IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

# Workload Prediction For Dynamic Voltage And Frequency Scaling For Multicore Systems Using Regression-Based Prediction Models

[1]Vinolya S, [2] Dr. R. Jayagowri

[1]Student, [2]Professor

[1]Department of ECE,

[1]BMSCE, Bangalore, India

***Abstract:*** Artificial Intelligence (AI) workloads place heavy demands on computational hardware, necessitating advanced power management strategies like Dynamic Voltage and Frequency Scaling (DVFS). This paper presents a workload prediction model for multicore systems based on metrics derived from simulations using the PARSEC benchmark suite and gem5. The proposed regression-based model uses system-level metrics such as IPC, CPI, cache misses, and memory access rates to accurately predict workload performance. The results show a high degree of accuracy in workload prediction, demonstrating the efficacy of machine learning models in optimizing energy efficiency through DVFS.

**Index Terms** - Dynamic Voltage and Frequency Scaling (DVFS), Workload Prediction, PARSEC Benchmark, Multicore Systems, Regression Model, Machine Learning.

## I. INTRODUCTION

The rapid advancements in Artificial Intelligence (AI) and machine learning (ML) have transformed industries ranging from healthcare to finance and autonomous systems. The increasing complexity of AI algorithms, such as deep neural networks (DNNs) and reinforcement learning models, demands substantial computational resources to train and deploy these models effectively. These workloads are typically characterized by intensive matrix operations, large data sets, and frequent memory access, which place considerable strain on hardware architectures. Consequently, AI chips, including Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and custom AI accelerators, have become indispensable for efficiently handling the growing computational demands of AI workloads.

However, the inherent variability of AI workloads presents significant challenges in maintaining optimal performance, especially when deploying these systems on energy-efficient multicore architectures. AI workloads are typically dynamic and exhibit varying levels of computational intensity depending on the specific task being performed, such as image classification, natural language processing, or predictive modelling. These tasks involve different resource requirements, including CPU utilization, memory bandwidth, cache usage, and floating-point arithmetic operations, which fluctuate throughout the execution of the AI models. As a result, managing performance and power consumption across a range of workloads becomes critical for maintaining efficiency.

Dynamic Voltage and Frequency Scaling (DVFS) is one of the most widely adopted techniques for optimizing energy efficiency in multicore processors. DVFS allows the system to dynamically adjust the operating voltage and clock frequency of the processor based on real-time workload demands. By scaling down the voltage and frequency during periods of low computational demand, the system can significantly reduce power consumption without sacrificing performance. Conversely, during periods of high demand, DVFS increases the voltage and frequency to ensure that performance requirements are met. Despite the proven effectiveness of DVFS, accurately predicting workload performance and determining the optimal frequency and voltage levels remains a complex challenge, particularly in systems with diverse AI workloads.

To address this challenge, researchers have turned to machine learning models to predict system performance and guide the application of DVFS. By leveraging detailed performance metrics such as instructions per cycle (IPC), cycles per instruction (CPI), cache misses, memory access latencies, and floating-point operations, machine learning algorithms can analyse workload patterns and predict future demands. These predictions can be used to adjust the system's operating parameters dynamically, ensuring that energy is conserved without compromising the execution speed of critical AI workloads.

The **PARSEC** (Princeton Application Repository for Shared-Memory Computers) benchmark suite is a widely used tool for simulating realistic workloads on multicore systems. PARSEC offers a variety of applications that cover a broad spectrum of computing domains, including financial modelling, computer vision, and data mining. These applications present diverse workload characteristics, making PARSEC an ideal benchmark for evaluating system performance under different AI workloads. Moreover, the gem5 simulator, a modular platform for simulating computer architectures, provides an environment to model CPU and memory interactions in detail. Together, PARSEC and gem5 enable the collection of granular performance data, which can be used to train machine learning models for workload prediction.

In this paper, we explore the development of a workload prediction model for Dynamic Voltage and Frequency Scaling (DVFS) on multicore systems. The study aims to build an efficient regression-based model that can predict the workload of a system based on performance metrics gathered from gem5 simulations of the PARSEC benchmark suite. The model is designed to optimize system performance and energy consumption by dynamically adjusting voltage and frequency in response to workload demands.

This research addresses several key challenges in the field of energy-efficient computing. First, it provides insights into how different AI workloads impact system performance, highlighting the critical factors that influence workload intensity, such as CPU efficiency, cache performance, and memory usage. Second, the machine learning models developed in this study demonstrate the feasibility of using predictive techniques to guide DVFS decisions, improving both energy efficiency and performance in real-world AI applications. Finally, the methodology presented here offers a framework for further research into the intersection of AI workload management, machine learning, and computer architecture optimization.

The remainder of this paper is structured as follows. Section 2 provides a detailed overview of the methodology, including tools used, dataset collection, and model development. Section 3 presents the results of the workload prediction model and discusses the effectiveness of the approach. Section 4 concludes the paper with a summary of findings and future research directions.

