Compiler Optimization Prediction with New Self-Improved Optimization Model

Chaitali Shewale[1], Sagar B. Shinde[2], Yogesh B. Gurav[3], Rupesh J. Partil[4], Sandeep U. Kadam[5]
Vishwakarma Institute of Information Technology, Pune[1]
Dr. D. Y. Patil Institute of Technology, Pimpri, Pune[2]
Navsahyadri College of Engineering, Pune[3, 4]
Anantrao Pawar College of Engineering & Research, Pune[5]

***Abstract*—Users may now choose from a vast range of compiler optimizations. These optimizations interact in a variety of sophisticated ways with one another and with the source code. The order in which optimization steps are applied can have a considerable influence on the performance obtained. As a result, we created a revolutionary compiler optimization prediction model. Our model comprises three operational phases: model training, feature extraction, as well as model exploitation. The model training step includes initialization as well as the formation of candidate sample sets. The inputs were then sent to the feature extraction phase, which retrieved static, dynamic, and improved entropy features. These extracted features were then optimized by the feature exploitation phase, which employs an improved hunger games search algorithm to choose the best features. In this work, we used a Convolutional Neural Network to predict compiler optimization based on these selected characteristics, and the findings show that our innovative compiler optimization model surpasses previous approaches.**

***Keywords*—Compiler optimization prediction; feature extraction; feature exploitation; improved hunger games search algorithm; convolutional neural network**

I. INTRODUCTION

As per Moore's Law, the density of transistors doubles every 2 years. Compilers, on the other hand, progress at a pace of a couple percent of year. Compilers were vital tools for connecting written software to destination hardware. In the field of compilers, there's several unresolved research issues [1].Compilers play a crucial role in software development. Its core objective is to boost software productivity [2].

Compilers were liable for two tasks: translation as well as optimization. They must first effectively convert programmes into binary. Secondly, they must discover the most cost-effective translation. There are numerous valid translations, each of which performs distinctively. The great majority of studies and technological activities are centered on this second performance objective, which has been referred to as optimization. The objective was mislabeled because, until recently, most people rejected obtaining an ideal translation as a difficult and impractical task [3].Compilers are now being improved so that every code block in a programmed may be transformed into an efficient application [4]. Traditional compiler optimization seems to be a difficult process with no assurances of producing the most effective and quickest target code [5].A compiler enables a multitude of code optimizations that could be activated or disabled via a compilation flag in order to enhance the throughput of compiled applications. Nevertheless, because the influence of compiler optimizations largely dependent on programme features (e.g., programme structures), the identical optimizations may not surely result in the identical runtime speed boost when implemented to various programmes [6].

Furthermore, there's an infinite range of flag combos owing to the enormous count of optimization flags. Users may find it difficult to comprehend all of the flags including their combos, and to correctly decide which flags should be activated or disabled in attempt for built programmes to attain the desired runtime performance [7].Compilers for machine learning (ML) tackle a lot of optimization issues in order to convert an ML programmed, which is often expressed as a tensor computational graph, into an efficiently executable for a hardware destination [8]. Prior efforts [9] – [14] have permitted optimizations that are implemented at the very same point in the compilation pipeline, notably the loop conversion phase. However, since compiler modifications are arranged as passes to minimize complication and also have rigorous ordering limitations, this is unfeasible in production compilers.

With a compiler's optimization capabilities influencing so many parts of product development, understanding and evaluating a compiler's optimization technology is more critical than ever. In this work, an improved optimization prediction model was created, which not only decreases computational time but also enables the compilers with faster convergence, more stable balance, and high-quality outcomes by selecting appropriate optimization. Our work made the following contributions:

- Several high-level characteristics may arise from the coefficients as a result of improved entropy extraction, which boosts the compiler optimization prediction.
- An Improved Hunger Game Search optimization was proposed to provide a very competitive performance to the compiler with less computational time.

The following is the flow of this article: Section II covers some previous relevant research, Section III gives a brief presentation of our proposed compiler optimization prediction model, Section IV gives the outcomes of our work, and Section V contains the conclusion, while the following section includes the references for this work.

II. Related Works and Review

Some of the researches presented by various researchers on compiler optimization were reviewed here.

Hui et al. [15] presented the ALIC iterative compiler optimization parameters estimation model, which has minimal overheads. Firstly, the target programmes were defined using static-dynamic characteristics format depending on feature significance, as well as an early optimization prediction model was built using the classifier. Subsequently, for every sample, a dynamic amount of sample observation methodology was being used. The most beneficial test from the collection of candidate samples typically the chosen and labeled with each mark increase the count of sample data. The optimization prognosis system is then built by using intermediate prediction network, which actively learns candidate samples.

Tiago et al. [16] suggested a new exploration approach to determine a compiler optimization strategy. This hybrid methodology utilizes previously created sequences for a series of training programmes in order to uncover optimizations as well as their deployment order. A clustering method selects optimizations during the first stage, and then a metaheuristic algorithm determines the order wherein the compiler would execute every optimization in the latter. The LLVM compilers as well as an I7 processor have been used to assess this strategy.

Supun et al. [17] developed HUMMINGBIRD, a unique prototype scoring technique that incorporates featurization operators with classic ML designs (e.g., decision trees) into a limited collection of tensor operational processes. This method decreases infrastructure overhead by using current investments in Neural Net compilers but also runtimes to produce efficient calculations for both CPU as well as hardware accelerators. The findings indicate that HUMMINGBIRD performs compatible.

Mircea et al. [18] introduced MLGO1, a methodology for comprehensively incorporating machine learning methods into an industrial compiler— LLVM. It's the first time ML has been fully integrated in a sophisticated compiler run in a real-world context. It's in the LLVM main repository. As contrasted to the state-of-the-art LLVM -Oz, we apply two alternative ML techniques to train the inlining-for-size method: Policy Gradient as well as Evolutionary Algorithms.

Aleksandar et al. [19] presented a revolutionary JIT compiler inlining approach that gradually investigates a program's call network and switches between inlining as well as optimizations. Three new heuristics have been developed to steer this inliner. Graal, a dynamic JIT compiler for the HotSpot JVM, was used to create this technique. Benchmarks such as Java DaCapo, Scalabench, and others were utilized to test the suggested algorithm.

