


CONESCAPANHONDURAS2025paper30.pdf

 Institute of Electrical and Electronics Engineers (IEEE)

Document Details

Submission ID

trn:oid:::14348:477750808

Submission Date

Jul 31, 2025, 11:52 PM CST

Download Date

Aug 12, 2025, 12:28 PM CST

File Name

CONESCAPANHONDURAS2025paper30.pdf

File Size

202.6 KB

6 Pages





4,424 Words

25,245 Characters




13% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

-  **34 Not Cited or Quoted 12%**
Matches with neither in-text citation nor quotation marks
-  **3 Missing Quotations 1%**
Matches that are still very similar to source material
-  **0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 10%  Internet sources
- 11%  Publications
- 0%  Submitted works (Student Papers)

Integrity Flags





0 Integrity Flags for Review

No suspicious text manipulations found.




Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

-  **34 Not Cited or Quoted** 12%
Matches with neither in-text citation nor quotation marks
-  **3 Missing Quotations** 1%
Matches that are still very similar to source material
-  **0 Missing Citation** 0%
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted** 0%
Matches with in-text citation present, but no quotation marks

Top Sources

- 10%  Internet sources
- 11%  Publications
- 0%  Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	www.arxiv-vanity.com	1%
2	Internet	previews.americangeosciences.org	1%
3	Internet	export.arxiv.org	1%
4	Internet	oaktrust.library.tamu.edu	<1%
5	Publication	Archit Gajjar, Priyank Kashyap, Aydin Aysu, Paul Franzon, Sumon Dey, Chris Chen...	<1%
6	Internet	www.pure.ed.ac.uk	<1%
7	Publication	Jahib Nawfal, Avinash Mungur. "Performance Evaluation Between Tiny YOLOv3 an...	<1%
8	Publication	Cavide Balki Gemirter, Gurhan Kucuk. "Fine-Tuning Throughput and QoS on SMT ...	<1%
9	Internet	zenodo.org	<1%
10	Publication	Ashkan Tousi, Mikel Lujan. "Comparative Analysis of Machine Learning Models fo...	<1%

11	Publication	Maynard, Logan. "Advanced Machine Learning and Low-Dimensionality Projectio...	<1%
12	Internet	www.mdpi.com	<1%
13	Publication	Lu, Xiaoyang. "Utilizing Concurrent Data Accesses for Data-Driven and AI Applicat...	<1%
14	Internet	research.cs.wisc.edu	<1%
15	Publication	Linhai Xu, Changsheng Zhang, Yu Liu, Gang Zhao, Shengping Yuan, Wei Guan, Jia...	<1%
16	Internet	arxiv.org	<1%
17	Publication	Henry Munroe, Bright Osatohanmwun, Reza Sharifi. " Multi-objective Evaluation a...	<1%
18	Publication	Thapa, Arjun. "Assessment of Effects of Agricultural Conservation Practices at the...	<1%
19	Publication	Fang Zhao, Ziyi Liang, Qiyang Zhang, Dewen Seng, Xiyuan Chen. "Research on PM2...	<1%
20	Publication	Shipman, William John. "The Extraction of Quantitative Mineralogical Parameters...	<1%
21	Internet	dipot.ulb.ac.be	<1%
22	Publication	Upadhyaya, Priyanka. "Mobile Vision-Based Tools for Yield Estimation in Wine Gra...	<1%
23	Internet	inass.org	<1%
24	Internet	orca.cardiff.ac.uk	<1%

25	Internet	
scholarworks.lib.csusb.edu		<1%
26	Internet	
www.eurecom.fr		<1%
27	Publication	
Xiaohui Wei, Changbao Zhou, Hengshan Yue, Joey Tianyi Zhou. "TC-SEPM: Charact...		<1%
28	Publication	
Yacine Yaddaden, Jerome Parent. "An Efficient Palmprint Authentication System ...		<1%

A Study on Machine Learning Models for Predicting Microarchitectural Performance

Abstract—Performance validation and debugging are essential yet time-consuming stages in microprocessor design, often relying on manual analysis of performance counter data. This work explores the use of machine learning to automate the estimation of microprocessor performance, specifically instructions per cycle (IPC), using execution traces collected from the ChampSim simulator. We evaluate multiple feature engineering strategies and ML models, including Random Forest and Gradient Boosted Decision Trees, to identify effective predictors of performance behavior. Our best-performing models achieve over a 45% reduction in prediction error compared to early-stage baselines. These results demonstrate the potential of machine learning to streamline performance validation workflows, reduce manual debugging effort, and enable future automation in microarchitectural analysis.

