autonomously adjust to workloads, detect bottlenecks, and recover from failures. Recent research shows that ML-driven modeling and optimization greatly enhance design automation and performance tuning. Machine learning now influences the entire architectural workflow, including data collection, model

construction, algorithm selection, and performance evaluation [4]. Traditional studies often rely on cycle-accurate simulators such as sim-outorder to obtain statistical measurements. Because these simulations are extremely slow, newer methods—like sampling representative execution segments and combining statistical and functional simulations—are being adopted to reduce evaluation time [5]. At the same time, the classification of computer architectures continues to advance. Modern taxonomies expand on Flynn's model by adding hierarchical distinctions, especially within multiprocessor systems. These classifications first separate architectures based on the number of instruction and data streams, and then further categorize them by different interconnection strategies [6]. Computer architecture education has also evolved. While earlier courses focused primarily on hardware mechanisms, modern teaching integrates visualization and simulation tools to support interactive learning. These tools help students understand complex topics such as pipelining, caching, and branch prediction by illustrating how software and hardware interact. Simulation-based learning bridges theoretical concepts with practical system behavior, enabling learners to understand performance trade-offs more intuitively [7]. New frameworks aim to unify design and modeling practices in computer architecture. For example, Python-based domain-specific modeling languages allow simultaneous structural and behavioral representation, supporting refinement from algorithm design to hardware implementation [8]. As technology scales further, researchers pursue innovative architectures that balance power, performance, and energy efficiency. However, validating these designs remains challenging without physical prototypes, so designers historically rely on multiple abstraction levels—from high-level operational models to detailed register-transfer models—to trade off speed and accuracy. To support both research and education, numerous computer architecture simulators have been developed. These tools help illustrate system performance, hardware-software interactions, and architectural behavior. They are particularly helpful for undergraduate students in fields such as Computer Science, Electronics, and Automation, enabling them to link theoretical material with hands-on programming and hardware concepts [9][10][11]. Advanced courses on multiprocessor and parallel architectures are becoming increasingly important as they teach critical topics such as concurrency, synchronization, and distributed computing [12]. Similarly, emerging fields like quantum computing introduce new architectural challenges—including error correction, qubit coherence, and scalability—which push innovation at both the hardware and logic levels [13]. Digital signal processing (DSP) architectures also share principles with general computer architecture but place greater emphasis on application-specific data flow and control optimization [14]. Simulation and animation tools remain valuable in both research and education by offering interactive visualizations that clarify complex architectural concepts [15]. Looking forward, the future of computer architecture is shaped by the integration of classical, quantum, and reconfigurable computing approaches. As interconnection technologies evolve, and non-volatile, fine-grained architectures become more common, system designs will favor modularity and local connectivity [18]. Developments in high-level language computer architectures (HLLCA) are further transforming how programming abstractions map to hardware, improving efficiency and reducing overhead [19]. Similarly, next-generation quantum architectures are adopting design models that separate logic and memory functions—reflecting the traditional von Neumann structure while introducing new ideas such as ideal qubit abstractions and quantum bus systems to improve fault tolerance [20].

2. MATERIAL AND METHODS

Materials: Clock_Speed_GHz: Clock speed, measured in **gigahertz (GHz)**, represents the operating frequency of a processor that is, how many cycles per second it can execute. A higher clock speed generally indicates faster processing capability, as the processor can complete more instructions within a given time frame. In modern computer architectures, clock speed is a critical factor influencing system performance, particularly for workloads that depend on sequential execution. However, increasing clock frequency also raises power consumption and heat generation, making it essential to balance performance with energy efficiency. In this dataset, the clock speed values range between approximately 1.0 and 5.5 GHz, reflecting realistic processor configurations found in contemporary computing systems.

Cache_Size_MB: Cache size, measured in **megabytes (MB)**, denotes the amount of high-speed memory integrated directly within the processor. This memory stores frequently accessed data and instructions, reducing the need to fetch information from slower main memory. Larger cache sizes typically lead to reduced latency and improved processing efficiency, particularly for data-intensive or repetitive tasks. The dataset includes cache sizes ranging from 2 MB to 64 MB, capturing a wide variety of processor architectures — from low-power embedded designs to high-performance computing systems. This parameter plays a vital role in determining the overall responsiveness and throughput of a processor.