Conventional systems to prediction model creation frequently employ a random selection search strategy, which can often lead to information redundancy. Moreover, the sample program gets exposed to a fixed number of repetitive measurements due to the influence of run-time disturbances. Unfortunately, if there are few sounds, the recurrent measurements will lead to a significant loss of iterative compilation time overheads. Decreasing iterative compilation overheads and predicting an appropriate compiler optimization with less computational time and increased compiler performance was still challenging.

III. Proposed Compiler Optimization Prediction Model

This proposed compiler optimization model comprises three working phases: model training, feature extraction [24, 25], as well as model exploitation (feature selection). First, the inputs were fed into the model training phase, which tries to match the right weights as well as bias to a learning algorithm [26, 27] in order to minimize a loss function throughout the validation range. The retrieved characteristics, such as static, dynamic, as well as improved entropy, were then transferred to the model exploitation phase [21], where the optimal features were chosen utilizing the improved chaos game optimization. These optimized features were given to Convolutional Neural Network for prediction of compiler optimization. The architecture of our improved compiler optimization prediction model is given in Fig. 1.

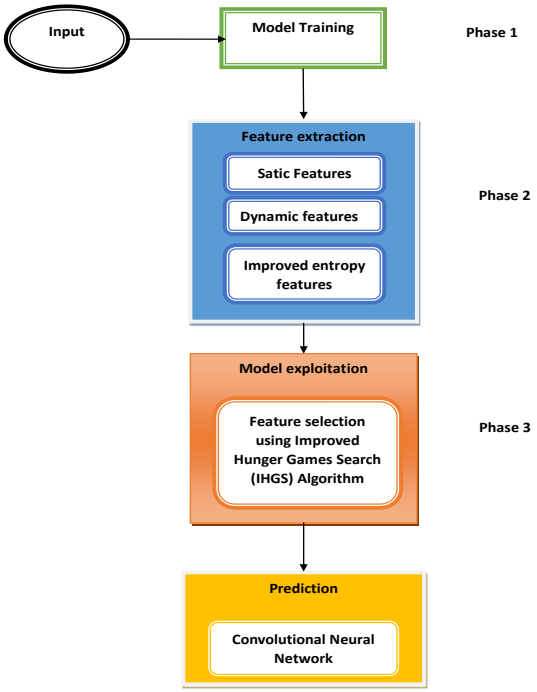


Fig. 1. Proposed improved compiler optimization prediction model architecture.

A. Model Training Phase

The initialization and candidate sample set generation takes place in the model training phase. The initialization model will be built in the training set with some labeled samples. The initialization model will be used as the intermediate prediction model later. The candidate samples set include both the unlabeled samples in the training set and the labeled samples with the number of observations.

B. Feature Extraction

Outputs from model training phase were given to the feature extraction phase to extract the static, dynamic and improved entropy features [26, 27].

1) Static features: The values of static features do not vary over time and are set for every sample. The lists of the static features extracted in this work were shown in Fig. 2.

2) Dynamic features: The values of dynamic features fluctuate over time and are not constant. Fig. 3 depicts the dynamic characteristics retrieved in this work.

3) Improved entropy feature extraction: The count of coefficients is generally so large that it is challenging to utilize them directly as features for categorization or prediction. As a result, several high-level features might emerge from these coefficients for improved prediction. Entropy seems to be a tool for measuring the uncertainty of data content in specific mechanisms, and it is frequently employed in signal analysis, pattern recognition, pattern matching, and other fields. . Some kinds of entropy include Shannon entropy (SE), log energy entropy (LEE), Renyi entropy (RE), as well as Tsallis entropy (TE). Renyi entropy is utilized to retrieve features from input data in this work. Entropy may be estimated via energy. Wavelet energy, described as Eq. (1), will be used to assess the data of the coefficient a of the b-th node at the c-th level.

$$E_{a,b,c} = \left\| d_{a,b,c} \right\|^2 \tag{1}$$

The total energy for the b-th node at the c-th level may then be determined utilizing Eq. (2)

$$E_{a,b} = \sum_{c=1}^{M} E_{a,b,c} \tag{2}$$

Where M indicates the number of node matching coefficients Eq. (3) may be used to compute the probability of the c-th coefficient at its associated node:

$$\rho_{a,b,c} = E_{a,b,c} / E_{a,b} \tag{3}$$

Where the sum of $\rho_{a,b,c}$ equals 1.

Renyi Entropy of order q(q ≥ 0 and q 6= 1) gets described as

$$RE(s) = \frac{1}{1-\beta} \log\left(\sum_{a=1}^{M} \rho_a^q \right) \tag{4}$$

The parameter of q in RE should be optimized to provide better results. In our work, the improved entropy features were extracted using the eq. (5)

$$RE(\rho) = \frac{1}{1-\beta} \log\left(\sum_{a=1}^{M} \rho_a^{\beta} \right)^{1/\beta} * \omega_a \tag{5}$$

Fig. 2. Extracted static features.

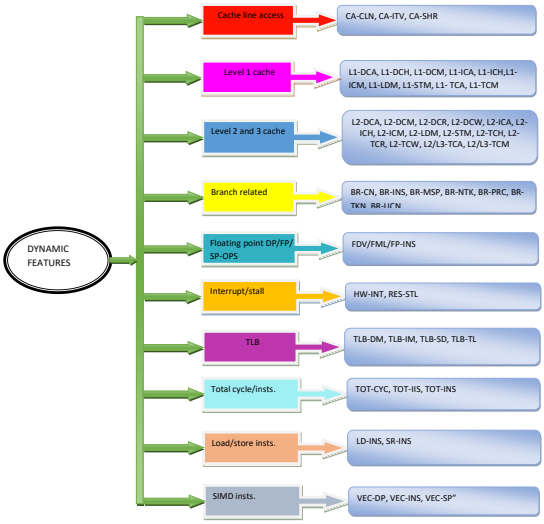


Fig. 3. Extracted dynamic features.

