Predictive Modeling and Interpretability Techniques for SoC Performance with Machine Learning

Ghazi Ben Henia
LIRMM, Univ. Montpellier
Montpellier, France
Email: ghazi.benhenia@lirmm.fr

Abdoulaye Gamatié
LIRMM, Univ. Montpellier - CNRS
Montpellier, France
Email: Abdoulaye.Gamatie@lirmm.fr

***Abstract*—Accurate prediction of System-on-Chip performance is essential for informed decision-making in hardware design and deployment. This paper introduces a machine learning-based approach to predict SoC performance using the SPEC 2017 benchmark results. Our primary objective is to develop a high-accuracy, compact feedforward machine learning model for performance prediction. In contrast to existing solutions facing challenges related to size, computational power and latency, our model achieves a harmonious balance, delivering high accuracy while maintaining compactness and low latency, showcasing impressive R^2 scores of 0.98. The utilization of machine learning offers a valuable alternative to traditional benchmarking approaches, enabling predictions for unseen hardware without relying on representative benchmark workloads. We specifically explored the use of autoencoders in analyzing and understanding the decision-making process for fully-connected network models. Additionally, we applied feature selection and SHAP to enhance transparency and confirm the results provided by the autoencoder.**

I. Introduction

Performance benchmarks play a crucial role in gaining insights into system behavior, informing procurement decisions, and guiding deployment and scaling choices during system operations. These benchmarks are designed to evaluate the expected performance of a user's workload and energy consumption. Despite previous efforts employing different networks[3], such as a fully-connected network and Convolutional Neural Networks (bespoke and ResNet inspired), yielding impressive R^2 scores of 0.96, 0.98, and 0.94, these models face problems related to size, computational power, and latency. Recognizing the challenges posed by these limitations, our research seeks to address the issue of evaluating the performance of previously unseen hardware-workload combinations, using the SPEC CPU 2017 dataset, encountered by previous models. Our proposed model strives to strike a harmonious balance by achieving high accuracy while maintaining compactness and low latency. Additionally, we incorporate the Multi-Layer Perceptron (MLP) model designed for prediction into an autoencoder [5]. This method offers the advantage of examining the model's output, providing insights into the impact of each input feature on the prediction. Firstly, we evaluate the architecture of the MLP model. Secondly, we undergo a hyperparameter tuning process within its network. After that, we use its structure to build the decoder part of the autoencoder. Lastly, we apply SHAP [4] and feature selection [7] on different models, including RandomForestRegressor (RFR), and compare the results provided by the three techniques. The structure of the remainder of this paper is organized as follows: In Section 2, we conduct a comprehensive review of previous work with a particular focus on performance prediction. Section 3 provides a detailed overview of our methodology, while Section 4 presents the results of our study. Concerns and challenges are analyzed in Section 5, and we conclude by outlining potential avenues for future research in Section 6.

II. Related Work

Several works have addressed the problem of predicting metrics for the SPEC datasets. However, the exploration of the SPEC 2017 dataset has been limited, with only three prior works focusing on it. In previous work [3], researchers conducted a comprehensive exploration to find the optimal neural network architecture for the SPEC data. They considered three types of network designs: fully-connected networks, convolutional neural networks (CNN), and networks using Residual blocks (inspired by ResNet architecture). For fully-connected networks, they evaluated trapezium, reverse trapezium, and rectangular structures, varying the number of neurons in each layer. In the case of CNN, 1D convolutional layers with trapezium-shaped structures were employed. For Residual design, they utilized Residual blocks and introduced a convolutional block to address data shape differences. These blocks were combined into superblocks, forming the basis for constructing the overall network. Baseline models such as Linear Regression and Support Vector Regression were initially evaluated, with CNN-based approaches demonstrating superior performance. Trapezium networks within MLP outperformed other MLP networks, securing high positions in terms of R^2 and MSE. However, CNN networks dominated the top positions for both metrics. Surprisingly, residual-inspired approaches did not perform well, ranking lower in comparison to CNN networks. The researchers noted that these models, while more complex and time-consuming to train, provided little benefit in their specific case.

Other studies[5] have extensively examined autoencoders, providing insights into their mathematical underpinnings, limitations, and typical applications, primarily in 2-D datasets.