Core_Count: Core count refers to the **number of independent processing units** within a single processor chip. Each core can execute its own sequence of instructions, allowing multiple tasks to be processed in parallel. As software applications increasingly rely on multithreading, the number of cores directly impacts the ability to handle concurrent operations efficiently. The dataset includes processors with core counts ranging from 2 to 32, representing a realistic mix of consumer, workstation, and server-grade architectures. A higher core count generally enhances performance for parallel workloads such as simulations, rendering, and machine learning, though it may yield diminishing returns for purely sequential tasks.

Copyright@ REST Publisher

Performance_Score: The **Performance Score** serves as a composite output variable that quantifies the processor's overall computational capability. It is derived from a combination of factors including clock speed, cache size, and core count, capturing their cumulative influence on system performance. In this dataset, the performance score is a synthesized metric reflecting benchmark-like behavior — higher values indicate superior processing power. This parameter provides a convenient means of comparing different architectural configurations and evaluating the trade-offs between speed, efficiency, and scalability. The performance score thus acts as a key indicator for regression modeling and predictive analysis within the domain of computer architecture optimization.

3. Machine Learning Algorithms

Linear Regression: Linear Regression is one of the simplest and most widely used predictive modeling techniques in machine learning. It establishes a linear relationship between an independent variable (input) and a dependent variable (output) by fitting a straight line that best represents the trend in the data. The model estimates coefficients that quantify how changes in input features influence the predicted outcome, making it highly interpretable and easy to implement. Linear Regression performs well when the data shows a clear linear pattern and the underlying assumptions—such as normality, independence, and homoscedasticity—are satisfied. However, its performance declines when relationships are nonlinear or when multicollinearity exists among features. Despite these limitations, Linear Regression remains a foundational method for predictive analytics due to its simplicity, transparency, and effectiveness in many real-world applications.

Random Forest Regression: Random Forest Regression is an advanced ensemble learning method that constructs multiple decision trees during training and aggregates their predictions to produce a more accurate and stable output. Each tree is built using a random subset of data and features, which ensures diversity among the models and reduces the risk of overfitting. By averaging predictions from many trees, Random Forest provides robust performance even when the data contains noise, nonlinear relationships, or complex interactions between variables. It handles both large datasets and high-dimensional feature spaces effectively, making it suitable for a wide range of regression tasks. Although Random Forest models are less interpretable than simpler techniques, they offer strong predictive power, resistance to overfitting, and high reliability, making them a popular choice in modern machine learning applications.

4. RESULT AND DISCUSSION

TABLE 1. Descriptive Statistics

	Clock_Speed_GHz	Cache_Size_MB	Core_Count	Performance_Score
count	150.0000	150.0000	150.0000	150.0000
mean	3.1275	34.0827	18.0267	195.3041
std	1.3343	18.0665	9.4670	154.2278
min	1.0200	2.3000	2.0000	7.6200
0.2500	1.9700	17.3250	11.0000	64.4200
0.5000	3.0150	36.4500	19.0000	169.2600
0.7500	4.3725	48.9750	27.0000	289.6350
max	5.4400	63.4000	32.0000	720.1300

The dataset comprises 150 records representing four key parameters of computer architecture: Clock_Speed_GHz, Cache_Size_MB, Core_Count, and Performance_Score. The average clock speed is 3.13 GHz, ranging from 1.02 GHz to 5.44 GHz, indicating a mix of low- and high-frequency processors. The Cache_Size_MB varies from 2.3 MB to 63.4 MB, with a mean of 34.08 MB, reflecting a diverse range of memory capacities used for faster data access. The Core_Count spans from 2 to 32, averaging 18.03, suggesting inclusion of both entry-level and multi-core processors. The Performance_Score, which quantifies processing capability, ranges widely from 7.62 to 720.13, with a mean of 195.30, showing considerable variation in performance.