Index Terms—Machine Learning, Performance, Random Forest, Microprocessor.

I. INTRODUCTION

The design and validation of microprocessors require meticulous performance analysis to ensure correctness and efficiency across a wide range of workloads and microarchitectural configurations. A crucial part of this process involves detecting and diagnosing performance bottlenecks, often relying on manual inspection of performance counters and trace-level behaviors [1]. However, this manual approach can be both time-consuming and error-prone, especially as architectures and workloads grow in complexity.

Recent advances in machine learning (ML) offer promising avenues to automate aspects of the validation and debugging process. In particular, ML models can be trained to predict performance metrics, such as instructions per cycle (IPC), from rich sets of hardware performance counters. By learning from simulation data across different microarchitectural configurations, these models can support early-stage performance estimation and flag anomalous behavior more effectively.

This work explores the feasibility of using ML to predict microprocessor performance based on simulation traces generated by ChampSim [2], a widely used trace-based microarchitecture simulator. We present a complete workflow that includes simulation data collection, performance counter extraction, feature cleaning and reduction, and the training of several ML models. Two primary data organization strategies are compared: the Unified model, which aggregates training data across all traces, and the Per Traces model, which organizes training data by trace.

We evaluate multiple learning algorithms, including Random Forests [3], Gradient Boosted Decision Trees (GBDT) [4], and Extreme Gradient Boosting (XGBoost) [5], and investigate the impact of dimensionality reduction using Principal Component

Analysis (PCA) [6] and Pearson correlation [7]. Our results show that, for the evaluated dataset, simpler models without dimensionality reduction outperform more complex alternatives, achieving up to 45.3% reduction in prediction error compared to early baseline runs.

These findings highlight the potential of integrating ML-based tools into microprocessor validation workflows, improving debugging efficiency and opening new opportunities for automation in the design and verification cycle.

II. RELATED WORK

Several studies have explored the use of machine learning (ML) techniques for microprocessor performance prediction. However, the objectives, data types, and algorithms used in those works differ in important ways from the approach presented in this study.

Ozisyilmaz *et al.* [8] propose two ML-based methodologies for performance prediction. The first is aimed at design space exploration, training models on a subset of architectural configurations to estimate performance across unseen configurations of the design. The second leverages performance data from existing commercial processors to predict behavior in newer designs. Their work primarily used Convolutional Neural Networks (CNNs) and linear regression models. In contrast, our work focuses on predicting IPC from dynamic performance counters collected via simulation, and evaluates a broader set of tree-based ML algorithms.

Tanaka *et al.* [9] compare the accuracy of two regression techniques: a tree-based non-linear model and a stepwise linear regression approach that selects features to avoid multicollinearity. They conclude that non-linear models better capture microarchitectural performance variations and highlight clock frequency as a key predictor. Unlike their work, which uses static hardware features, our study focuses on dynamic features, such as performance monitoring counters collected during simulation, thus enabling more fine-grained modeling of workload-specific behavior.

Inal and Küçük [10] use offline training with regression models and neural networks to predict future IPC values for running applications. Their work focuses on power efficiency and latency reduction by exploiting historical data from the same processor. In contrast, our approach trains on performance traces from multiple microarchitectures and aims to generalize across designs rather than make predictions within the same system.

Carvajal *et al.* [11] propose a two-stage methodology for performance bug detection. In the first stage, ML models

predict the expected (bug-free) IPC using performance counters; the second stage flags discrepancies between predicted and observed IPC as possible indicators of bugs. Our work shares similarities with their first stage, particularly in using ML to estimate IPC and in applying performance-counter-based feature selection techniques. However, our objective is limited to improving estimation accuracy, without attempting to detect or localize faults.

A follow-up study by the same authors [1] extends their prior work to fault localization using CNNs and Gradient Boosted Decision Trees (GBDT). They find GBDT to outperform CNNs given the dataset size, but suggest CNNs may improve with larger datasets. Their methodology, like ours, relies on performance counters from simulated workloads and multiple microarchitectures. However, their focus is on diagnosing fault location, while our study centers on improving IPC prediction accuracy through feature engineering, model selection, and dataset organization.