These investigations delved into critical aspects such as the role of activation functions in the output layer, the selection of appropriate loss functions, and the interpretation of reconstruction errors. Moreover, [2] introduced a novel approach leveraging autoencoders for the joint exploration of latent and real spaces, thus advancing the field of eXplainable Artificial Intelligence (XAI). By revealing the intricate relationships between input and latent features, this methodology offers valuable insights into data structures, including their relevance to external variables like classification model predictions.

Despite the demonstrated potential of autoencoders and eXplainable Artificial Intelligence (XAI) techniques, their utilization in regression and supervised learning tasks remains largely unexplored within the context of System-on-Chip (SoC) systems. Recognizing the capacity of these methodologies to augment the efficiency and accuracy of performance prediction models for SoC architectures, further exploration and investigation in this domain are warranted.

III. Methodology

This experiment was conducted on Google Colab (Intel Xeon CPU running at 2.30 GHz with 13 GB of memory.)

A. Dataset Processing

In this study, we explore the application of the SPEC 2017 benchmark dataset for machine learning, which has been previously processed in an earlier work[3]. The dataset consists of 34 attributes, with the numeric columns "Peak Result" and "Base Result" representing system response times under load or no load, respectively—these values are the focus of prediction in our investigation. We use the dataset that was initially prepared[3], Their preprocessing strategy aimed to optimize its suitability for model training. Their approach included alphanumeric cleaning to remove non-alphanumeric characters, elimination of spaces in column names, and conversion of all characters to lowercase for consistency. They also addressed outliers, particularly zero values in "Base Result," by identifying and removing them. The units across the dataset were standardized to MB for consistency (e.g., memory in KB, MB, GB). To handle columns that appeared arbitrary but had actual constraints, such as memory, they transformed them into categorical labels. Additionally, highly correlated columns, including 'CPU(s) Orderable,' 'Energy Base Result,' 'License,' 'Parallel,' 'System,' 'Test Sponsor,' and 'Tested By,' were identified using Kendall's rank correlation[6]. It assesses the degree of agreement in ranking between two sets of data to measure their monotonic association. These columns, found to be over 0.7 correlation score with others, were eliminated to reduce redundancy in prediction abilities.

B. Exploring Optimal Neural Network Structures

The configuration of layers and neurons significantly influences the performance and size of Deep Learning networks. In our quest for the best network architecture, both for the autoencoder and prediction model, we focus on investigating the trapezium network structure. We populate this structure with neurons, with a particular emphasis on fully-connected networks as a foundational design.

1) Prediction Model: The Prediction Model is a trapezium network[3], the first layer contains 2n neurons, and each subsequent layer has half the number of neurons as the previous one. The penultimate layer features 2n-m neurons. We vary the values of n in the range [6, ..., 11] and m in the range [1, ..., 10]. Given that our task involves regression, the output layer consists of just one neuron with a linear activation function.