Copyright@ REST Publisher

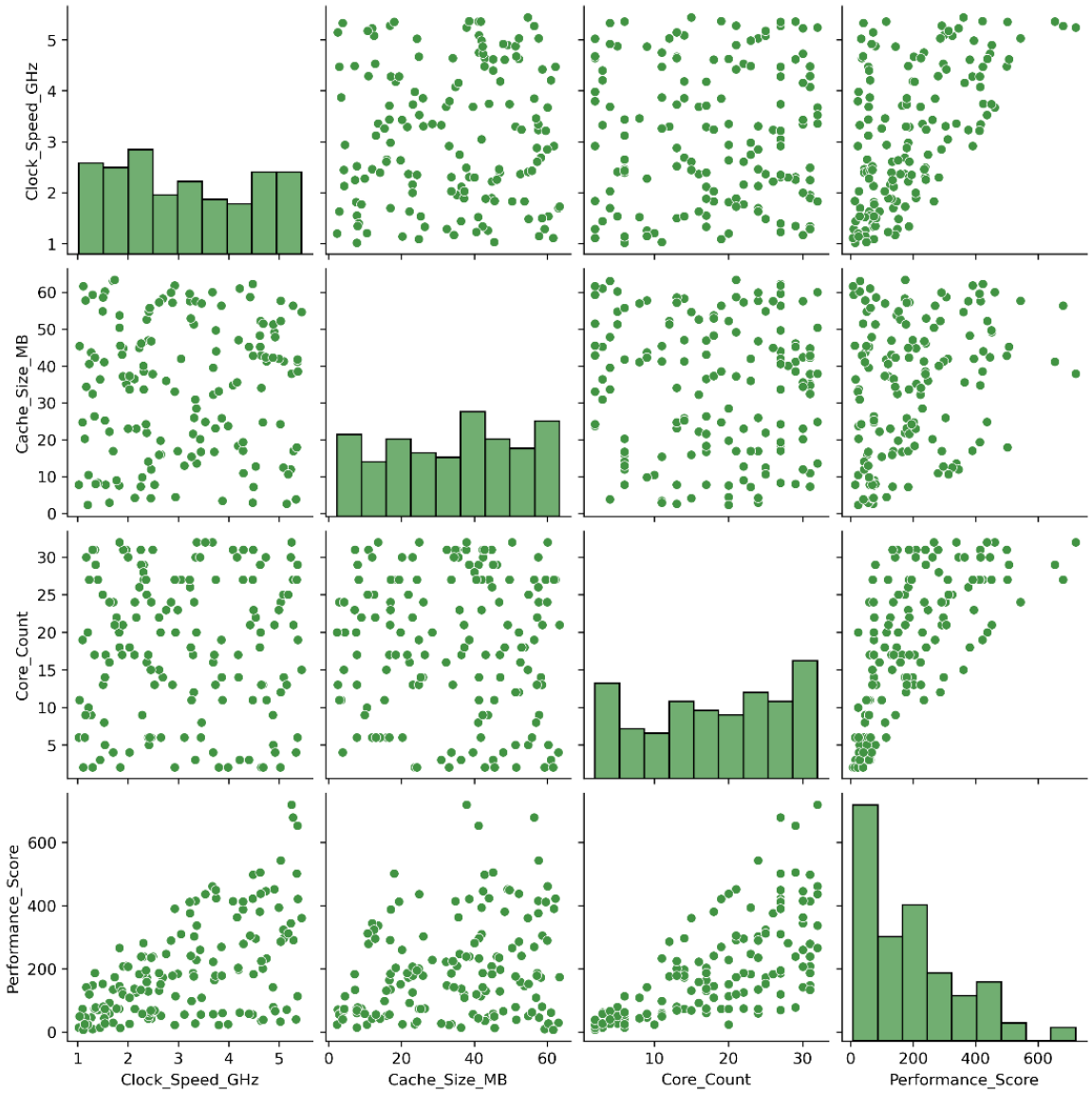


FIGURE 1. Pairwise Relationship Between Computer Architecture Parameters

Figure 1 presents a pairwise scatter plot matrix that visualizes the relationships among the four key parameters in the computer architecture dataset: Clock_Speed_GHz, Cache_Size_MB, Core_Count, and Performance_Score. Each diagonal plot displays the distribution (histogram) of individual variables, while the off-diagonal scatter plots show how pairs of variables interact with one another. The relationships observed indicate that Performance_Score increases with higher Clock_Speed_GHz, larger Cache_Size_MB, and greater Core_Count, suggesting a positive correlation between system performance and these architectural attributes. The plots also reveal a fairly uniform spread of input values, demonstrating that the dataset is well-distributed and free from significant bias or clustering.