## II. LITERATURE SURVEY

In recent years, Dynamic Voltage and Frequency Scaling (DVFS) has emerged as a prominent technique to optimize energy efficiency in multicore systems, especially in the context of AI workloads. Liu and Karanth in [1] proposed a hardware accelerator architecture that applies DVFS to improve energy efficiency, focusing on machine learning-based voltage-frequency prediction and granularity at different hardware levels. This approach demonstrated the effectiveness of combining power gating techniques with proactive DVFS state selection.

## III. METHODOLOGY

This section describes the methodology adopted in building a regression-based workload prediction model for Dynamic Voltage and Frequency Scaling (DVFS) in multicore systems. The process involves simulating workloads using the PARSEC benchmark suite, collecting and processing performance metrics from the gem5 simulator, and developing machine learning models for predicting workloads based on system-level parameters. The methodology is divided into four major parts: tools used, dataset collection, dataset pre-processing, and workload prediction model development.

### A. Tools Used

#### 1. Gem5 Simulator

The gem5 simulator is a widely-used, modular simulation platform designed for computer architecture research, encompassing system-level and microarchitecture-level simulations. It allows for detailed modeling of the behavior of computer systems under different workloads. Gem5 supports various Instruction Set Architectures (ISAs) including x86, ARM, RISC-V, and SPARC, making it highly flexible for a wide range of research and development tasks.

In this project, we focused on the x86 architecture due to its widespread use in modern multicore systems. The gem5 platform was configured to simulate a system with a TimingSimpleCPU, which is a relatively simple out-of-order processor ideal for capturing CPU-related performance metrics. We modeled both instruction and data caches (L1i and L1d), system memory (DDR3), and bus architectures to simulate realistic multicore performance scenarios.

The key features of the system configuration include:

- CPU type: TimingSimpleCPU
- Clock frequency: 1 GHz
- Memory size: 512 MB (DDR3)
- Caches: L1 Instruction Cache (32kB, 2-way associative), L1 Data Cache (32kB, 2-way associative)
- Simulation Output: Various performance metrics including cache hits/misses, memory accesses, instruction execution rates, and floating-point operations.

Figure 1 shows the System built to run the PARSEC Benchmark Suite applications.

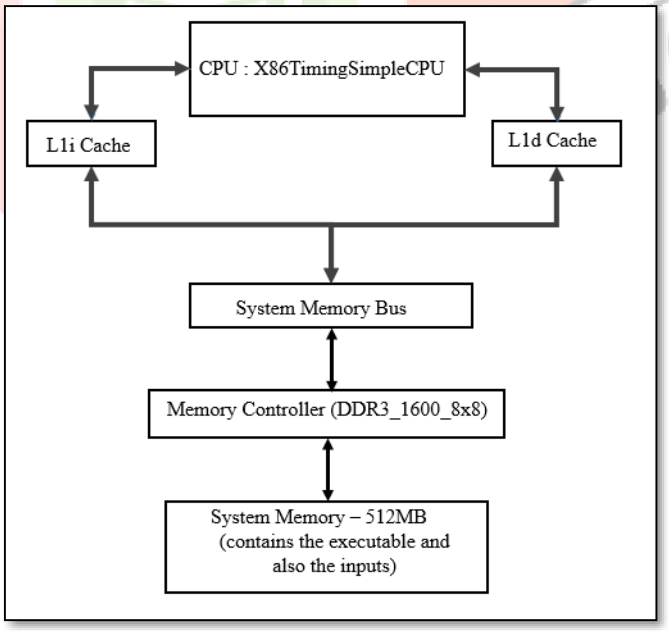

CPU : X86TimingSimpleCPU
L1i Cache
L1d Cache
System Memory Bus
Memory Controller (DDR3_1600_8x8)
System Memory – 512MB (contains the executable and also the inputs)

*Figure 1: System built to run the PARSEC Benchmark Suite applications*

### 2. PARSEC Benchmark Suite

The PARSEC (Princeton Application Repository for Shared-Memory Computers) benchmark suite is a widely used tool for evaluating the performance of multicore systems. It offers a comprehensive set of real-world applications that span domains such as financial analysis, computer vision, and data mining, making it an ideal benchmark for simulating diverse workload characteristics.
The following applications were chosen from the PARSEC benchmark suite for this study:

- ***Blackscholes:*** A financial analysis application that uses the Black-Scholes equation to price options. It is highly computation-intensive, making it an ideal candidate for evaluating CPU performance.
- ***Bodytrack:*** A computer vision application used for tracking 3D body movements. It is moderately memory-intensive.
- ***Freqmine:*** A data mining application that uses the FP-growth algorithm to discover frequent itemsets. It is I/O and memory intensive.
- ***Swaptions:*** A financial application that prices swaptions (interest rate derivatives) using a Monte Carlo simulation. It involves both computation and memory accesses.