In summary, while previous research has demonstrated the utility of ML in microprocessor performance modeling and fault analysis, our work contributes a detailed comparison of feature reduction strategies, dataset structuring techniques (Unified vs. Per Trace), and multiple ML algorithms to improve prediction quality using dynamic simulation data. Unlike prior efforts, our emphasis is not on bug detection or static feature modeling, but rather on optimizing performance estimation from dynamic workload traces in a cross-microarchitecture setting.

III. METHODOLOGY

The methodology adopted in this study is divided into two main stages. The first involves data generation and processing, focusing on how simulation data is collected and prepared for machine learning (ML) model training. The second stage details the training and evaluation procedures used to assess different ML models. Each stage is described in the following subsections.

A. Data Collection and Processing

This stage consists of two primary steps: generating the simulation data and processing it into a format suitable for ML model training.

1) Data Generation

All performance counter data used for training and evaluation was generated using the ChampSim simulator [2], a trace-based microarchitecture simulator that outputs performance metrics for configurable hardware setups. The simulator was used in conjunction with a subset of SPEC CPU 2017 traces made available by the 3rd Data Prefetching Championship.

By default, ChampSim reports aggregated performance counter values at the end of the simulation run. To obtain more granular insights, we modified the simulator to enable periodic reporting of performance counters at regular instruction intervals. These intervals, referred to as phases, divide the simulation into multiple segments, each capturing a snapshot of microarchitectural behavior over a specific instruction window.

This phase-based approach allows us to capture temporal variations in performance, offering a richer and more detailed

dataset for model training. To emulate a variety of systems, we created multiple microarchitectural configurations by varying parameters such as cache size, latency, and associativity for all cache levels, as well as pipeline width, reorder buffer size, instruction queue size, clock frequency, and branch penalties. Each trace was simulated across all configurations, resulting in a comprehensive dataset of performance counter values segmented by phase.

2) Data Processing

The output produced by ChampSim is structured for human readability, rather than machine learning pipelines. Therefore, it must be transformed into a structured tabular format before model training. Each simulation output was parsed to extract numerical values for all counters and reorganized into CSV files, with one row per phase and one column per counter.

To ensure data quality, we applied a filtering process to remove invalid or unusable entries. Specifically, we scanned each counter column to count the occurrences of invalid or undefined values. If the proportion of invalid values for a given counter exceeded a predefined threshold γ , the entire column was removed. Otherwise, only the rows (i.e., phases) containing invalid entries were discarded.

This cleaning step ensures that the dataset used for training is both reliable and representative, avoiding noise that could degrade model performance.

B. Training and Test of Machine Learning models

1) Training and Test Data

For each combination of a microarchitecture and a trace, a separate CSV file containing performance counter data was generated. Formally, let M denote the set of microarchitectural configurations and T the set of traces; then a total of $|M| \times |T|$ simulations were conducted, each producing a distinct data table.

A subset of the microarchitectural configurations in M was selected to form the training set, while the remaining configurations were reserved for the test set. This split enables the evaluation of how well the trained ML models generalize to previously unseen hardware configurations. Importantly, traces used in the test set are not excluded from the training data, only the microarchitecture configurations are disjoint, which allows for controlled variation in microarchitectural design while maintaining trace consistency.

C. Training Strategy

We evaluated two strategies for organizing the training data and training the ML models:

- **Unified Model:** A single model is trained using data from all traces across the microarchitectures in the training set. This model aims to generalize across both traces and hardware configurations, ideally functioning as a general-purpose predictor.
- **Per Trace Model:** A separate model is trained for each trace using its data across all training microarchitectures. This results in $|T|$ models—one per trace. While each model is designed to generalize across microarchitectures, it is tailored to the specific instruction pattern and behavior

of a single trace.

Although the Unified Model offers the advantage of a universal predictor (*one model fits all*), it is expected to perform less accurately than the Per Trace models due to the increased variability and complexity introduced by combining all traces into a single training set.

D. Feature Engineering

To reduce input dimensionality and improve model efficiency, we evaluated two feature selection and reduction techniques:

- **Pruning by Correlation:** This technique proceeds in two stages. First, the Pearson correlation coefficient between each performance counter and the target variable (IPC) is computed. Counters with low correlation ($< \alpha$) are discarded as irrelevant. Then, for the remaining features, pairwise Pearson correlations are computed. If two features are highly correlated ($> \beta$), one is removed to eliminate redundancy.
- **Principal Component Analysis:** PCA is applied to perform linear dimensionality reduction via singular value decomposition. We evaluated the number of principal components retained by varying the percentage of variance explained (e.g., 100%, 95%, 90%, 85%). The impact of scaling the input data before applying PCA was also assessed.