Copyright@ REST Publisher

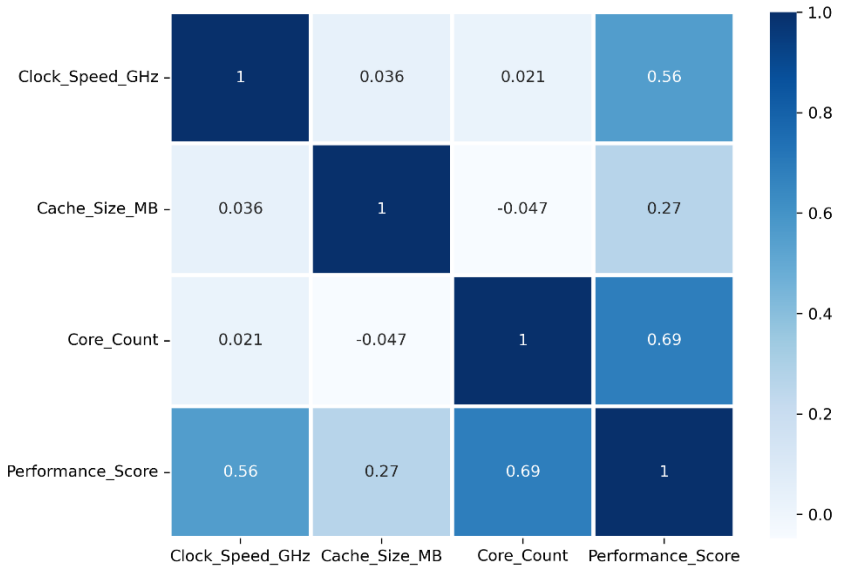


FIGURE 2. Correlation Matrix of Computer Architecture Parameters.

Figure 2 illustrates the correlation heatmap showing the interrelationships among the key parameters: Clock_Speed_GHz, Cache_Size_MB, Core_Count, and Performance_Score. The color gradient, ranging from light blue to dark blue, represents the strength and direction of correlation coefficients, with values closer to 1 indicating a strong positive correlation. The matrix reveals that Performance_Score has the strongest association with Core_Count (0.69) and Clock_Speed_GHz (0.56), suggesting that both processor speed and the number of cores significantly contribute to system performance. In contrast, Cache_Size_MB shows a weaker correlation (0.27) with performance, implying a lesser but still positive influence. The minimal correlations among the independent variables indicate limited multicollinearity, confirming that each parameter contributes uniquely to performance prediction.

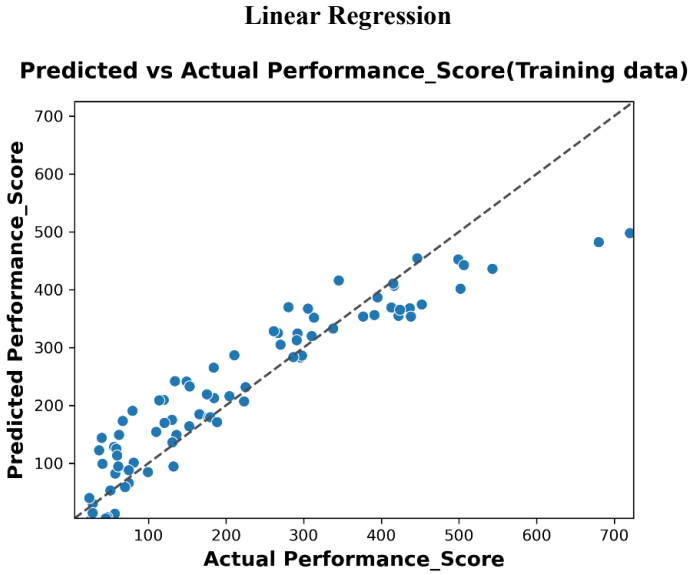


FIGURE 3. Predicted vs Actual Performance Score (Training Data)

Figure 3 illustrates the relationship between the predicted and actual Performance_Score values generated during the training phase of the regression model. Each point represents an individual training instance, while the dashed diagonal line indicates the ideal scenario where predicted values perfectly match the actual values. The close clustering of points around the diagonal line suggests that the model demonstrates strong predictive accuracy, capturing the underlying patterns in the data effectively.