Here ω_a denotes the weight, which is calculated by

$$\omega_a = \frac{\rho_a^{\beta}}{\sum_{b=1}^{m} \rho_b^{\beta}} \tag{6}$$

Following the calculation of the entropy of each terminal node, the entropies of all terminal nodes are concatenated to form a feature vector. These features were sent to the model exploitation phase for feature selection.

C. Model Exploitation

The feature selection procedure is taking place during this model exploitation phase. When creating forecasting models, feature selection is the technique of minimizing the count of input variables. It is preferable to limit the count of input variables in order to reduce modeling computational costs and, in certain situations, increase model performance. For that reason we used an improved HGS Optimization.

1) Improved Hunger Game Search (IHGS) Optimization: The HGSO technique is influenced by normal animal behaviors including terror of being eaten by predators as well as hunger. The mathematical modelling of the HGSO strategy is explained in this portion of the publication. The modelling is based on social selection and hunger-driven behaviour.

This section quantitatively models the approaching behaviour of hunger. Eq. (7), which explains the foraging hunger as well as individual supportive communication activities, contains the game instructions. The contraction mode is imitated by the mathematical formula in Eq. (7).

$$\overrightarrow{Z(t+1)} = \begin{cases} \overrightarrow{Z(t)}.(1+randn(1)), & r1 < l \\ \overrightarrow{W_1}.\overrightarrow{Z_d} + \vec{R}.\overrightarrow{W_2}.\left|\overrightarrow{Z_d} - \overrightarrow{Z(t)}\right|, & r1 > l, r2 > H \\ \overrightarrow{W_1}.\overrightarrow{Z_d} - \vec{R}.\overrightarrow{W_2}.\left|\overrightarrow{Z_d} - \overrightarrow{Z(t)}\right|, & r1 > l, r2 < H \end{cases} \quad (7)$$

where $\overrightarrow{Z(t)}$ signifies the location of all individuals, $\overrightarrow{Z_d}$ indicates the position of the best individual, $\overrightarrow{W_1}$. and $\overrightarrow{W_2}$ seem to be hunger weights of hunger, $\vec{R}$ is between [-a, a], r1 and r2 seem to be random numbers between [0, 1], randn(1) denotes a normal distributed random number, and t seems to be the count of current iterations. The parameter l represents the HGSO algorithm's control variable that governs the algorithm's sensitivity. H stands for variation control for all locations.

2) Opposite behavior learning: Amongst the most effective instructional procedures, opposition-based learning (OBL), has been extensively embraced as an excellent learning phase to improve the searching capabilities of algorithms. When assessing a solution Y to a given issue, a novel opportunity will be gained that brings the candidate solution closer to the optimal solution if the opposing solution of Y is estimated at the same time. The opposing number as well as opposite point notions were described as follows.

OBL is a learning approach that is centered on the inverse number Y^o. Y is described as a real number, Y ∈ [e, f]. Y^o's opposite may be defined as (8), where e, f are the bounds.

$$Y^o = e + f - Y \quad (8)$$

When $Y = (Y_1, Y_2, \ldots Y_D)$seems to be a point in a D-dimensional space, $Y_j, \ldots, Y_D$. e_j as well as f_j represent the current population's low and high borders, which vary with each iteration. An opposing point in several dimensions has been described as

$$Y_j^o = e_j + f_j - Y_j, j = 1:D \quad (9)$$

In our work, to generate chaotic opposite solution we have used the following equation

$$Y_j = lbj + rand * (efj - lfj) \quad (10)$$

Here rand was generated using the sine map.

$$z_{u+1} = \frac{e}{4}\sin(\pi z_k) \qquad e \in (0,4) \quad (11)$$

Variation control for all positions H stated in eq. (12)

$$H = \text{sech}(|F(i) - BF|) \quad (12)$$

where F(i) represents the cost function value of every population, I = 1, 2,..., n, BF represents the best cost function value acquired during the latest incarnation, and Sech represents the hyperbolic function and thus is equal to $\left(sech(x) = \frac{2}{e^x + e^{-x}}\right)$.

In our work, we used the reciprocal of the hyperbolic function Csch,which is expressed in eqn.,(13)

$$H = Csch(|F(k) - BF|) \quad (13)$$

Here $Csch = \frac{2e^x}{2e^x - 1}$

Eqn.,(14) gives the expression for $\vec{R}$.

$$\vec{R} = 2 \times h \times r - h$$
$$h = 2 \times \left(1 - \frac{t}{Max_{iter}}\right) \quad (14)$$

Where Max_{iter} denotes the maximum number of iterations and rand denotes a random number between [0, 1]

3) Hunger role: This portion quantitatively models the hunger behavior of all individuals during the search. The formula for W_1 is given in Eq. (15).

$$\overrightarrow{\wedge W_1(k)} = \begin{cases} hungry(k).\frac{N}{SHungry} \times ra4, & ra3 < l \\ 1 & ra3 > l \end{cases} \quad (15)$$

The expression for $\overrightarrow{W_2}$ is presented in Eq. (16).

$$\overrightarrow{\wedge W_2(k)} = \left(1 - \exp\left(-|hungry(k) - SHungry|\right)\right) \times ra5 \times 2 \quad (16)$$

where N represents population size, hungry means population starvation, SHungry represents the total of population starvation, i.e., sum(hungry), as well as ra3, ra4, and ra5 signify random values between [0, 1]. Each population's starving is quantitatively represented in Eq. (17).

$$hungry(k) = \begin{cases} 0 & AllFitness(k) == BF \\ hungry(k) + H_{new}, & AllFitness(k) = BF \end{cases} \quad (17)$$

whereAllFitness(k) would be the present iteration's cost function value for every population Depending on the real starving, a new starvation H_{new} H is added. The equation represents the formula for H_{new}.

$$H_{new} = \begin{cases} LH \times (1+ra) & TH < LH \\ TH, & TH \geq LH \end{cases} \quad (18)$$

$$TH = \frac{F(k) - BF}{WF - BF} \times ra6 \times 2 \times (UB - LB) \quad (19)$$

where H_{new} has been constrained to a lower bound LH, ra would be a random number among [0, 1], WF as well as BF seem to be the worst best fitness acquired during the latest iteration, respectively, F (k) has become the fitness of every population, ra6 would be a random number between [0, 1], and LB as well as UB are indeed the lower and upper boundaries of the dimensions, respectively. The selected features were sent to the CNN for compiler optimization prediction.