These strategies aim to balance feature relevance and redundancy, reducing model complexity without significantly compromising predictive accuracy.

E. ML Engines

We evaluated three different Machine Learning engines, in order to identify the best performing. The evaluated engines were Random Forest (RF) [3], Gradient Boosted Decision Trees (GBDT) [4] and Extreme Gradient Boosting (XGBoost) [5].

For each algorithm, we explored a range of values for the number of estimators (trees), a key hyperparameter that directly influences model complexity and prediction performance. This exploration allowed us to identify the best-performing model configurations under each training strategy.

IV. EXPERIMENTAL SETUP

A. Simulation Configuration

We selected a set of eight representative traces from the SPEC CPU 2017 benchmark suite, made available for ChampSim as part of the 3rd Data Prefetching Championship (DPC-3). The selected traces are: 400.perlbench-41B, 401.bzip2-226B, 410.bwaves-1963B, 429.mcf-184B, 433.milc-127B, 434.zeusmp-10B, 435.gromacs-111B, and 436.cactusADM-1804B. At the time of experimentation, the ChampSim version used was 2023-08 [2].

We developed 14 microarchitectural configurations for evaluation. Seven of these were based on real-world microarchitectures (Broadwell, Sandy Bridge, Cedarview, AMD Jaguar, AMD Ryzen, Skylake, and Skylake X), while the remaining seven were synthetically generated to diversify the dataset while maintaining realistic hardware parameters. Each configuration was varied in terms of cache size, associativity, latency, pipeline

width, reorder buffer size, and other relevant parameters.

All eight traces were executed on each of the 14 microarchitectures, yielding a total of 112 simulations. Using the modified per-phase data dumping mechanism, each simulation was divided into phases of 250,000 instructions, resulting in 400 phases per simulation (100 million instructions in total per trace).

B. Data Processing

Among the 14 microarchitectures, two (Skylake and Skylake X) were selected as the *test set*. The remaining 12 configurations served as the *training set*. This setup resulted in 38,400 training samples and 6,400 testing samples. While the Unified model was trained on the full 38,400 samples, each Per Trace model used only the 4,800 training samples corresponding to that specific trace.

During preprocessing, we applied a data cleaning step to remove invalid or missing values. If a performance counter had more than 5% invalid entries ($\gamma = 0.05$), the entire counter was removed. After pruning such columns, any phase (row) with at least one remaining invalid entry was also discarded.

C. Feature Engineering Evaluation

1) Correlation-Based Pruning

Feature selection using correlation followed the two-step methodology described earlier. We set the relevance threshold to $\alpha = 0.3$ and the redundancy threshold to $\beta = 0.8$. That is, counters with Pearson correlation coefficient less than 30% with respect to IPC were considered irrelevant and removed. Among the remaining counters, those with pairwise correlation above 80% were deemed redundant, and one from each highly correlated pair was eliminated.

2) Principal Component Analysis

We also explored dimensionality reduction using Principal Component Analysis (PCA). The number of retained components was adjusted based on the amount of variance preserved: 100%, 95%, 90%, and 85%. Each scenario was tested both with and without feature scaling prior to applying PCA, in order to evaluate the impact of normalization on the transformation.

D. ML Engine Evaluation

Our preliminary results showed that the best performance was obtained using the original feature set (after data cleaning), without additional pruning via correlation or PCA. Therefore, for the ML engine comparison, we trained models using the full set of valid counters.

We evaluated three learning algorithms: Random Forest (RF) [3], Gradient Boosted Decision Trees (GBDT) [4], and Extreme Gradient Boosting (XGBoost) [5]. Each algorithm was tested with multiple values of the number of estimators: 50, 100, 200, 300, 500, and 1000. The goal was to identify the best-performing combination of model and hyperparameters for IPC prediction.

All the evaluations were performed using Python, and the ML Engine implementation was performed using the Scikit-Learn Library [12].

