International Information and Engineering Technology Association
Advancing the World of Information and Engineering

International Journal of Computational Methods and Experimental Measurements
Vol. 12, No. 3, September, 2024, pp. 281-287
Journal homepage: http://iieta.org/journals/ijcmem

Optimizing Program Efficiency by Predicting Loop Unroll Factors Using Ensemble Learning

Esraa H. Alwan[1*], Ali Kadhum M. Al-Qurabat[2]

[1] Department of Computer Science, College of Science for Women, University of Babylon, Babylon 51002, Iraq
[2] Department of Cyber Security, College of Sciences, Al-Mustaqbal University, Babylon 51001, Iraq

Corresponding Author Email: esraa.hadi@uobabylon.edu.iq

Copyright: ©2024 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/ijcmem.120308

Received: 19 August 2024
Revised: 10 September 2024
Accepted: 19 September 2024
Available online: 30 September 2024

Keywords:
loop unroll, compiler optimization, ensemble learning, Random Forest, Bagging, XGBoost

ABSTRACT

Loop unrolling is a well-known code-transforming method that can enhance program efficiency during runtime. The fundamental advantage of unrolling a loop is that it frequently reduces the execution time of the unrolled loop when compared to the original loop. Choosing a large unroll factor might initially save execution time by reducing loop overhead and improving parallelism, but excessive unrolling can result in increased cache misses, register pressure, and memory inefficiencies, eventually slowing down the program. Therefore, identifying the optimal unroll factor is of essential importance. This paper introduces three ensemble-learning techniques—XGBoost, Random Forest (RF), and Bagging—for predicting the efficient unroll factor for specific programs. A dataset comprises various programs derived from many benchmarks, which are Polybench, Shootout, and other programs. More than 220 examples, drawn from 20 benchmark programs with different loop iterations, used to train three ensemble-learning methods. The unroll factor with the biggest reduction in program execution time is chosen to be added to the dataset, and ultimately it will be a candidate for the unseen programs. Our empirical results reveal that the XGBoost and RF methods outperform the Bagging algorithm, with a final accuracy of 99.56% in detecting the optimal unroll factor.

1. INTRODUCTION

A considerable percentage of a program's execution time spent in a small portion of its code, usually loop constructs. Research reveals that about ninety percent of the time spent executing a program focuse on only ten percent of its code. Thus, improving these frequently run parts can significantly improve the program's total execution speed [1-4]. Therefore, code optimization approaches for efficient loop execution are critical. Loop unrolling is one such technique, wherein the body of the loop repeats several times and the loop termination code is adjusted. This technique can speed up execution by lowering the number of branch instructions required upon completing the loop body [2, 5, 6]. Table 1 shows the loop unrolling technique that if the unrolling factor is set to 2, the loop unrolls. When the factor is set to 4, as illustrated in Table 2, the unrolling technique improves program speed by allowing many iterations to run concurrently.

Table 1. Original loop and the the result after unroll the loop with loop unroll factor set to 2

Before Loop Unroll	After Loop Unroll
For *(k=0; k<N; k++){* *b[k]=k+1;}*	For *(k=0; k<N; k+=2){* *b[k]=k+1;* *b[k+1]=(k+1)+1;}*

Table 2. Unroll factor set to 4

After Loop Unroll
For *(k=0; k<N; k+=4){* *b[k]=k+1;* *b[k+1]=(k+1) + 1;* *b[k+2]=(k+2) + 1;* *b[k+3]=(k+3) + 1;}*

Probably a particularly essential aspect of loop unrolling is the capability to show instruction-level parallelism (ILP) to the compiler. Unrolling loops enables the compiler to rearrange activities within the expanded loop body to achieve iteration overlap [7, 8]. Bulldog is the compiler that used this approach for the first time and is still required when compiling on computers with a high level of ILP [9]. The unrolling approach, when used with additional transformation passes, will expand the size of the scheduled window. Similar techniques comprise trace scheduling and hyperblock generation.

These methods are especially beneficial for scheduling loops with control flow or function calls, which pose significant challenges to software pipelining [9].

Loop unrolling is critical for various optimizations, particularly those focused on improving the memory system. Moreover, the process of loop unrolling creates numerous static memory instructions that can be rearranged to benefit

from memory locality. In practical use, unrolling the loop leads to performance gains in the majority of instances when it is used. However, if performed poorly, this technique might interfere with other vital optimizations, slowing down overall performance. Therefore, selecting the suitable unrolling factor is critical. In spite of the unrolling technique having many benefits, possible drawbacks should also be considered.

-Unrolling has a well-known drawback of reducing instruction Cache speed.

-Increased scheduling flexibility can result in longer variable live ranges, increasing register pressure [10-14].