D. Prediction using CNN

Convolutional networks were deep training strategies that extract information from input pictures by convolving them with filters or kernels. Convolution of a GCG picture with a $f_S C f_S$ filter learns the same characteristic on the whole picture. After each action, the window moves, and the feature maps learn the characteristics. The feature maps record the image's local receptive area and operate with mutual weights as well as biases. Equation (20) depicts the output matrix size without padding, whereas Equation (21) depicts the convolution procedure. Padding has been utilized to keep the size of the given picture constant. The output picture size is the same as the input image size in a 'SAME' padding, and there is no padding in a "VALID" padding. Equation depicts the output matrix size with padding (22).

$$GCG * f_S C f_S = G - F + 1 \quad (20)$$

$$o = \sigma\left(m + \sum_{v=0}^{2} \sum_{q=0}^{2} w_{v,q} \chi_{\alpha+v,m+q}\right) \quad (21)$$

$$GCG * f_s * f_s = (G + 2P - f_s)/(S + 1) \quad (22)$$

Here, O is the output, P is the padding, S is the stride, m is the bias, σ is the sigmoidal activation function, w is a 3x3 weight matrix of shared weights and $\chi_{p1,p2}$ is the input activation at position p1,p2. The output O provides the prediction results.

IV. Results and Discussion

A. Simulation Setup

The unique methodology for compiler optimization utilizing IHGS was implemented in Python. The standard performance evaluation group created the SPEC CPU2006 training set to evaluate general-purpose CPU performance [20]. The input scale of the SPEC2006 benchmark may be split into test, train, as well as reference scales; we utilize the reference scale to test." In this case, analysis was performed for multiple measures such as accuracy [22,23] and error metrics such as MSE, MSLE, and so on. In addition, IGHS outperformed the HGS, PRO, CMBO, ARCHOA, DO, as well as GOA models.

B. Performance Analysis

The research on diverse metrics including accuracy, sensitivity, specificity as well as precision was detailed here. Here, the analysis was done for LPs (Learning Percentages) of 60, 70, 80 and 90 over HGS, PRO, CMBO, ARCHOA, DO, GOA models which is shown in Fig. 4. For 60 LP, CMBO and HGS achieve the accuracy rate of 0.69 and 0.76 whereas our proposed IHGS model achieves the accuracy rate of 0.84. At 80 and 90 LPs our proposed IHGS achieves the accuracy rate of 0.9 and 0.94 which is higher than other models. When our proposed IHGS achieves the precision value of 0.9, ARCHOA, CMBO models achieves only 0.8 and 0.81 for 60 LP which proves the superiority of proposed IHGS model. For 80 and 90 LPs, PRO model attain the sensitivity and specificity values of 0.83, 0.85 and 0.83, 0.85 while our proposed IHGS method achieves the values of 0.85, 0.93 and 0.89, 0.94 which proves that our proposed IHGS method achieves high performance for the compiler optimization identification than other conventional models.

The most often employed KPIs to estimate forecast accuracy were MAPE, MAE, RMSE(MSE), as well as MSLE which were analyzed for the models such as HGS, PRO, CMBO, ARCHOA, DO and GOA for 60, 70, 80 and 90 LPs which is compared with our proposed IHGS model that is shown in Fig. 5. "MAE is indeed a metric of error between matched observations reflecting the same phenomena in statistics." The MAE should be less to increase forecast accuracy. Our proposed IHGS method obtain the MAE value of 0.48, 0.45, 0.42 and 0.4 for 60, 70, 80 and 90 LPs which is lower than other conventional methods. MSLE may be regarded of as a measurement of the ratio between true as well as forecasted values. When HGS method achieves the high MAPE values of 2.3, 1.5, 0.7 and 2.0 for 60, 70, 80,90 LPs, our proposed IHGS method obtain the values of 0.5, 0.4, 0.3 and 0.2.Unlike MAE, RMSE doesn't really handle every error in the same way. It prioritises the most critical errors. That implies that a single large mistake might result in a very bad RMSE. Our proposed IHGS approach yields MSE values of 0.49, 0.46, 0.43, as well as 0.42 for all LPs, which is lower than other standard approaches. In statistics, the MAPE, also referred as the MAPD, is specified as "a metric of prognosis accuracy of a forecasting technique". "The MSE or MSD of an estimator in statistics estimates the average of the squares of the errors, or the average squared difference between the predicted as well as real values." For optimized prediction, the MSE and MAPE must be lower. When the CMBO approach produces MSLE values of 0.27, 0.22, 0.23, 0.26, our proposed IHGS method achieves lower values of 0.23, 0.22, 0.21, 0.20, demonstrating that our proposed IHGS method can outperform other standard compiler optimization forecasting models.