TABLE I: Prediction error (RMSE) in the Per-Trace model for different feature engineering techniques

Traces	Baseline	Pearson Pruning	100% explained variance		95% explained variance		85% explained variance	
			PCA	PCA+Scaling	PCA	PCA+Scaling	PCA	PCA+Scaling
400.perlbenc-41B	0.0457	0.0565	0.108	0.0998	0.0942	0.1023	0.1194	0.0947
401.bzip2-226B	0.0903	0.1215	0.166	0.1982	0.2002	0.1933	0.5787	0.1908
410.bwaves-1963B	0.0567	0.0682	0.0575	0.0619	0.0762	0.0706	0.0845	0.071
429.mcf-184B	0.0384	0.0548	0.0145	0.0394	0.0147	0.0355	0.0227	0.0559
433.milc-127B	0.0296	0.0432	0.0342	0.0366	0.0764	0.0531	0.1268	0.055
434.zeusmp-10B	0.121	0.1172	0.1513	0.5493	0.1302	0.5502	0.1329	0.6143
435.gromacs-111B	0.1224	0.1372	0.5502	0.29	0.2846	0.6172	0.2613	0.6026
436.cactusADM-1804B	0.0419	0.1111	0.0587	0.0616	0.1171	0.0678	0.1459	0.0631
Geometric Mean	0.0599	0.0817	0.0841	0.1071	0.0956	0.1253	0.1307	0.1320

V. RESULTS

A. Comparison of Feature Engineering Techniques

To ensure a fair comparison among feature engineering methodologies, all experiments in this section were conducted using the same machine learning model configuration. Specifically, a Random Forest (RF) model with 100 estimators was used across all tests. Evaluation was performed on the *test dataset*, and results are reported using Root Mean Squared Error (RMSE).

Table I presents the RMSE for the different feature engineering techniques under the Per Trace model strategy. The geometric mean is computed across all evaluated traces. The column labeled “Baseline” corresponds to the scenario in which no feature pruning is applied. For each trace, the lowest RMSE is highlighted in bold.

While dimensionality reduction techniques such as correlation-based pruning and Principal Component Analysis (PCA) are commonly effective in high-dimensional datasets, they did not yield improved results in this case. The number of features (i.e., performance counters) ranged from approximately 70 to 112, which is relatively low compared to the number of training samples. As a result, the “Baseline” model (which retains all valid counters) consistently achieved the lowest RMSE across most traces. These results suggest that preserving the full set of performance counters was beneficial, likely because each counter contributed useful information that might have been lost during feature pruning.

A similar trend was observed for the Unified model, where the Baseline configuration also resulted in the lowest prediction errors. Figure 1 compares the RMSE across traces between the Unified and Per Trace training strategies. While overall performance is similar, some traces (such as 434.zeusmp-10B) show significant performance differences between strategies, with the Unified model achieving lower error in that particular case.

B. Comparison of Machine Learning Engines

As described in the Experimental Setup, we evaluated different combinations of Random Forest (RF), Gradient Boosted Decision Trees (GBDT), and XGBoost (XGB) algorithms, varying the number of estimators from 50 to 1000. Table II summarizes the best-performing configuration for each algorithm under both training strategies.

For the Unified model, the best result was obtained using Random Forest with 200 estimators. For the Per Trace model,

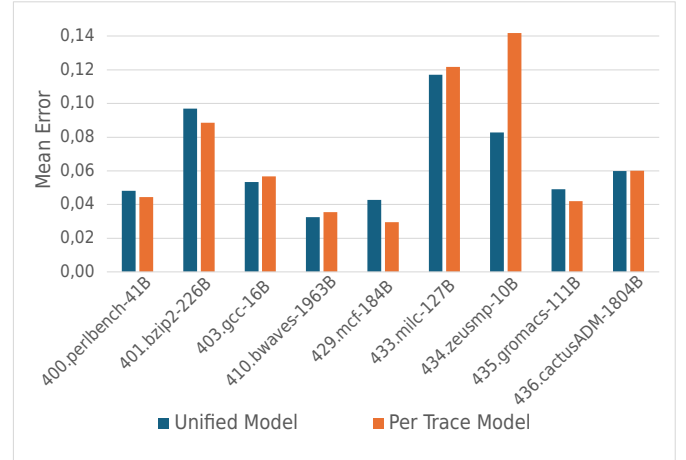


Fig. 1: RMSE comparison between the training strategies

the best result came from GBDT with 1000 estimators, which also achieved the lowest overall RMSE among all configurations evaluated. However, this does not imply that the same model performs best across all traces. For example, trace 433.milc-127B achieved its lowest RMSE (0.0286) using Random Forest with 200 estimators under the Per Trace model, demonstrating that optimal configurations can vary at the individual trace level.