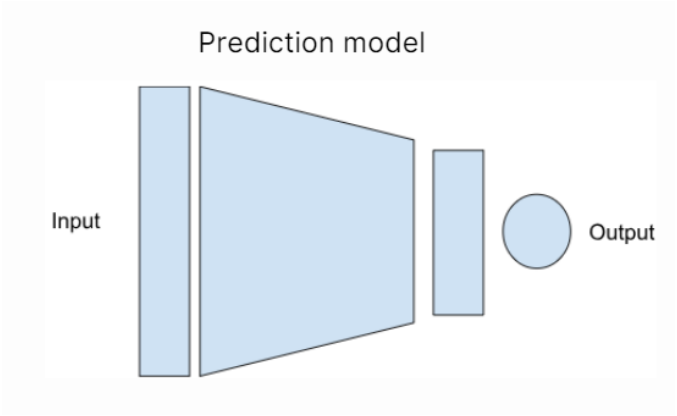

Fig. 1. Prediction model structure

2) Feed Forward Autoencoder: An autoencoder is a tool crafted to glean an insightful representation of data by proficiently reconstructing input observations. It has some typical applications as dimensionality reduction, classification, denoising, and anomaly detection. It comprises three essential components: an encoder, a latent feature representation, and a decoder. The Feed-Forward Autoencoder (FFA)[5] is constructed using dense layers with a distinctive structure. The common FFA architecture entails a reduction in the number of neurons as we progress through the network until reaching the midpoint. From there, the neuron count starts growing again until the last layer matches the input dimensions. The layer with fewer neurons often termed the bottleneck, is a critical aspect of this architecture (trapezium network + bottleneck + a reverse trapezium). The prediction model will play the role of the encoder (or the trapezium network) by just eliminating the output layer. Reverse trapezium networks invert the order of layers (excluding the last one), with the narrowest layer first and the widest layer last. The bottleneck layer (the last layer of the trapezium network) has l neurons and we vary the values of l in the range [3, ..., 16].

C. Hyperparameter Search

In addition to executing a neural architecture search over the specified structure, we thoroughly explored hyperparameters applicable to the networks, covering the optimizer, the number of training epochs, the loss function, and the activation function.

1) Optimizer: The optimizer dictates how the network weights are updated after each training step. This investigation

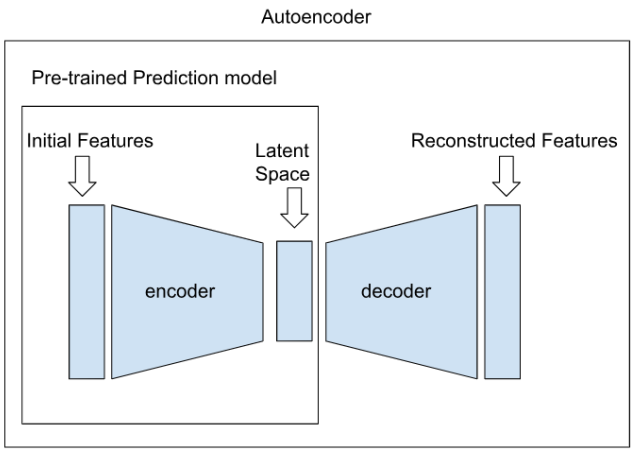

Fig. 2. Autoencoder structure

concentrates on three widely used optimizers: RMSprop[3] (Root Mean Squared Propagation): Applying decaying average partial gradients to the step size of each parameter, RMSprop prioritizes more recent gradients. Adam[3] (ADAptive Moment Estimation): Extending SGD, Adam employs a distinct learning rate for each parameter. Unlike RMSprop, Adam considers the average of the second moment when adapting learning rates.

2) Loss Function: The loss function measures the difference between predicted values and true values. We assess the Mean Absolute Error (MAE)[1], denoted as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}(y_i' - y_i)^2[1]$$

where N is the number of samples, and y_i' and y_i are the predicted and true values, respectively.

3) Activation Function: The activation function[3] introduces a non-linear element within the networks. We evaluate three commonly used activation functions: sigmoid, tanh, and ReLU. Although ReLU has demonstrated effectiveness in many problems, other activation functions may be more suitable in certain cases. As per convention, no activation function is applied to the final output layer of the prediction model, allowing for arbitrary output values.

D. SHapley Additive exPlanations (SHAP)

As machine learning models evolve to become more intricate and powerful, delivering accurate predictions, they concurrently transition into "black boxes," making it increasingly challenging to comprehend the rationale behind their predictions. This phenomenon has prompted a heightened emphasis on the interpretability and explainability of machine learning models. SHAP[4] (SHapley Additive exPlanations) values offer a method for elucidating the output of any machine learning model. Employing a game-theoretic approach, SHAP measures each feature's contribution to the outcome. In machine learning, each feature is allocated an importance value, signifying its contribution to the model's output. SHAP values provide insights into how each feature influences each final prediction, the relative significance of each feature compared to others, and the extent to which the model relies on the interaction between features. However, it's essential to acknowledge that the downside of SHAP lies in the significant computing time it demands. The Shapley value's computational requirements grow exponentially with the number of features, posing a potential limitation.

E. Feature Selection

Feature selection[7] is widely employed in machine learning and data processing, serving to identify a subset of relevant variables within high-dimensional data spaces. It is a crucial step in the development of predictive models, aiming to reduce the number of input variables for various reasons. Primarily, it seeks to diminish the computational cost associated with modeling. Additionally, in certain cases, it can enhance the model's overall performance. Statistical-based feature selection methods play a key role in this process, involving the evaluation of the relationship between each input variable and the target variable using statistical measures. The goal is to select those input variables that exhibit the strongest relationship with the target variable. These methods are known for their speed and effectiveness, although the choice of statistical measures depends on the data types of both the input and output variables.