Copyright@ REST Publisher

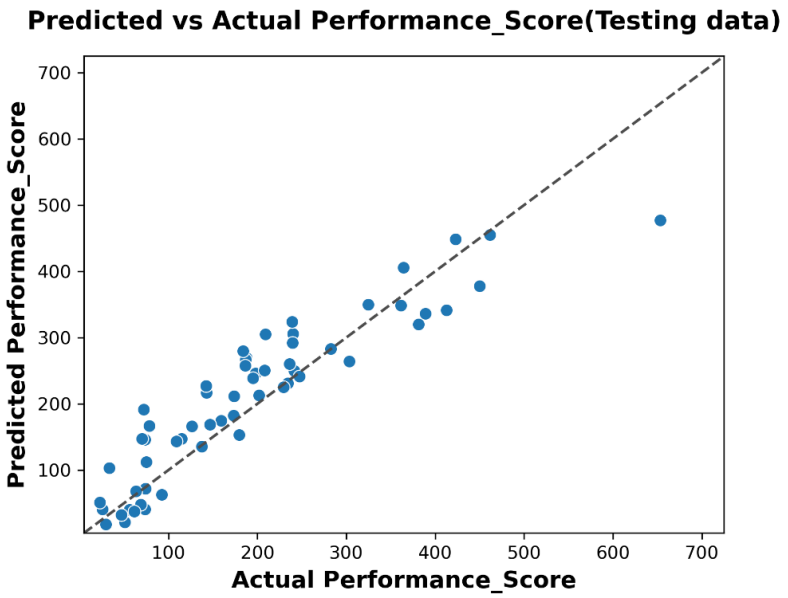


FIGURE 4. Predicted vs Actual Performance Score (Testing Data)

Figure 4 shows Model Performance Evaluation on Test Data: Comparison of Predicted and Actual Performance_Score Values. This visualization assesses the predictive accuracy of the linear regression model, with the ideal fit being a close clustering of data points along the line of perfect prediction (y=x). The spread and pattern of the points are used to evaluate prediction errors and potential biases.

Random Forest Regression

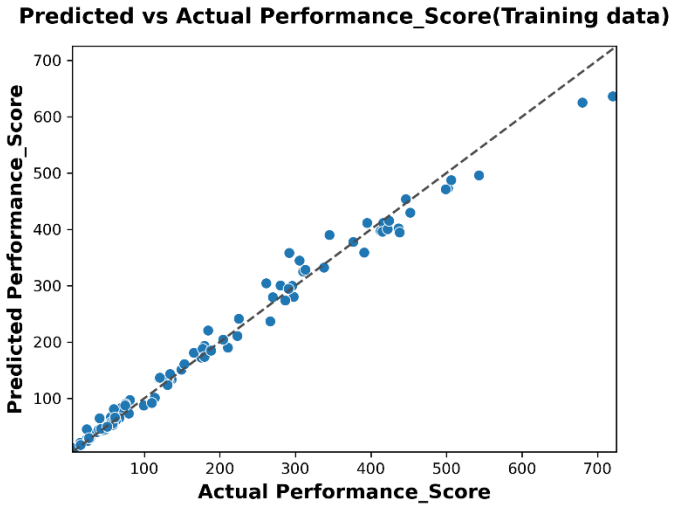


Figure 5. Random Forest Model Fit on Training Data: Predicted vs. Actual Performance_Score

Figure 5 displays the performance of the Random Forest model on the training dataset, comparing its predictions against the true values. The tight clustering of points around the 45-degree reference line signifies a very strong fit, confirming that the model has successfully captured the complex patterns present in the training data. This result is a crucial first step in validating the model's learning capability before assessing its generalizability on unseen testing data.