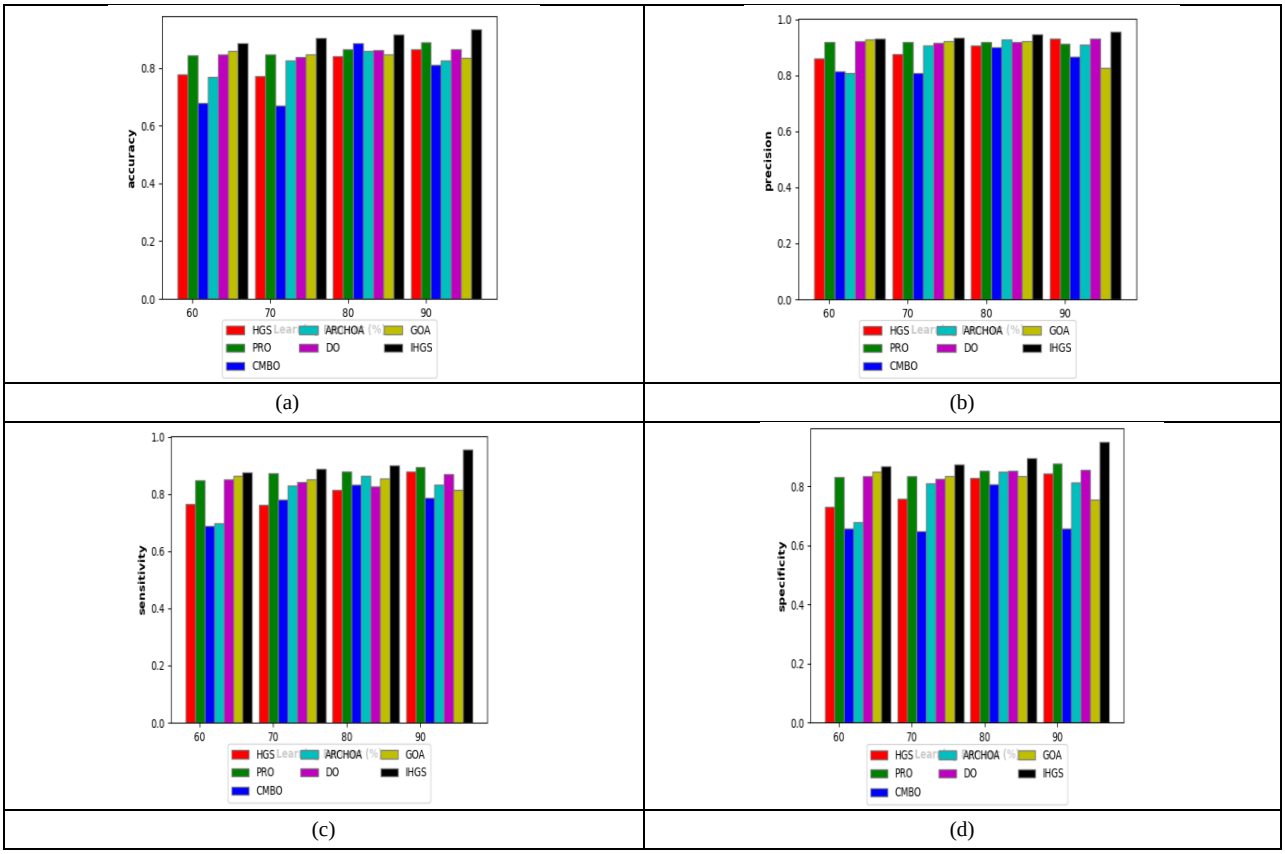


Fig. 4. Comparison of performance matrices such as (a)Accuracy, (b)Precision, (c) Sensitivity, (d) Specificity.

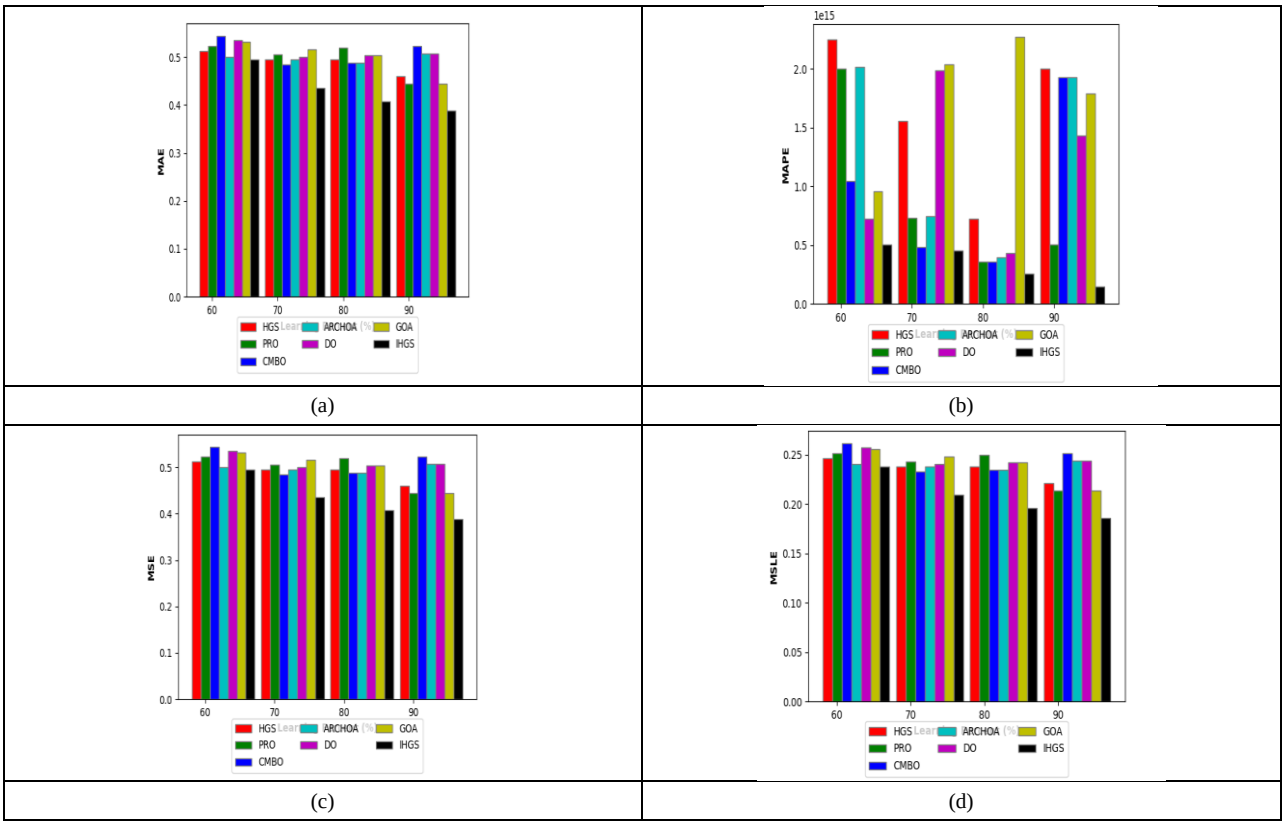
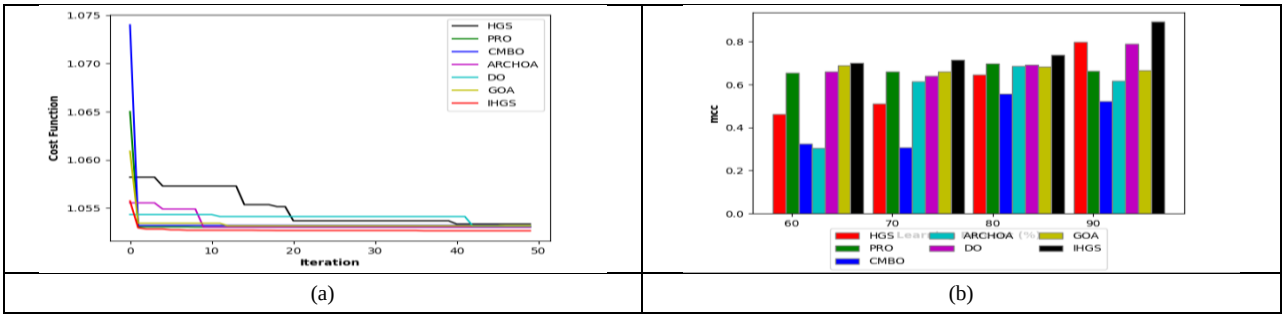