F. Implementation Details

We implement a 70-15-15 split for training, testing, and validation, respectively, with a fixed batch size of 10. The intentional design of the data splitting process aims to showcase the model's performance on real-world, unseen future data. Using a controlled split approach like cross-validation to adjust class distributions would contradict the goal of simulating an uncontrolled future data configuration. To prevent exploding or vanishing gradients, we maintain constant variance. The number of epochs is allowed to range from 50 to 300 in steps of 50, with training ceasing after 300 epochs, where models exhibit optimal performance on the test set.

Evaluation Metrics: We assess model performance using Mean Squared Error (MSE)[1], Mean Absolute Error (MAE)[1], and R^2[1]:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where y_i is the true value, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of all true values. For both R^2 and MSE, values farther from their predicted counterparts exert a more pronounced impact on the results. To gauge the models' susceptibility to outliers, prioritizing MSE is advisable. Opting for R^2 would diminish the influence of outliers at the expense of overall accuracy.

We initially train our prediction model on two datasets, starting with the processed dataset. Upon achieving the desired performance, we proceed to construct the remainder of the

autoencoder and continue training it using the same dataset. The autoencoder's training is halted upon observing optimal performance.

The subsequent phase involves an analysis of the reconstructed features. We compute the Mean Absolute Error (MAE) for each input feature, ordering them from highest to lowest MAE. Additionally, we calculate the correlation value for each input feature and its reconstruction. Ultimately, we apply SHAP[4] and feature selection[7], including RandomForestRegressor (RFR), and compare the results provided by the three techniques.

Furthermore, we conduct a comparative analysis between the results obtained from the autoencoder, SHAP plots, and the feature selection technique applied to the RandomForestRegressor model.

IV. Results

We present the results of our model training in comparison to the performance results obtained in previous studies. Table I provides a comprehensive overview of the performance metrics, size, number of parameters, and latency (for 1000 inputs), for all the models.

A. Our Model

For our MLP network (Figure 1), the Trapezium network exhibited high performance, particularly in terms of the R^2 score. While MLP networks are generally not competitive with CNN-based approaches, it's worth noting that the CNN is approximately 19 times larger in size and 3.5 times slower than the MLP. This tradeoff may be significant when evaluating the overall efficiency of the models. To summarise the design choice of the prediction model:

- **Optimizer:** Consistent with prior research findings, Adam consistently demonstrated superior performance.
- **Loss function:** In all scenarios, Mean Absolute Error (MAE) consistently produced the most favorable results.
- **Activation Function:** Relu outperformed others in both MAE and R^2 metrics. Sigmoid and the tanh activation functions failed to surpass 0.01 in terms of R^2. The use of a linear activation function in the layer just before the output aids in the reconstruction with the autoencoder, considering that this layer serves as the bottleneck.
- **Training Epochs:** Figure 3 illustrates the impact of training epochs on our model's performance, measured by MAE. The epoch count for stopping our training was determined empirically.

B. TriMLP

In reference [3], they examined a trapezium network, depicted in Figure 4, along with its reverse counterpart – referred to as a reverse trapezium – and a rectangular network with an equal number of neurons in each layer. In the trapezium network, the initial layer comprises 2^9 neurons, with subsequent layers halving the number of neurons compared to the previous layer. The penultimate layer contains 2^{n-m} neurons, where the difference between n and m is greater than 1. The