To further illustrate model behavior beyond a single RMSE value, we examined prediction plots comparing the real and predicted IPC values across all simulation phases. These plots help reveal patterns or localized anomalies that may be hidden in the average error metric.

Figure 2 shows the prediction plot for trace 400.perlbenc-41B using the Per Trace model with GBDT and 1000 estimators. The x-axis represents simulation phases, and the y-axis indicates IPC values. The first 400 phases correspond to predictions for the Skylake architecture, while the remaining 400 phases represent Skylake X. Despite abrupt changes in IPC, the model successfully tracks the general behavior across both microarchitectures.

Figure 3 shows the results for 433.milc-127B, where predictions for Skylake align well with the actual IPC, but predictions for Skylake X show a systematic offset. Even though the absolute values deviate, the model still captures the underlying trend, suggesting the presence of minor calibration issues rather than fundamental prediction failures.

These qualitative observations align with the quantitative

TABLE II: Prediction error (RMSE) for different ML engines

Traces	Unified Model Strategy			Per Trace Strategy		
	RF estimators=200	GBDT estimators=1000	XGBoost estimators=1000	RF estimators=200	GBDT estimators=1000	XGB estimators=1000
400.perlbenc-41B	0.0473	0.0634	0.0606	0.0456	0.0448	0.0455
401.bzip2-226B	0.0948	0.1133	0.1247	0.088	0.1108	0.1126
410.bwaves-1963B	0.0536	0.0634	0.0631	0.0159	0.0167	0.0177
429.mcf-184B	0.0318	0.0453	0.0578	0.0357	0.0365	0.0342
433.milc-127B	0.0431	0.0518	0.0524	0.0286	0.0262	0.0261
434.zeusmp-10B	0.1147	0.1093	0.1122	0.0666	0.0515	0.0487
435.gromacs-111B	0.0813	0.1195	0.1986	0.4479	0.3254	0.3671
436.cactusADM-1804B	0.0482	0.0451	0.046	0.0415	0.0331	0.0345
Geometric Mean	0.0591	0.0708	0.07876	0.0547	0.0507	0.0516

findings, reinforcing the importance of selecting appropriate training strategies and models based on the specific characteristics of the workload and target architecture.

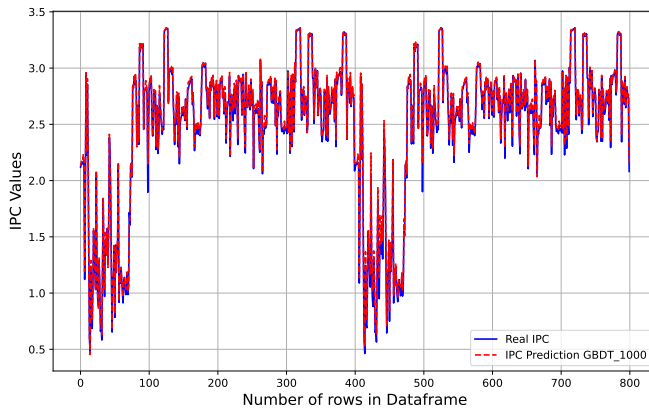


Fig. 2: Comparison between Real IPC and IPC Prediction for 400.perlbenc-41B using Per Traces model

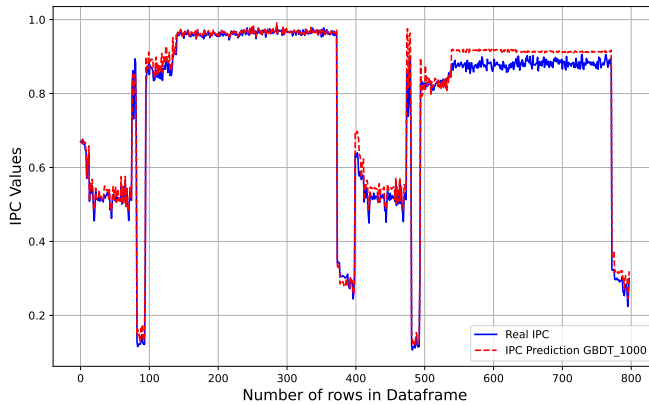


Fig. 3: Comparison between Real IPC and IPC Prediction for 433.milc-127B using Per Traces model

To quantify the overall improvement achieved throughout this study, we compared the final results against those obtained during the initial experimentation phase. Table III summarizes this comparison, highlighting the error reduction between the baseline experiment and the best-performing configuration for each training strategy.