Copyright@ REST Publisher

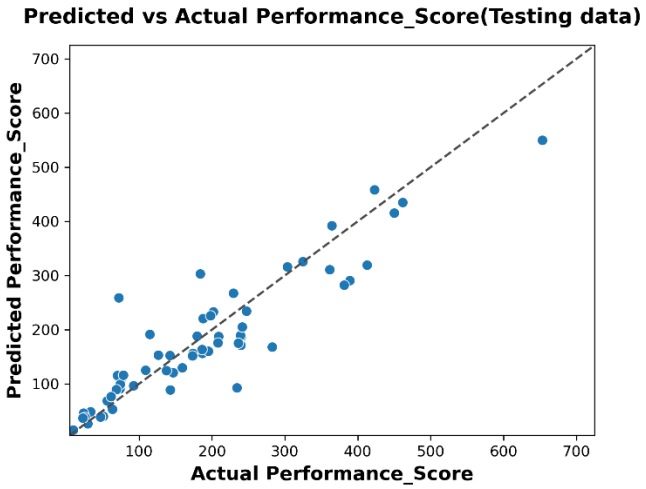


FIGURE 6. Random Forest Model Generalization: Predicted vs. Actual Performance_Score on Testing Data

Figure 6 presents the evaluation of the Random Forest model's predictive performance on the unseen testing dataset. This scatter plot compares the model's predicted Performance_Score values against the true, actual values. The proximity of the data points to the 45-degree line of perfect prediction demonstrates the model's ability to generalize the patterns learned during training to new, unseen data. A strong alignment here, following the good fit observed on the training data (Figure 5), confirms that the model is robust and has not significantly overfitted, validating its utility for making reliable predictions on future data.

TABLE 2. Model Performance Comparison on Training Dataset

Data	Symbol	R2	EVS	MSE	RMSE	MAE	MaxError	MSLE	MedAE
Train	LR	0.845519	0.845519	4257.627	65.25049	50.40028	221.9681	0.0027	2.700482
Train	RFR	0.984027	0.984028	440.2176	20.98136	14.37873	83.56265	0.020228	10.10197

The training results reveal a significant difference in performance between Linear Regression (LR) and Random Forest Regression (RFR). The Linear Regression model achieves an R^2 score of 0.8455, indicating that it explains about 84.5% of the variance in the training data. Its error values—MSE of 4257.63, RMSE of 65.25, and MAE of 50.40—suggest that the model has moderate prediction errors, with a maximum error of 221.97. Although LR performs reasonably well, the Random Forest Regression model shows substantially stronger results. With an R^2 score of 0.9840, RFR explains 98.4% of the variance, demonstrating excellent predictive power. Its error values are significantly lower, with an MSE of 440.22, RMSE of 20.98, and MAE of 14.38, along with a reduced maximum error of 83.56.

TABLE 3. Model Performance Comparison on Testing Dataset

Data	Symbol	R2	EVS	MSE	RMSE	MAE	MaxError	MSLE	MedAE
Test	LR	0.811148	0.811698	18.38937	4.288283	3.449168	11.4477	0.001872	2.875475
Test	RFR	0.837035	0.839764	2866.824	53.54273	38.91511	186.7468	0.109579	27.8534

The test results show notable differences in how Linear Regression (LR) and Random Forest Regression (RFR) generalize to unseen data. The Linear Regression model performs reasonably well, achieving an R^2 value of 0.8111, meaning it explains about 81% of the variance in the test dataset. Its error values are relatively low, with an MSE of 18.39, RMSE of 4.29, and MAE of 3.45, indicating that its predictions remain fairly close to the actual values. The maximum error is 11.45, which further suggests consistent performance without extreme deviations. In contrast, the Random Forest Regression model shows only slightly higher explanatory power with an R^2 of 0.8370, but its error values are significantly larger: an MSE of 2866.82, RMSE of 53.54, and MAE of 38.92, along with a much higher maximum error of 186.75. These results indicate that although RFR captured complex patterns during training, it did not generalize as effectively to the test data.

5. CONCLUSION

The comparative analysis of Linear Regression and Random Forest Regression demonstrates that while the Random Forest model achieves exceptionally high accuracy on the training data, it fails to generalize effectively to unseen test data, indicating clear overfitting. In contrast, Linear Regression delivers more balanced and stable performance across both training and testing sets, with lower error values and more consistent predictive behavior. Although LR captures

simpler linear relationships and shows slightly lower training accuracy, its superior test performance highlights its reliability and robustness for this dataset.