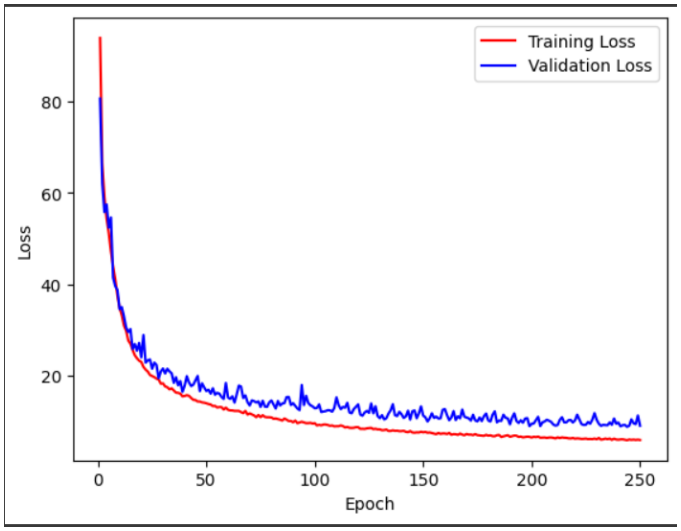

Fig. 3. Impact of training epochs on the model's performance

values of n range from 4 to 11, and m ranges from 1 to 10. The final layer of the network consists of a single neuron to yield the regression outcome.

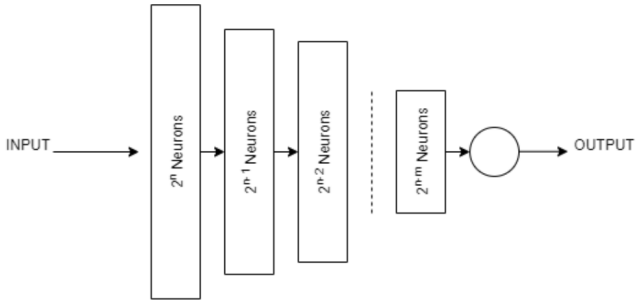

Fig. 4. Impact of training epochs on the model's performance

C. TriCNN

The CNN network[3] is composed of several convolutional layers, followed by a set of fully-connected layers. Figure 5 illustrates the structure of this network. It's important to mention that the fully-connected layers are smaller in these cases compared to when only fully-connected layers are used. Since our data is tabular, we utilize 1D convolutional layers, meaning our kernels/filters are 1D and have a size of $k = 3$. Once again, we adopt the trapezium format, where the first convolutional layer has a width (number of filters) of 2^n, and each subsequent layer has half the width of the previous layer, with the last convolutional layer having a width of 2^2. The fully-connected layers also follow a trapezium shape, with the number of nodes per layer ranging between 2^9 and 2^4.

D. Residual Model

We use the same structure as [3]. They adopt Residual blocks, incorporating a 'bypass' link around a set of convolu-

TABLE I
OVERVIEW OF THE PERFORMANCE METRICS, SIZE, NUMBER OF PARAMETERS, AND LATENCY (FOR 1000 INPUTS), FOR ALL THE MODELS.

Architecture	Total params	Size (KB)	Latency	R2	MAE
Our model	89741	350.55	0.1841 s	0.978	8.7
TriCNN	1685121	6430	0.6172 s	0.98638	5.6738
TriMLP	2801089	10690	0.4264 s	0.97347	9.1244
Residual	16334465	62310	19.8323 s	0.95007	10.5950

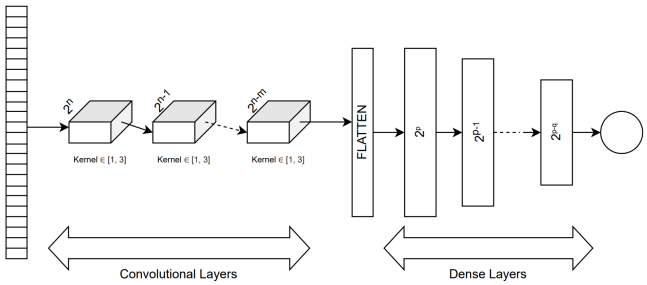

Fig. 5. The CNN network structure

tional units merged with their output. One restriction is that the shape of data entering the block must match the output for successful merging. To address this, they introduced a convolution unit to the 'bypass' path, aligning its output width with the final convolution in the main path. This configuration called a convolutional block, features convolutions with widths of 2^{p-2} for the main path and widths of 2^p for both paths. The two block templates are combined to produce a superblock, starting with a convolutional block followed by r identity blocks, each with an output width of 2^p. Superblocks can be concatenated together (Figure 6), with each subsequent one doubling the output width following the ResNet convention. The output from the last superblock is flattened before being fed into a single neuron for regression prediction.