Fig. 5. Comparison of performance such as (a) MAE, (b) MAPE, (c) MSE, (c) MSLE.

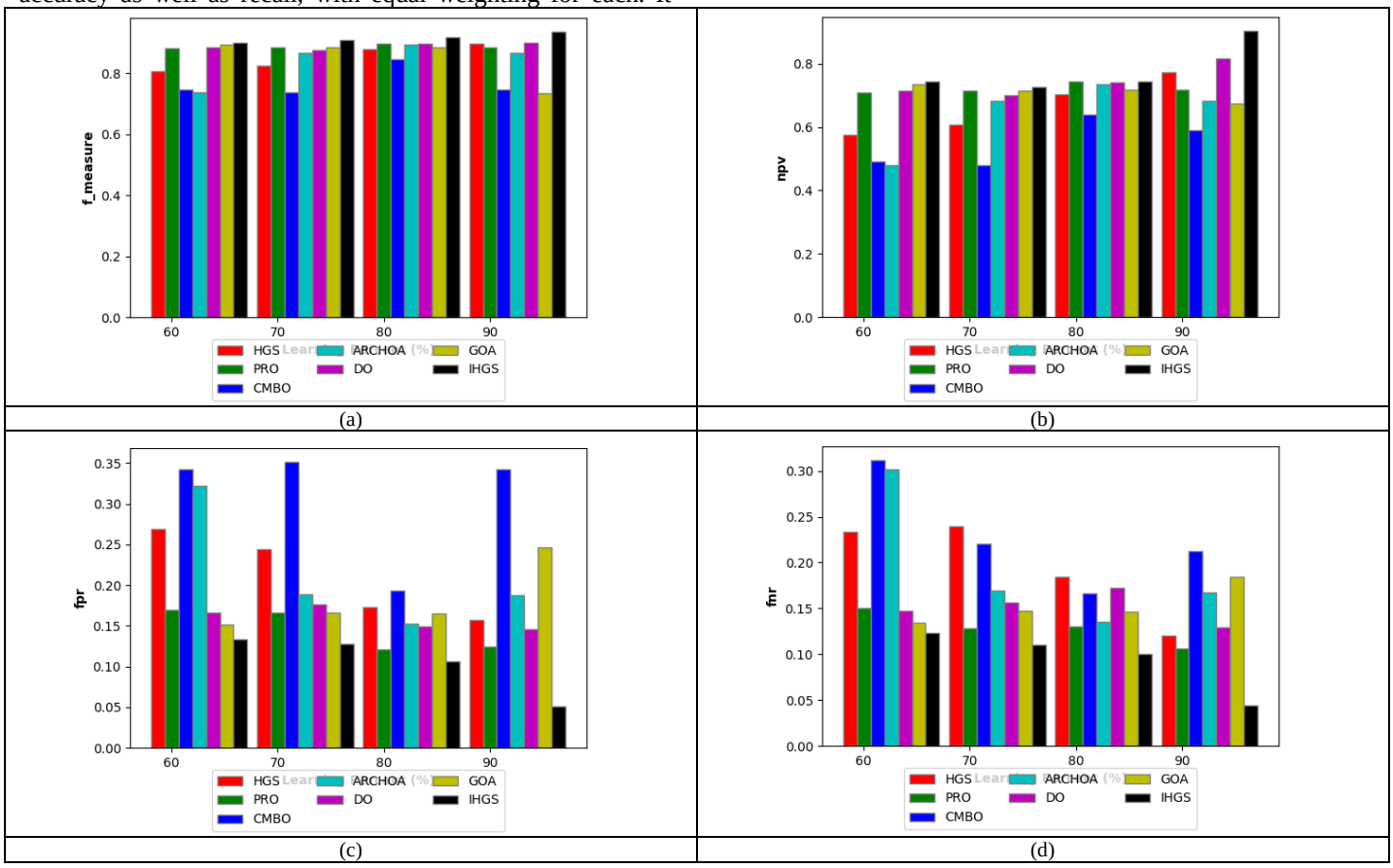


(a) (b)

Fig. 6. Comparison of MSC and Cost function values for different LPs.

The cost function for 0-50 iterations was evaluated in this work, to assess the performance of our proposed IHGS model which is shown in Fig. 6(a). When cost function of CMBO is 1.074, our proposed method obtains the value of 1.052 for iteration 0 whereas 1.058 and 1.062 for GOA method which states that IHGS method has the lowest cost function for all five iterations. A more trustworthy statistical rate known as the Matthews correlation coefficient (MCC) was evaluated, which yields a high score only if the prediction performed well in all the confusion matrix classes which are given in Fig. 6(b). MCC values of all the LPs were 0.67, 0.68, 0.7 and 0.9 for our proposed IHGS method which proves our prediction was performed well with good results.