REFERENCES

[1]. Alhakami, Wajdi. "Computational Study of Security Risk Evaluation in Energy Management and Control Systems Based on a Fuzzy MCDM Method." Processes 11, no. 5 (2023): 1366.

[2]. Flynn, Michael. "Computer architecture." Wiley Encyclopedia of Computer Science and Engineering (2007).

[3]. Kozyrakis, Christoforos E., and David A. Patterson. "A new direction for computer architecture research." Computer 31, no. 11 (1998): 24-32.

[4]. Skadron, Kevin, Margaret Martonosi, David I. August, Mark D. Hill, David J. Lilja, and Vijay S. Pai. "Challenges in computer architecture evaluation." Computer 36, no. 8 (2003): 30-36.

[5]. Wu, Nan, and Yuan Xie. "A survey of machine learning for computer architecture and systems." ACM Computing Surveys (CSUR) 55, no. 3 (2022): 1-39.

[6]. KleinOsowski, A. J., and David J. Lilja. "MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research." IEEE Computer Architecture Letters 1, no. 1 (2002): 7-7.

[7]. Skillicorn, David B. "A taxonomy for computer architectures." Computer 21, no. 11 (1988): 46-57.

[8]. Reddi, Vijay Janapa, Alex Settle, Daniel A. Connors, and Robert S. Cohn. "Pin: a binary instrumentation tool for computer architecture research and education." In Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture, pp. 22-es. 2004.

[9]. Lockhart, Derek, Gary Zibrat, and Christopher Batten. "PyMTL: A unified framework for vertically integrated computer architecture research." In 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 280-292. IEEE, 2014.

[10]. Nikolic, Bosko, Zaharije Radivojevic, Jovan Djordjevic, and Veljko Milutinovic. "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization." IEEE Transactions on Education 52, no. 4 (2009): 449-458.

[11]. Branovic, Irina, Roberto Giorgi, and Enrico Martinelli. "WebMIPS: a new web-based MIPS simulation environment for computer architecture education." In Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture, pp. 19-es. 2004.

[12]. Djordjevic, Jovan, Aleksandar Milenkovic, and Nenad Grbanovic. "An integrated environment for teaching computer architecture." IEEE Micro 20, no. 3 (2000): 66-74.

[13]. Gehringer, Edward F., and Barry W. Peddycord III. "The inverted-lecture model: a case study in computer architecture." In Proceeding of the 44th ACM technical symposium on Computer science education, pp. 489-494. 2013.

[14]. Fu, Xiang, Leon Riesebos, Lingling Lao, Carmen G. Almudever, Fabio Sebastiano, Richard Versluis, Edoardo Charbon, and Koen Bertels. "A heterogeneous quantum computer architecture." In Proceedings of the ACM International Conference on Computing Frontiers, pp. 323-330. 2016.

[15]. Allen, Jonathan. "Computer architecture for digital signal processing." Proceedings of the IEEE 73, no. 5 (1985): 852-873.

[16]. Yehezkel, Cecile, William Yurcik, Murray Pearson, and Dean Armstrong. "Three simulator tools for teaching computer architecture: Little Man computer, and RTLSim." Journal on Educational Resources in Computing (JERIC) 1, no. 4 (2001): 60-80.

[17]. Monroe, Christopher, Robert Raussendorf, Alex Ruthven, Kenneth R. Brown, Peter Maunz, L-M. Duan, and Jungsang Kim. "Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects." Physical Review A 89, no. 2 (2014): 022317.

[18]. Djordjevic, Jovan, Bosko Nikolic, and Aleksandar Milenkovic. "Flexible web-based educational system for teaching computer architecture and organization." IEEE Transactions on Education 48, no. 2 (2005): 264-273.

[19]. Beckett, Paul, and Andrew Jennings. "Towards nanocomputer architecture." In Conferences in Research and Practice in Information Technology Series, vol. 19, pp. 141-150. 2002.

[20]. Ditzel, David R., and David A. Patterson. "Retrospective on high-level language computer architecture." In Proceedings of the 7th annual symposium on Computer Architecture, pp. 97-104. 1980.

Copyright@ REST Publisher