Machine learning is an important tool in artificial intelligence that improves program performance via experience. However, a single machine-learning model can often suffer from issues such as overfitting, which occurs when the model performs well on training data but badly on unseen data. Furthermore, certain models may be sensitive to noise in the data or fail to reflect the entire complexity of the problem. To solve these constraints, we introduce ensemble learning, which combines different models' predictions to produce a more accurate and robust system. Ensemble approaches decrease mistakes, increase generalization, and deliver more trustworthy predictions by combining the capabilities of various models [15]. The contribution of this paper is an investigation into the use of ensemble learning techniques with program dynamic features. Additionally, this research seeks to develop a model capable of predicting the ideal unrolling factor using ensemble learning. Three ensemble-learning techniques are used to train our dataset, which are Bagging, XGBoosting, and Random Forest. The remainder of the paper organizes as follows: Section 2 provides an overview of the related studies on loop unrolling. Section 3 covers three ensemble-learning approaches are introduced in this study. Section 4 describes the proposed approach. Section 4 includes the findings from benchmark programs. Finally, Section 5 summarizes our findings and offers concluding observations.

2. RELATED WORK

Some researchers have pointed to loop unrolling as an established approach for reducing loop overhead. This approach was frequently employed to enhance average-case speed in code by duplicating statements within the loop body, which causes fewer loop iterations, lower jump overhead, and less branching. Loop unrolling additionally expands the size of the basic block, making scheduling more efficient.

Stephenson and Amarasinghe [9] concentrated on loop unrolling, an important optimization approach for revealing instruction-level parallelism. Employing the Open Research Compiler as a platform, they show how to use supervised learning approaches to assess the suitability of loop unrolling. They use almost 2,500 loops from 72 benchmarks to train two separate learning algorithms to estimate unroll factors (i.e., how much a loop should be unrolled) for each new loop. The approach accurately estimates the unroll factor for sixty-five percent of the loops in the dataset, resulting in a five percent overall improvement for the SPEC 2000 benchmark suite (9 percent for the floating-point benchmarks).

Booshehri et al. [14] investigated the impact of loop unrolling on consumption of power, consumption of energy, and program speed by utilizing instruction-level parallelism (ILP). They investigated the concept of extended loop unrolling and presented a new method for traversing linked lists to improve loop-unrolling results. Their research carried out using a Pentium 4 CPU, which represents a superscalar design, which as well as a supercomputer outfitted with superscalar node computers. The studies revealed that, while loop unrolling has little effect on both power and energy utilization, it can be a useful strategy for speeding up applications.

As mentioned in the study [11], the machine learning model improve the unrolling capabilities by anticipating its factor. Initially, they enhance a basic Random Forest model by applying weighting and the imbalanced dataset. After creating the training set, they train the model. Experimental findings show that the model can accurately estimate the optimal or suboptimal unrolling factor 81% of the time. The model has also been tested on numerous SPEC2006 test sets. While Open64's built-in loop unrolling model increases program performance by an average of 5%, the method suggested in this study, which predicts the factors of unrolling through applying a weighted decision forest, improves the performance of program by approximately 12%.

To improve the precision of the compiler's loop unrolling factor, Singh et al. [1] suggested an improved loop unrolling technique that utilizes a modified random choice forest. Initially, the standard Random Forest improved by including weight values. Second, they addressed the issue of unbalanced datasets using a BSC technique based on the SMOTE algorithm. Nearly 1,000 loops were selected from different benchmarks, and the characteristics extracted from them served as the training set for the suggested method to anticipate the unroll factor. The model predicted the unrolling factor with 81% precision, compared to 36% for the current Open64 compiler.

3. ENSEMBLE LEARNER (EL)

Ensemble Learning is a subset of Machine Learning that seeks to improve task performance, such as classification and regression, by training a group of relatively weak learners and integrating their results using voting or averaging. Despite their particular weaknesses, these learners can create strong results when they work together. The ensemble method's effectiveness is intuitive: a group of learners, each good in a specific work, can complement one another. Their collaboration frequently results in superior overall performance than a single learner could achieve alone.

Below we are going to present three popular ensemble methods named Bagging, Boosting, and Random Forest. These methods employ resampling algorithms to generate distinct training sets for each classifier [15].

3.1 Random Forest (RF)

Random Forest, the most popular ensemble method produced, was introduced separately by the studies [16, 17] around that exact time. It is becoming increasingly popular, thanks to their flexibility and predictive effectiveness. Furthermore, RF is regarded as a simple way to adjust in comparison to other systems that necessitate precise tuning. It consists of a large number of decision trees that operate independently to estimate the outcome of a class, with the ultimate prediction determined by their majority vote. Figure 1 illustrates how the separate trees are built.