The F-measure is derived as the harmonic mean of accuracy as well as recall, with equal weighting for each. It enables a system to be assessed utilizing a single score that accounts for both accuracy and recall that is useful for reporting system performance as well as comparing models. With F1 measure, fnr, fpr, as well as npv values were also estimated and compared with conventional models which is shown in Fig. 7. In comparison to the CMBO as well as ARCHOA approaches, our proposed IHGS method achieves f1 measure values of 0.9, 0.92, 0.93, and 0.95 for all LPs. The IHGS approach produces anpv value of 0.92 for 90 LP, whereas the CMBO and ARCHOA methods yield relatively low values such as 0.6 and 0.68. Our proposed IHGS technique achieves fpr values of 0.13, 0.12, 0.11, and 0.05, and assessed fnr values of 0.13, 0.12, 0.11, and 0.04, which are lower than other traditional methods, demonstrating that IHGS method achieves superior performance than other methods.



(a) (b)

(c) (d)

Fig. 7. Comparison of performance matrices (a) F1 measure, (b) fnr, (c) fpr, (d) npv values.

The computation time for each method which is compared with our proposed IGHS method is shown in Table I which shows that IGHS method have low computational time of 55.64. Accuracy and error matrices also compared with without optimization, to evaluate the performance of our proposed IGHS method which is given in Tables II and III. Without optimization our proposed method achieves only 88% accuracy whereas, with optimization it achieves 95% accuracy.

TABLE I. COMPUTATIONAL TIME COMPARISON FOR CONVENTIONAL AND PROPOSED APPROACHES

Methods	Computation time
HGS	76.2114
PRO	200.538
CMBO	537.269
ARCHOA	139.731
DO	95.0285
GOA	111.956
IHGS	55.6467

TABLE II. PERFORMANCE MATRICES OF THE PROPOSED IGHS METHOD (ACCURACY MATRICES) WITH AND WITHOUT OPTIMIZATION

Accuracy matrices	Proposed with Optimization	Proposed without Optimization
sensitivity	0.955614	0.882424
specificity	0.949348	0.765633
accuracy	0.933705	0.843407
precision	0.955031	0.882424
F -measure	0.936337	0.882424
mcc	0.892841	0.648057
npv	0.903579	0.765633
fpr	0.050652	0.234367
fnr	0.044386	0.117576

TABLE III. PERFORMANCE MATRICES OF THE PROPOSED IGHS METHOD (ERROR MATRICES) WITH AND WITHOUT OPTIMIZATION

Error matrices	Proposed with Optimization	Proposed without Optimization
MSE	0.388571	0.515957
MAE	0.388571	0.515957
MSLE	0.185908	0.247893
MAPE	1.43E+14	2.32E+15

Table IV Shows the RMSE values obtained for distinct datasets and statistical tests such as Wilcoxon and chi-square were conducted for conventional and proposed methods and the p, statistic values were tabulated in Tables V and VI which proves the effectiveness of our proposed compiler optimization prediction approach.

TABLE IV. RMSE FOR EACH BENCHMARK IN THE DATASET

Bench mark	RMSE
400.perlbench	0.701
401.bzip2	0.707107
403.gcc	0.701
429.mcf	6.61E-01
445.gobmk	0.809156
456.hmmer	0.75
458.sjeng	0.75
462.libquantum	0.707107
464.h264ref	0.661438
471.omnetpp	0.696107
473.astar	0.612372
483.xalancbmk	0.644378

TABLE V. COMPARISON OF WILCOXAN TEST RESULTS FOR PROPOSED AND CONVENTIONAL METHODS

Methods	P value	Statistic
HGS	1.36E-06	253
PRO	6.48E-18	2701
CMBO	2.54E-29	7475
ARCHOA	6.97E-29	7.63E+03
DO	3.55E-12	1128
GOA	2.54E-29	7475
IHGS	2.54E-29	7875

TABLE VI. CHI-SQUARE TEST RESULTS FOR PROPOSED AND TRADITIONAL TECHNIQUES

Methods	P value	Statistic
HGS	5.83E-03	45
PRO	8.83E-05	59
CMBO	5.83E-03	45
ARCHOA	5.83E-03	4.50E+01
DO	4.37E-06	68
GOA	5.83E-03	45
IHGS	1.54E-06	71

V. CONCLUSION

Selecting the optimal, or even a good, combination of optimizations for an unpredictable programmed on an arbitrary design is a task so tough that traditional manual analysis approaches are impractical. For that reason, a novel optimization prediction model with improved optimization was developed in this work, which has three working phases including model training, feature selection as well as feature exploitation phase. First, the inputs are being sent to the model

training phase that aims to link the appropriate weights as well as bias to a learning algorithm in order to minimize a loss function throughout the validation range. The retrieved characteristics, including static, dynamic, and enhanced entropy, were then transferred to the model exploitation phase, where the best features was determined using the improved chaos game optimization. These improved characteristics were fed into a Convolutional Neural Network to predict the appropriate compiler optimization.

REFERENCES

[1] Ashouri, A.H., Killian, W., Cavazos, J., Palermo, G. and Silvano, C., 2018. A survey on compiler autotuning using machine learning. ACM Computing Surveys (CSUR), 51(5), pp.1-42.

[2] Georgiou, K., Chamski, Z., Amaya Garcia, A., May, D. and Eder, K., 2022. Lost in translation: Exposing hidden compiler optimization opportunities. The Computer Journal, 65(3), pp.718-735.

[3] Wang, Z. and O'Boyle, M., 2018. Machine learning in compiler optimization. Proceedings of the IEEE, 106(11), pp.1879-1901.

[4] Tağtekin, B., Höke, B., Sezer, M.K. and Öztürk, M.U., 2021, August. FOGA: Flag Optimization with Genetic Algorithm. In 2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA) (pp. 1-6). IEEE.