Each application was run with three different input sets—test, simdev, and simsmall—to capture variations in workload intensity and performance. These input sets provide progressively larger workloads, from basic functionality tests to more complex, real-world simulations.

### 3. Jupyter Notebook for Data Analysis

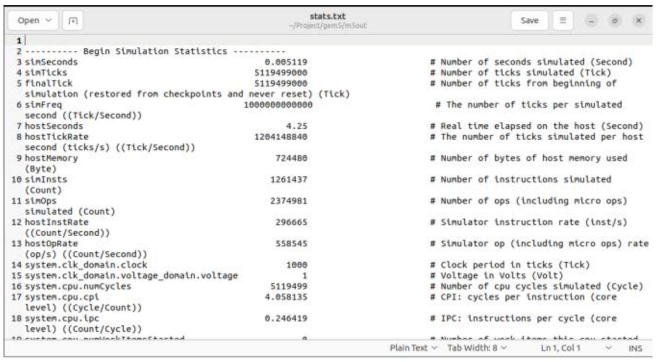
The Jupyter Notebook environment was chosen for the data analysis, visualization, and machine learning modeling due to its interactive interface and support for multiple programming languages. It allowed seamless integration of Python libraries like Pandas, NumPy, Scikit-learn, and Matplotlib, which were essential for dataset manipulation, model training, and performance evaluation.
The key libraries and their purposes in this study were:

- ***Pandas***: Used for handling and processing large datasets, including loading the performance data extracted from gem5 simulations.
- ***NumPy***: Employed for numerical computations and manipulation of array structures.
- ***Scikit-learn:*** Provided the machine learning algorithms, including linear regression and Random Forest models used to predict workloads.
- ***Matplotlib/Seaborn:*** Used for visualizing the results of data analysis, such as plotting the feature importance, workload predictions, and performance metrics.

## B. Dataset Collection

### 1. gem5 Simulation and Metric Extraction

The process of collecting simulation statistics begins with configuring the gem5 simulator to run the selected applications from the PARSEC benchmark suite. Each application was executed with different input sets to generate a diverse range of workload scenarios. For each simulation, gem5 outputs a comprehensive set of system performance metrics, which are stored in a stats.txt file.



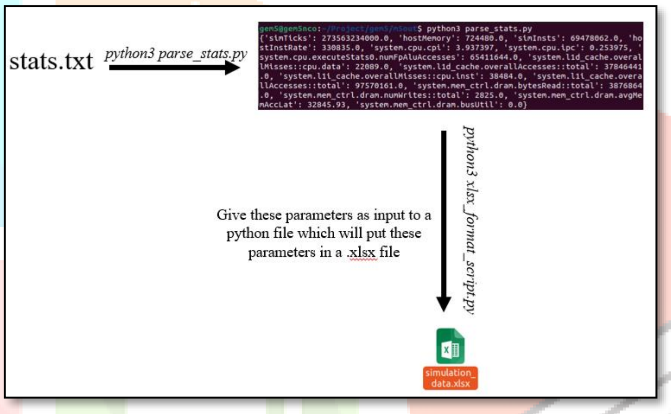
```

---------- Begin Simulation Statistics ----------
simSeconds                                   0.005119                       # Number of seconds simulated (Second)
simTicks                                   5119499000                       # Number of ticks simulated (Tick)
finalTick                                  5119499000                       # Number of ticks from beginning of
simulation (restored from checkpoints and never reset) (Tick)
simFreq                                 1000000000000                        # The number of ticks per simulated
second ((Tick/Second))
hostSeconds                                      4.25                       # Real time elapsed on the host (Second)
hostTickRate                               1204148840                       # The number of ticks simulated per host
second (ticks/s) ((Tick/Second))
hostMemory                                     724480                       # Number of bytes of host memory used
(Byte)
simInsts                                      1261437                       # Number of instructions simulated
(Count)
simOps                                        2374981                       # Number of ops (including micro ops)
simulated (Count)
hostInstRate                                   296665                       # Simulator instruction rate (inst/s)
((Count/Second))
hostOpRate                                     558545                       # Simulator op (including micro ops) rate
(op/s) ((Count/Second))
system.clk_domain.clock                          1000                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage            1                       # Voltage in Volts (Volt)
system.cpu.numCycles                          5119499                       # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                               4.058135                       # CPI: cycles per instruction (core
level) ((Cycle/Count))
system.cpu.ipc                               0.246419                       # IPC: instructions per cycle (core
level) ((Count/Cycle))
```

*Figure 2: stats.txt file contents*

As shown in figure 2, the stats.txt file contains over 700 parameters, capturing everything from CPU performance (IPC, CPI) to memory operations (cache accesses, cache misses, memory read/write operations). However, not all these metrics are equally relevant to predicting workload performance, so the following key metrics were selected based on their importance in previous studies and general computer architecture principles:

- ***system.cpu.ipc (Instructions Per Cycle)***: A key measure of CPU efficiency. Higher IPC values typically indicate better CPU performance.
- ***system.cpu.cpi (Cycles Per Instruction):*** Reflects the number of cycles needed to execute an instruction, with higher values indicating more strain on the system.
- ***system.l1d_cache.overallMisses:*** The number of data cache misses, which causes delays in execution as the CPU fetches data from slower memory levels.
- ***system.l1i_cache.overallMisses:*** The number of instruction cache misses, which slows down the execution of instructions.
- ***system.cpu.executeStats0.numFpAluAccesses:*** The number of floating-point ALU operations, indicating the level of computational intensity.
- ***system.mem_ctrl.dram.bytesRead and bytesWritten***: Total bytes read from and written to DRAM, indicating memory bandwidth usage.
- ***system.mem_ctrl.dram.avgMemAccLat:*** Average memory access latency, providing insights into memory subsystem bottlenecks.
- ***simInsts and simTicks:*** General statistics on the number of instructions executed and the simulation time in ticks.


stats.txt
python3 parse_stats.py
python3 xlsx_format_script.py
Give these parameters as input to a python file which will put these parameters in a .xlsx file
simulation_data.xlsx

```
gem5@gem5nco:~/Project/gem5/m5out$ python3 parse_stats.py
{'simTicks': 273563234000.0, 'hostMemory': 724480.0, 'simInsts': 69478062.0, 'ho
stInstRate': 330835.0, 'system.cpu.cpi': 3.937397, 'system.cpu.ipc': 0.253975, '
system.cpu.executeStats0.numFpAluAccesses': 65411644.0, 'system.l1d_cache.overal
lMisses::cpu.data': 22089.0, 'system.l1d_cache.overallAccesses::total': 37846441
.0, 'system.l1i_cache.overallMisses::cpu.inst': 38484.0, 'system.l1i_cache.overa
llAccesses::total': 97570161.0, 'system.mem_ctrl.dram.bytesRead::total': 3876864
.0, 'system.mem_ctrl.dram.numWrites::total': 2825.0, 'system.mem_ctrl.dram.avgMe
mAccLat': 32845.93, 'system.mem_ctrl.dram.busUtil': 0.0}
```

*Figure 3: Process of converting stats.txt file to simulation_data.xlsx file*

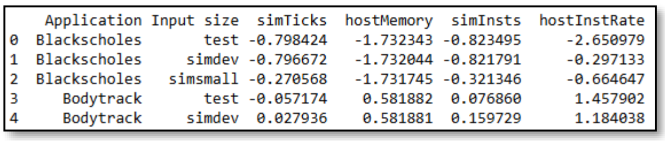
These selected metrics were stored in a structured dataset (saved as an Excel file) for further analysis using the procedure as shown in figure 3.

## *C. Dataset Pre-Processing*

The raw dataset collected from the gem5 simulations required pre-processing before it could be used for machine learning model development. The primary steps in the pre-processing pipeline included:

### *1. Normalization*

Since the raw performance metrics are on different scales (e.g., IPC is typically less than 1, while memory accesses can be in the millions), the dataset was normalized to bring all values onto a common scale. We applied the **Standard Scaler** from Scikit-learn, which normalizes the data such that each feature has a mean of 0 and a standard deviation of 1. This ensures that no single feature dominates the model purely due to its larger numerical range. Figure 4 shows the header of the normalized dataset.



```
    Application Input size  simTicks  hostMemory  simInsts  hostInstRate
0  Blackscholes       test -0.798424   -1.732343 -0.823495     -2.650979
1  Blackscholes     simdev -0.796672   -1.732044 -0.821791     -0.297133
2  Blackscholes   simsmall -0.270568   -1.731745 -0.321346     -0.664647
3     Bodytrack       test -0.057174    0.581882  0.076860      1.457902
4     Bodytrack     simdev  0.027936    0.581881  0.159729      1.184038
```

*Figure 4: Normalized Dataset after applying Standard Scaler*

### *2. Data Augmentation*

The original dataset had only 12 data points (3 input sets for 4 applications), which is insufficient for developing a robust regression model. To increase the number of data points, we applied two augmentation techniques:

***Random Perturbation:*** Small amounts of noise (±5%) were added to each feature in the dataset to create new, slightly modified data points that preserve the overall workload characteristics.

***Interpolation:*** New data points were generated by interpolating between existing data points. This technique helps to create additional workload scenarios that lie between the extremes captured by the original dataset.

After applying these techniques, the size of the dataset increased to 100 data points, providing sufficient diversity for training and testing the machine learning model.

## *D. Workload Prediction Model Development*

### *1. Linear Regression Model*

The first model developed was a simple linear regression model. Linear regression is a straightforward approach that attempts to model the relationship between a dependent variable (workload) and one or more independent variables (system metrics). The goal was to predict the overall workload score based on the collected metrics such as IPC, CPI, and cache misses.