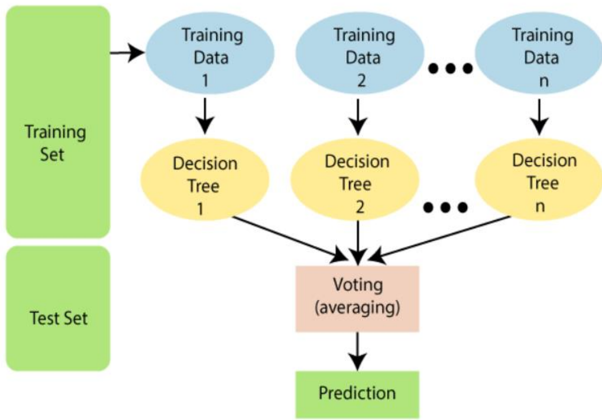

Figure 1. Random Forest

3.2 Bagging

Bagging [18] is a straightforward but efficient approach for creating an ensemble of independent models. This strategy envolves training each model on a portion of the main dataset's occurrences. To guarantee that each model has enough instances, these samples are typically the same size as the main dataset. The ultimate anticipate for an unseen occurrence is chosen by a majority vote on the models' projections. Because sampling uses replacement, certain instances from the original dataset may appear several times in a sample, while others may be excluded entirely. Because the models are trained independently, Bagging can be done in parallel, with each model trained on a separate processing unit.

3.3 Extreme gradient boosting (XGBoost)

The XGBoost algorithm, presented by Chen and Guestrin [19], was created to break the computational limits of boosting trees, resulting in quick computation and superior performance. XGBoost brings various advancements over classic gradient boosting algorithms and is known for its exceptional performance in both classification and regression problems. In XGBoost, the prediction for a given sample is the sum of the leaf weights from each weak classifier. A fundamental feature in XGBoost is the introduction of a regularization term to limit tree models' inherent tendency to overfit, allowing for little overfitting before pruning. Furthermore, XGBoost balances model performance and computation speed using an objective function that combines a standard loss function with a regularization term to manage model complexity [20].

4. PROPOSED METHOD

Figure 2 depicts the suggested model, which consists of four steps.

4.1 Constructing dataset

For this step, we gathered more than 220 samples—drawn from 20 benchmark programs with different problem sizes. The execution time of each program was calculated using a variety of loop unroll factors. Four unroll factors—2, 4, 6, and 8—were chosen based on prior testing showing a considerable impact on program performance. Each program was executed at least five times for each unroll factor, implying that the same program was run five times with the same unroll factor, and the average execution time was computed. The unroll factor (2, 4, 6, or 8) with the minimum execution time was chosen as the optimal factor for the program. To unroll the loop, two LLVM optimization passes were used: --loop-unroll and --unroll-count.

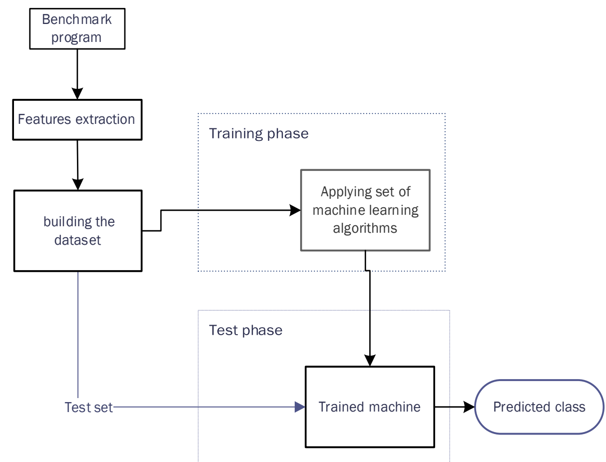

Figure 2. The suggested approaches

Features extraction

The dynamic dataset includes features that vary during program execution. These dynamic characteristics are obtained using the Linux 'perf' tool, which provides an empirical picture of the program's dynamic behavior as it interacts with the computing machine while running. For each program, 35 dynamic features are collected. Table 3 shows the dtnamic features [21, 22].

Table 3. Perf event

Event	Type
instructions,LLC-loads,LLC-load-misses,L1-dcache- loads, L1-dcache-load-mises, cache-misses, cache-refernces, dTLB-loads,LLC-stores,LLC-store-misses,dTLB-loads,dTLB-load-miss, iTLB-loads, iTLB-load-miss, dTLB-store,dTLB-store-misses,itlb_misses.walk_completed,branch-instructions,branch-misses,L1-dcache-stores-misses,L1-dcache-stores,cpu-cycles,bus-cycles,ref-cycles,page-faults,context-switches,cpu-migrations,minor-faults,major-faults,alignment-faults,emulation-faults,cpu-clock,task-clock,mem-loads,mem-stores	Hardware & software events

4.2 Features preprocessing

In the field of machine learning, data preparation is an important step before modeling.