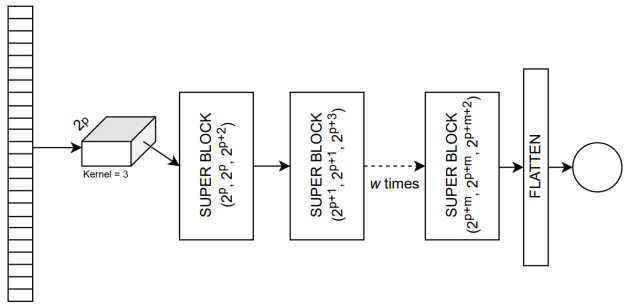

Fig. 6. The ultimate design of the ResNet model

E. Autoencoder

The prediction model will play the role of the encoder by just eliminating the output layer. For the decoder, we invert the order of layers (excluding the last one), with the narrowest layer first. Normally we will use the same number of neurons and activation functions of the prediction model. First, we fix the weights of the encoder part then we train the autoencoder to reconstruct the input data. Figure 2 presents the full structure of the autoencoder.

F. Applying SHAP Method

The beeswarm plot (Figure 7) is designed to present a concise and information-rich summary of how the top features in a dataset influence the model's output. Each instance of the given explanation is depicted by a singular dot on each feature row. The x position of the dot is determined by the SHAP value (shap_values.value[instance,feature]) of that feature. Dots accumulate along each feature row, illustrating density, and color is utilized to represent the original value of a feature. By default, features are ordered using shap_values.abs.mean(0), which corresponds to the mean absolute value of the SHAP values for each feature. This default order places greater emphasis on broad average impact and less on rare but high-magnitude impacts.

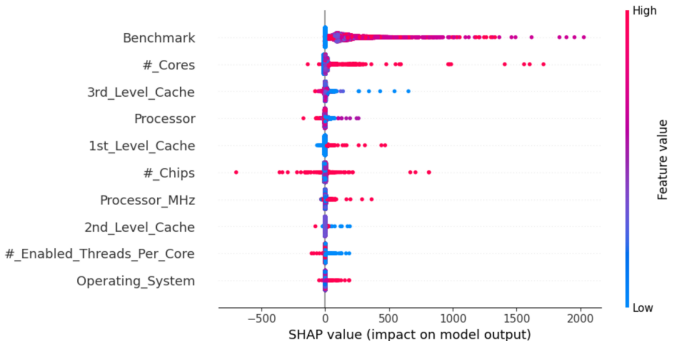

Fig. 7. Beeswarm plot

G. Applying Feature Selection Techniques on RFR Model

Certain feature selection techniques cannot be effectively applied to neural networks. In our study, we employed Sequential Feature Selection (SFS)[7], a forward feature selection method, in conjunction with a RandomForestRegressor model.

SFS systematically added features to the model in iterations, aiming to identify the most pertinent ones. Moreover, we utilized Recursive Feature Elimination (RFE)[7], a backward feature elimination approach, with the same RandomForestRegressor model. RFE systematically pruned less influential features to refine the model's feature set and enhance its performance.

H. Evaluation of the 3 Methods

In our evaluation, we divided the analysis into two distinct steps. Initially, we applied feature selection techniques to a random forest regressor model to gain preliminary insight into the dataset and to identify the most relevant features. Table II provides the selected subsets of the most informative features for the 2 techniques[7]. Upon comparing these results with the broader dataset, we found it both convincing and unsurprising that features such as "Benchmark", "#_Cores," and "#_Chips" held greater significance compared to others like "Published," "Test_Date," and "Updated."

TABLE II
THE SELECTED SUBSETS OF THE MOST INFORMATIVE FEATURES FOR THE 2 TECHNIQUES

SFS	RFE
Benchmark	Benchmark
#_Cores	#_Cores
#_Chips	#_Chips
Processor	Processor
2nd_Level_Cache	2nd_Level_Cache
3rd_Level_Cache	3rd_Level_Cache
Energy_Peak_Result	Processor_MHz
Hardware_Vendor	Memory
1st_Level_Cache	File_System