TABLE III: Error reduction of best performing vs initial experiment

Traces	Unified Model Strategy	Per Trace Model Strategy
400.perlbenc-41B	70.44%	7.89%
401.bzip2-226B	23.92%	-4.74%
410.bwaves-1963B	51.80%	94.02%
429.mcf-184B	59.64%	56.32%
433.milc-127B	5.90%	53.06%
434.zeusmp-10B	54.97%	29.42%
435.gromacs-111B	48.51%	-357.16%
436.cactusADM-1804B	16.75%	26.60%
Geometric Mean	45.36%	36.03%

As shown in Table III, the geometric mean error for the Unified model was reduced by 45.3%, while the Per Trace model saw an improvement of 36%. Although these improvements are substantial, it is worth noting that a few individual traces, such as 435.gromacs-111B, experienced a noticeable degradation in prediction accuracy. This suggests that while the overall methodology is effective, certain workloads may require specialized handling or further tuning to achieve consistent performance across all scenarios.

VI. CONCLUSION

This study explored the use of machine learning to predict microarchitectural performance, specifically IPC, from performance counter data generated using the ChampSim simulator. Two training strategies were evaluated: the Unified model and the Per Trace model. The best result for the Unified model achieved a 45.3% reduction in prediction error using Random Forest with 200 estimators, while the Per Trace model reached a 36% error reduction using Gradient Boosted Decision Trees (GBDT) with 1000 estimators. Among both strategies, the Per Trace model with GBDT achieved the lowest overall geometric mean RMSE of 0.0507.

These results suggest that tree-based models, particularly when properly tuned, can offer robust performance for this type of prediction task. Future work could explore alternative learning algorithms, more sophisticated hyperparameter tuning, or ensemble strategies to further reduce prediction error.

An important observation during this study was the value of prediction plots as a complementary evaluation tool. While RMSE provides a compact metric, it can obscure localized prediction failures. Visualizing predicted IPC values across simulation phases helped identify subtle model behaviors, such as systematic offsets or inconsistent prediction accuracy across different microarchitectures. Despite these limitations, predictions that successfully track IPC trends (even with minor

offsets) can still be valuable, especially for downstream tasks like performance anomaly detection or debugging.

Regarding feature reduction techniques, neither Pearson correlation-based pruning nor Principal Component Analysis (PCA) improved prediction accuracy. In the case of Pearson pruning, the thresholds used in this study were relatively permissive, and results suggest that ChampSim counters may already exhibit low inter-correlation. For PCA, the lack of improvement may stem from suboptimal parameter selection, such as the explained variance thresholds or the absence of task-specific feature scaling.

Overall, this study demonstrates the feasibility and potential of ML-based IPC prediction using simulation-derived performance counters, while also outlining clear directions for improving model accuracy and generalization in future research.

REFERENCES

- [1] E. Carvajal Barboza, M. Ketkar, P. Gratz, and J. Hu, "Aiding microprocessor performance validation with machine learning," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2024, pp. 1–9.
- [2] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," Oct 2022. [Online]. Available: <https://arxiv.org/abs/2210.14324>
- [3] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.
- [4] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.
- [5] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.
- [6] K. Pearson, "On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, vol. 2, no. 11, pp. 559–572, 1901.
- [7] —, "Note on regression and inheritance in the case of two parents," *Proceedings of the Royal Society of London*, vol. 58, no. 347-352, pp. 240–242, 1895.
- [8] B. Ozisikyilmaz, G. Memik, and A. Choudhary, "Machine learning models to predict performance of computer system design alternatives," in *International Conference on Parallel Processing*. IEEE, 2008, pp. 495–502.
- [9] Y. Tanaka, K. Oka, T. Ono, and K. Inoue, "Accuracy analysis of machine learning-based performance modeling for microprocessors," in *International Japan-Egypt Conference on Electronics, Communications and Computers (JEC-ECC)*. IEEE, 2016, pp. 83–86.
- [10] G. Inal and G. Küçük, "Application of machine learning techniques on prediction of future processor performance," in *International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, 2018, pp. 190–195.
- [11] E. Carvajal Barboza, S. Jacob, M. Ketkar, M. Kishinevsky, P. Gratz, and J. Hu, "Automatic microprocessor performance bug detection," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 545–556.
- [12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.