Each of the selected metrics was treated as an independent variable, while the workload was the dependent variable. Initially, equal weights were assigned to each metric in the linear regression model, which resulted in a perfect R² score of 1.0—a sign of overfitting as shown in figure 5.

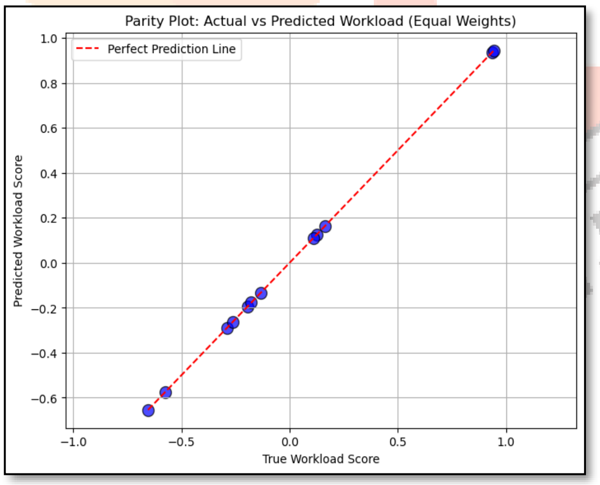


*Figure 5: Plot of perfect prediction line versus True Workload Score*

This graph shows that the prediction of workload will happen only if the newly given input's workload matches with the workload which is already calculated and kept in the dataset. This means that the prediction model went into Overfitting model. These kind of models are not suitable for workload predictions as they cannot understand and analyse new data.

### *2. Random Forest Regressor and Feature Importance*

To mitigate overfitting and improve the model's generalizability, a Random Forest Regressor was used to evaluate the relative importance of each metric. The Random Forest algorithm builds multiple decision trees and averages their predictions, providing a more robust model that can handle non-linear relationships between variables.

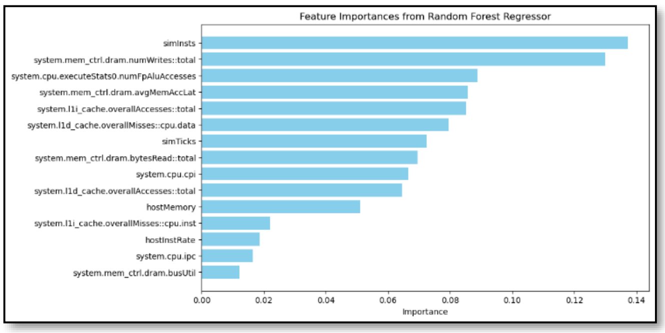


*Figure 6: Feature importance extracted from the Random Forest Regressor*

The feature importance analysis as shown from table III-I and figure 6 gave a detailed analysis such that parameters such as system.cpu.ipc,system.cpu.cpi,system.cpu.executeStats0.numFpAluAccesses, and cache misses had the highest impact on workload prediction. This allowed us to fine-tune the linear regression model by assigning appropriate weights to each feature based on its importance.

### *3. Workload Calculations*

The parameters which has higher influence over the workload were selected. They are:
•***system.cpu.ipc (Instructions Per Cycle):*** A high IPC indicates efficient CPU usage. A lower IPC under the same workload suggests a heavier system load.
•***system.cpu.cpi (Cycles Per Instruction):*** Higher CPI means more cycles are needed per instruction, which reflects a higher workload for the system.
•***system.cpu.executeStats0.numFpAluAccesses (Floating Point ALU Accesses):*** High numbers of ALU accesses suggest intensive compute operations.
•***system.l1d_cache.overallMisses::cpu.data (L1 Data Cache Misses):*** Cache misses result in more memory access, increasing the load.
•***system.l1i_cache.overallMisses::cpu.inst (L1 Instruction Cache Misses):*** Similar to data cache misses, but focusing on instructions, increasing memory access and system load.

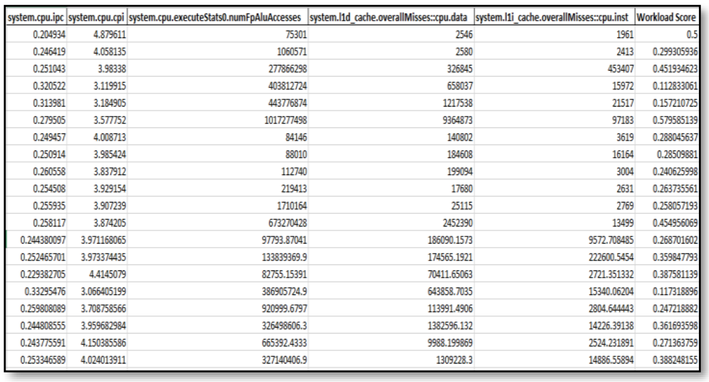
The workload was thus calculated using the formula:

$$\boldsymbol{Workload} = \left(\boldsymbol{wipc} * \frac{\mathbf{1}}{\boldsymbol{ipc}}\right) + (\,\boldsymbol{wcpi} * \boldsymbol{cpi}) + (\boldsymbol{wfp} * \ \mathbf{numFpAluAccesses}) + (\,\boldsymbol{wl1d} * \mathbf{l1d}_{\mathbf{cache}_{\mathbf{misses}}}) + (\mathbf{wl1i} * \ \mathbf{l1i}_{\mathbf{cache}_{\mathbf{misses}}})$$

where,
→ $\boldsymbol{wipc}, \boldsymbol{wcpi}, \boldsymbol{wfp}, \boldsymbol{wl1d}, \mathbf{wl1i}$ are the weights for each parameter, based on their relative importance.
→ $\frac{\mathbf{1}}{\boldsymbol{ipc}}$ is used because a higher IPC means lower workload, and thus its inverse correlates directly with workload.

## 4. Random Perturbation and Interpolation

The dataset that was used in Trial – 1 have only 12 datapoints, but in reality for a linear regression model, minimum of about 100 datapoints are required. Therefore to increase the datapoints in the dataset the combination of two methods were chosen. They are:



| system.cpu.ipc | system.cpu.cpi | system.cpu.executeStats0.numFpAluAccesses | system.l1d_cache.overallMisses::cpu.data | system.l1i_cache.overallMisses::cpu.inst | Workload Score |
|---|---|---|---|---|---|
| 0.204934 | 4.879611 | 75301 | 2546 | 1961 | 0.5 |
| 0.246419 | 4.058135 | 1060571 | 2580 | 2413 | 0.299305936 |
| 0.251043 | 3.98338 | 277866298 | 326845 | 453407 | 0.451934623 |
| 0.320522 | 3.119915 | 403812724 | 658037 | 15972 | 0.112833061 |
| 0.313981 | 3.184905 | 443776874 | 1217538 | 21517 | 0.157210725 |
| 0.279505 | 3.577752 | 1017277498 | 9364873 | 97183 | 0.579585139 |
| 0.249457 | 4.008713 | 84146 | 140802 | 3619 | 0.288045637 |
| 0.250914 | 3.985424 | 88010 | 184608 | 16164 | 0.28509881 |
| 0.260558 | 3.837912 | 112740 | 199094 | 3004 | 0.240625998 |
| 0.254508 | 3.929154 | 219413 | 17680 | 2631 | 0.263735561 |
| 0.255935 | 3.907239 | 1710164 | 25115 | 2769 | 0.258057193 |
| 0.258117 | 3.874205 | 673270428 | 2452390 | 13499 | 0.454956069 |
| 0.244380097 | 3.971168065 | 97793.87041 | 186090.1573 | 9572.708485 | 0.268701602 |
| 0.252465701 | 3.973374435 | 133839369.9 | 174565.1921 | 222600.5454 | 0.359847793 |
| 0.229382705 | 4.4145079 | 82755.15391 | 70411.65063 | 2721.351332 | 0.387581139 |
| 0.33295476 | 3.066405199 | 386905724.9 | 643858.7035 | 15340.06204 | 0.117318896 |
| 0.259808089 | 3.708758566 | 920999.6797 | 113991.4906 | 2804.644443 | 0.247218882 |
| 0.244808555 | 3.959682984 | 326498606.3 | 1382596.132 | 14226.39138 | 0.361693598 |
| 0.243775591 | 4.150385586 | 665392.4333 | 9988.199869 | 2524.231891 | 0.271363759 |
| 0.253346589 | 4.024013911 | 327140406.9 | 1309228.3 | 14886.55894 | 0.388248155 |

*Figure 7: New Dataset after applying Perturbation and Interpolation*

•***Random Perturbation:*** Varying the existing data by adding small amounts of noise to each feature (parameter).

•**Interpolation (Between Existing Data Points):** Generating new datapoints by interpolating between existing values.
Figure 7 shows the new dataset after applying Perturbation and Interpolation.

## 5. Model Training and Validation

After 'n' number of trails on assigning different weights to the final set of selected parameters, the final model achieved an $R^2$ score of 0.916 and a Mean Squared Error (MSE) of 0.000428, indicating a high level of accuracy in predicting workload performance.

## IV. RESULTS

**Analysis – 1:** The entire dataset is split into 80% training and 20% testing. The predictions on test set is as shown below. The dotted line is the actual workload, whereas the blue dots represent the predicted workload. Figure 8 shows the Train vs. Test dataset prediction model output. Some of the test points are scattered whereas the others are very close by to the predicted line.