[5] Gong, Z., Chen, Z., Szaday, J., Wong, D., Sura, Z., Watkinson, N., Maleki, S., Padua, D., Veidenbaum, A., Nicolau, A. and Torrellas, J., 2018. An empirical study of the effect of source-level loop transformations on compiler stability. Proceedings of the ACM on Programming Languages, 2(OOPSLA), pp.1-29.

[6] Chen, J., Xu, N., Chen, P. and Zhang, H., 2021, May. Efficient compiler autotuning via bayesian optimization. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (pp. 1198-1209). IEEE.

[7] L. P. Cáceres, F. Pagnozzi, A. Franzin, and T. Stützle, "Automatic configuration of gcc using irace," in International Conference on Artificial Evolution (Evolution Artificielle). Springer, 2017, pp. 202–216.

[8] Phothilimthana, P.M., Sabne, A., Sarda, N., Murthy, K.S., Zhou, Y., Angermueller, C., Burrows, M., Roy, S., Mandke, K., Farahani, R. and Wang, Y.E., 2021, September. A Flexible Approach to Autotuning Multi-Pass Machine Learning Compilers. In 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT) (pp. 1-16). IEEE.

[9] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L. and Guestrin, C., 2018. {TVM}: An Automated {End-to-End} Optimizing Compiler for Deep Learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18) (pp. 578-594).

[10] Steiner, B., Cummins, C., He, H. and Leather, H., 2021. Value learning for throughput optimization of deep learning workloads. Proceedings of Machine Learning and Systems, 3, pp.323-334.

[11] Zheng, L., Jia, C., Sun, M., Wu, Z., Yu, C.H., Haj-Ali, A., Wang, Y., Yang, J., Zhuo, D., Sen, K. and Gonzalez, J.E., 2020. Ansor: Generating {High-Performance} Tensor Programs for Deep Learning. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20) (pp. 863-879).

[12] Jia, Z., Padon, O., Thomas, J., Warszawski, T., Zaharia, M. and Aiken, A., 2019, October. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In Proceedings of the 27th ACM Symposium on Operating Systems Principles (pp. 47-62).

[13] Zheng, S., Liang, Y., Wang, S., Chen, R. and Sheng, K., 2020, March. Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system. In Proceedings of the Twenty-Fifth International Conference on Li, M., Zhang, M., Wang, C. and Li, M., 2020. Adatune: Adaptive tensor program compilation made efficient. Advances in Neural Information Processing Systems, 33, pp.14807-14819.

[14] Li, M., Zhang, M., Wang, C. and Li, M., 2020. Adatune: Adaptive tensor program compilation made efficient. Advances in Neural Information Processing Systems, 33, pp.14807-14819.

[15] Liu, H., Zhao, R., Wang, Q. and Li, Y., 2018. ALIC: A low overhead compiler optimization prediction model. Wireless Personal Communications, 103(1), pp.809-829.

[16] de Souza Xavier, T.C. and da Silva, A.F., 2018. Exploration of compiler optimization sequences using a hybrid approach. Computing and Informatics, 37(1), pp.165-185.

[17] Nakandala, S., Saur, K., Yu, G.I., Karanasos, K., Curino, C., Weimer, M. and Interlandi, M., 2020. A tensor compiler for unified machine learning prediction serving. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20) (pp. 899-917).

[18] Trofin, M., Qian, Y., Brevdo, E., Lin, Z., Choromanski, K. and Li, D., 2021. Mlgo: a machine learning guided compiler optimizations framework. arXiv preprint arXiv:2101.04808.

[19] Prokopec, A., Duboscq, G., Leopoldseder, D. and Wirthinger, T., 2019, February. An optimization-driven incremental inline substitution algorithm for just-in-time compilers. In 2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO) (pp. 164-179). IEEE Algorithm for just in time compilers. In 2019, IEEE/ACM International Symposium on Code Generation and Optimization(CGO)(pp.164-179).

[20] SPEC CPU2006: SPEC CPU2006 benchmark suite. http://www.spec.org/cpu/.

[21] Sandeep U. Kadam, Sagar B. Shinde, Yogesh B. Gurav, Sunil B Dambhare and Chaitali R Shewale, "A Novel Prediction Model for Compiler Optimization with Hybrid Meta-Heuristic Optimization Algorithm" International Journal of Advanced Computer Science and Applications(IJACSA), 13(10), 2022. http://dx.doi.org/10.14569/ IJACSA.2022.0131068.

[22] A. D. Sutar and S. B. Shinde, "ECU diagnostics validator using CANUSB," 2017 International Conference on Inventive Computing and Informatics (ICICI), Coimbatore, India, 2017, pp. 856-860, doi: 10.1109/ICICI.2017.8365257.

[23] A. D. Sutar and S. B. Shinde, "ECU Health Monitor Using CANUSB," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 2018, pp. 415-419, doi: 10.1109/ICICCT.2018.8473000.

[24] S. Shinde and R. B. Waghulade, "An improved algorithm for recognizing mathematical equations by using machine learning approach and hybrid feature extraction technique," 2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE), Karur, India, 2017, pp. 1-7, doi: 10.1109/ICEICE.2017.8191926.

[25] S. Shinde, R. B. Waghulade and D. S. Bormane, "A new neural network based algorithm for identifying handwritten mathematical equations," 2017 International Conference on Trends in Electronics and Informatics (ICEI), Tirunelveli, India, 2017, pp. 204-209, doi: 10.1109/ICOEI.2017.8300916.

[26] Kalyani Wagh, K. Vasanth, Sagar Shinde , "Emotion Recognition Based On Eeg Features With Various Brain Regions", Indian Journal of Computer Science and Engineering (IJCSE), Vol. 13 No. 1 Jan-Feb 2022, p-ISSN : 2231-3850, DOI : 10.21817/indjcse/2022/v13i1/ 221301095.

[27] Khoje, s., Shinde, S. Evaluation of Ripplet Transform as a Texture Characterization for Iris Recognition. J. Inst. Eng. India Ser. B (2023). https://doi.org/10.1007/s40031-023-00863-6.