In the subsequent phase, as both the autoencoder method and SHAP are employed on the same model, we proceed to compare their outcomes. To achieve this, we iteratively reconstruct the input features 20 times and then compute the presence rate of each feature within the top 10 best-reconstructed features. This evaluation is based on correlation scores and mean absolute error (MAE). Tables III and IV compare the selected subsets of the best-reconstructed features with those obtained from the SHAP plot. Features that have a 100% presence rate and are identified by SHAP are highlighted in dark grey, while features identified by both SHAP and feature selection but with a presence rate less than 100% are shown in a lighter color. Remarkably, we observe that 6 out of the 9 features identified by SHAP exhibit a 100% presence rate among the top 10 best-reconstructed features, while the remaining features demonstrate a presence rate ranging between 70% and 90% across the 20 test runs. This comparison highlights a notable finding: reconstructing the input from a pre-trained model yields an accuracy of approximately 70% compared to SHAP, while significantly reducing the computational time required.

TABLE III
AUTOENCODER VS SHAP

Features	AE	SHAP
Benchmark	100%	X
3rd_Level_Cache	100%	X
Processor	100%	X
2nd_Level_Cache	100%	X
Processor_MHz	100%	X
#_Cores	100%	X
#_Chips	90%	X
1st_Level_Cache	85%	X
#_Enabled_Threads_Per_Core	68%	X
Energy_Peak_Result	90%	

TABLE IV
AUTOENCODER VS FEATURE SELECTION

Features	AE	FS
Benchmark	100%	X
3rd_Level_Cache	100%	X
Processor	100%	X
2nd_Level_Cache	100%	X
Processor_MHz	100%	X
#_Cores	100%	X
#_Chips	90%	X
Memory	68%	X
File_System	90%	
1st_Level_Cache	85%	

V. LIMITATIONS

This study focuses primarily on predicting performance results using a specific dataset, SPEC CPU 2017 retrieved on 10 September 2022. However, several limitations warrant acknowledgment. Firstly, the analysis primarily pertains to the prediction of performance results, and further investigation is necessary to assess whether the predictive performance extends to other columns "Energy Peak Result" and "Energy Base Result" within the dataset. Secondly, while the current study employed a particular set of data and techniques for feature selection and SHAP analysis, it remains uncertain whether similar results would be obtained with different datasets or if the input data were reconstructed differently. Evaluating the generalizability of the findings across diverse datasets and alternative methodologies for input data reconstruction is a pertinent avenue for future research.

VI. CONCLUSION

We have explored autoencoders and their application in offering interpretability for machine learning models. While achieving model accuracy is undoubtedly crucial, companies must extend their focus beyond accuracy alone. Emphasizing interpretability and transparency becomes essential for gaining the trust of users and satisfying regulatory requirements. The ability to articulate the rationale behind a model's specific prediction serves as a powerful tool for debugging potential biases, identifying data-related issues, and providing justifications for the model's decisions. Although SHAP and feature selection techniques boast high accuracy, they are not without limitations. Certain feature selection methods are not directly applicable to neural networks and require alternative approaches. Additionally, while SHAP is accurate, its computational intensity and time-consuming nature are notable constraints. Autoencoders, on the other hand, present a

promising alternative, offering results close to those provided by SHAP values but in a fraction of the time.

REFERENCES

[1] Olayemi Muyideen ADESANYA and Solomon Onen ABAM. Data analytics evaluation metrics essentials: Measuring model performance in classification and regression ismail olaniyi muraina department of computer science, college of information and technology education, lagos state university of education, lagos nigeria.

[2] Francesco Bodria, Salvatore Rinzivillo, Daniele Fadda, Riccardo Guidotti, Fosca Giannotti, and Dino Pedreschi. Explaining black box with visual exploration of latent space. *EuroVis–Short Papers*, 2022.

[3] Mehmet Cengiz, Matthew Forshaw, Amir Atapour-Abarghouei, and Andrew Stephen McGough. Predicting the performance of a computing system with deep networks. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, pages 91–98, 2023.

[4] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[5] Umberto Michelucci. An introduction to autoencoders. *arXiv preprint arXiv:2201.03898*, 2022.

[6] David Sarmento. Chapter 22: Correlation types and when to use them, 2022.

[7] B Venkatesh and J Anuradha. A review of feature selection and its methods. *Cybernetics and information technologies*, 19(1):3–26, 2019.