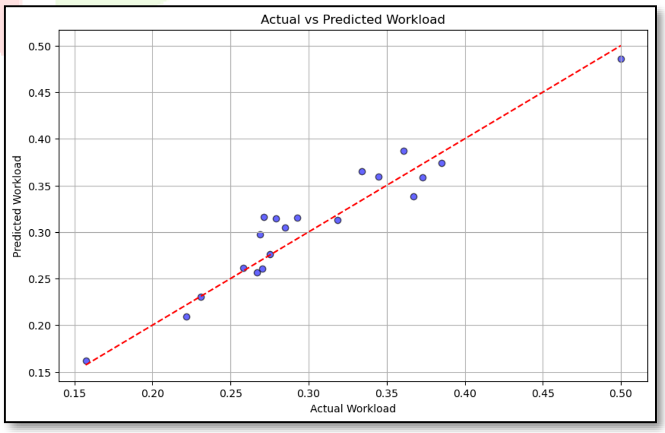
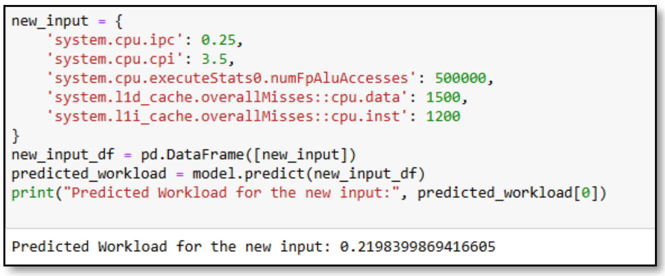


*Figure 8: Test Dataset Prediction*

**Analysis – 2:** Applying new inputs to the dataset to test for the workload prediction. As shown in figure 9, the model is able to predict the workload of the system when provided with new set of input parameters.



```
new_input = {
    'system.cpu.ipc': 0.25,
    'system.cpu.cpi': 3.5,
    'system.cpu.executeStats0.numFpAluAccesses': 500000,
    'system.l1d_cache.overallMisses::cpu.data': 1500,
    'system.l1i_cache.overallMisses::cpu.inst': 1200
}
new_input_df = pd.DataFrame([new_input])
predicted_workload = model.predict(new_input_df)
print("Predicted Workload for the new input:", predicted_workload[0])

Predicted Workload for the new input: 0.2198399869416605
```

*Figure 9: Workload Prediction for new input Dataframe*

**Analysis – 3:** Figure 10 shows the comparison of the actual workload (in blue) and the predicted workload (in orange) over a simulated index. The actual workload shows the true values, while the predicted workload illustrates how well the model forecasts these values. A close alignment indicates a good model performance.

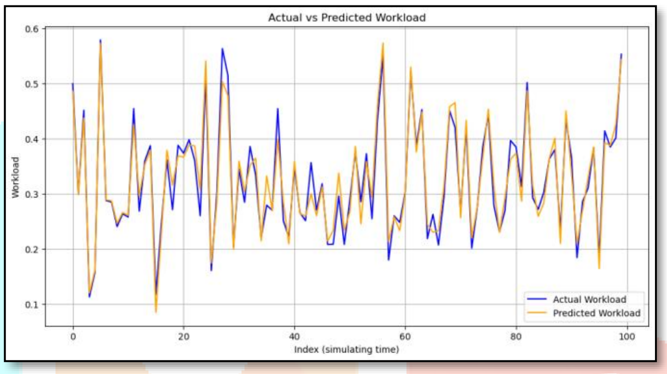
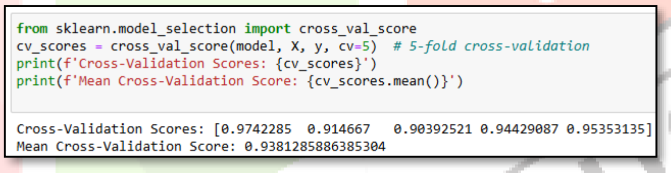


*Figure 10: Comparison of Actual vs. Predicted Workload by Index*

**Analysis – 4:** Evaluating Model Performance through 5-Fold Cross-Validation



```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(model, X, y, cv=5)  # 5-fold cross-validation
print(f'Cross-Validation Scores: {cv_scores}')
print(f'Mean Cross-Validation Score: {cv_scores.mean()}')

Cross-Validation Scores: [0.9742285  0.914667   0.90392521 0.94429087 0.95353135]
Mean Cross-Validation Score: 0.9381285886385304
```

*Figure 11: Performance evaluation through 5-Fold Cross-Validation*

The cross-validation scores represent the model's performance on different subsets of the data during the training process. Each score reflects the model's accuracy on a specific fold. The mean cross-validation score of approximately 0.938 indicates that, on average, the model is expected to explain about 93.8% of the variance in the target variable across various subsets of the dataset. From the figure 11 one can conclude that the model generalizes well and is likely to perform consistently across different training and validation splits.

**Analysis – 5:** K-Means Clustering of Workload Data Based on CPU Efficiency and Performance visualizes the results of K-Means clustering applied to the workload data, using system.cpu.ipc and Workload Score as the two dimensions in the form of scatter plot. Each point on the graph as shown in the figure 12 represents an observation, colored according to its assigned cluster.

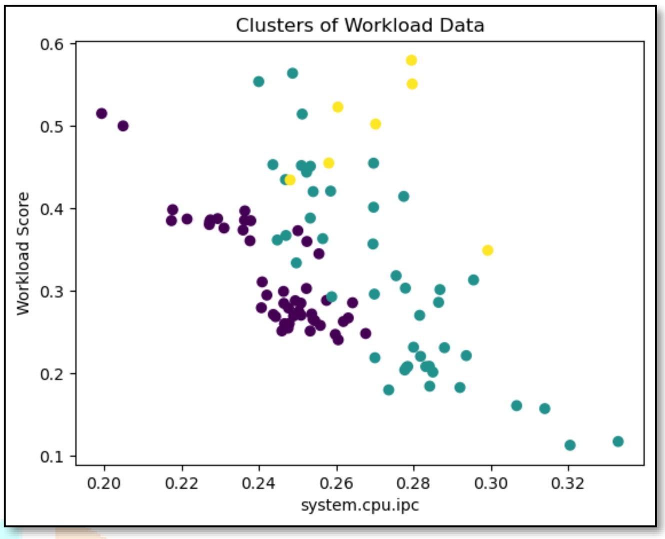


*Figure 12: K-Means Clustering of Workload Data*

***Clusters:*** The plot contains three distinct clusters, indicated by different colors. This separation suggests that there are groups of observations that exhibit similar characteristics in terms of CPU efficiency (system.cpu.ipc) and workload performance (Workload Score).

***Interpretation:*** Points within the same cluster are likely to share similar performance metrics, which may indicate specific patterns or behaviors in workload management.

The proximity of points within a cluster signifies that the respective workloads have similar CPU efficiency, providing insights into optimal configurations for system performance.

## V. CONCLUSION

The exploration of power consumption dynamics in AI chips is critical for advancing the field of energy-efficient computing. By utilizing the PARSEC benchmark suite and the gem5 simulation tool, we have established a robust dataset that captures various workload characteristics. This dataset serves as a foundation for developing predictive models aimed at forecasting power consumption based on dynamic workload behaviors. The initial regression analyses have demonstrated the potential for accurately predicting the workload on the system using a set of relevant features.

## REFERENCES

[1] S. Liu and A. Karanth, "Dynamic Voltage and Frequency Scaling to Improve Energy-Efficiency of Hardware Accelerators," 2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC), Bengaluru, India, 2021, pp. 232-241, doi: 10.1109/HiPC53243.2021.00037.

[2] C. Lin, K. Wang, Z. Li, and Y. Pu, "A Workload-Aware DVFS Robust to Concurrent Tasks for Mobile Devices," Proceedings of the 29th Annual International Conference on Mobile Computing and Networking, Association for Computing Machinery, USA, Article 19, 2021, pp. 1-16.

[3] M. Gupta, L. Bhargava, and S. Indu, "Dynamic workload-aware DVFS for multicore systems using machine learning," Computing, vol. 103, pp. 1747–1769, 2021.

[4] H. Huang, M. Lin, L. T. Yang, and Q. Zhang, "Autonomous Power Management With Double-Q Reinforcement Learning Method," IEEE Transactions on Industrial Informatics, vol. 16, pp. 1938-1946, 2020.

[5] B. K. Reddy, A. Singh, D. Biswas, G. Merrett, and B. Al-Hashimi, "Inter-Cluster Thread-to-Core Mapping and DVFS on Heterogeneous Multi-Cores," IEEE Transactions on Multi-Scale Computing Systems, 2017, pp. 1-1, doi: 10.1109/TMSCS.2017.2755619.

[6] Y. Ma, T. Chantem, R. Dick, and X. Hu, "Improving System-Level Lifetime Reliability of Multicore Soft Real-Time Systems," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2017, pp. 1-11, doi: 10.1109/TVLSI.2017.2669144.

[7] Y. Turakhia, G. Liu, S. Garg, and D. Marculescu, "Thread Progress Equalization: Dynamically Adaptive Power and Performance Optimization of Multi-threaded Applications," IEEE Transactions on Computers, vol. PP, 2016, doi: 10.1109/TC.2016.2608951.

[8] R. Cochran, C. Hankendi, A. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and thread packing under power caps," Proceedings of the Annual International Symposium on Microarchitecture, 2011, doi: 10.1145/2155620.2155641.

[9] C. Isci, A. Buyuktosunoglu, C. Y. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), USA, 2006, pp. 347-358, doi: 10.1109/MICRO.2006.8.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," 2008 International Conference on Parallel Architectures and Compilation Techniques (PACT), Toronto, ON, Canada, 2008, pp. 72-81.

[11] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times," Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 1, 2004, pp. 4-9.