A the first, the columns that have the zero value are removed. After this the left features become 29. Then the standardization is applied, specifically the MinMax technique, which is used to calibrate the range of feature values, assuring scale consistency. The MinMax method is very useful for converting data to a bounded interval, often [0, 1]. The mathematical formula for this operation is as follows:

$$Xscaled = \frac{X-Xmin}{Xmax-Xmin} \quad (1)$$

where, $Xmin$ and $Xmax$ indicate the minimum and maximum values of feature X, respectively.Thisrecalibration is required to guarantee that each feature contributes equally to the analytical results. Algorithm 1 show the steps of building the dataset.

Algorithm 1 Building Dataset

Input:set of programs Pro[M] , set of loop unroll K=[2,4,6,8]
Output: set of program M with less execution time
Begin
for j =1 to M
X=Pro[j]
Execute each program X will all loop unroll factor K
Chose the one with less execution time.
Add this program X to the Dataset with its loop unroll factor
Extract the dynamic fatures for X program using Perf tool
Features preprocessing.
Add this features to the Dataset.
End
End

As a consequence, our dataset has 31 columns. The first column contains the program name. The program features take up columns 2 through 30, while the label (the optimal unroll factor for this program) occupies column 31.

4.3 Classifiers

Set of machine learning techniques are use to classify our data set. These are RF, Bagging and XGBossting. These are ensemble methods that combine the predictions of multiple base estimators to improve overall performance. Firstly we train our dataset with all the ensemble learning techniques. Then we compare their outcomes to determine the best one.

5. EXPERMENTAL RESULTS

In this research, we use Google Colab, an online platform that provides a Jupyter notebook environment, to analyze data and execute ensemble learning models. The platform's smooth interface with Python, as well as its robust computational capabilities, enabled us to effectively process our dataset and train our models. The results from training and testing each of the proposed learning models discussed above are shown. The classification models are trained with various parametrs, and the results are very promising.

5.1 Training set programs

The constructed dataset is used to train three different EL models that estimate the program's time efficiency. The present research focuses on three ensemble learning models, as stated in Section 3. These classifiers take 29 dynamic characteristics as input, and the output is one of the loop unroll factors (2, 4, 6, and 8). Moreover, a result of the small size of the datasets, k-fold cross-validation is preferable to a separate training and testing split since it maximizes data consumption by using every data point for training and validation. This approach, by averaging findings across numerous folds, provides a more reliable performance estimate while lowering the risk of overfitting.

5.1.1 Bagging classifier results

The dataset is loaded into a pandas DataFrame, with all column names being strings. The data is then divided into two categories: features (X) and labels (y), with the first 29 columns serving as features and the 30th column as the label. A Bagging classifier is built with a Decision Tree as the base estimator and ten decision trees (n_estimators=12). A 15-fold stratified cross-validation is used to ensure that each fold has a proportional representation of the classes. Log loss is determined for each fold to determine how closely the projected probabilities match the true labels, with smaller log loss indicating higher predictive accuracy.

Mean accuracy with 15-fold cross-validation: 98.63% while the mean log loss with 15-fold cross-validation: 0.2960 as shown in Figure 3.

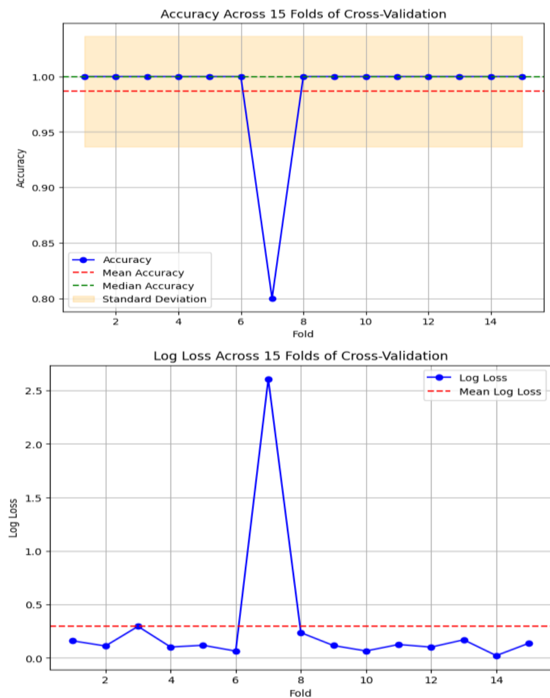

Figure 3. The accuracy of 15 folds of cross validation and loss function of Bagging classifier

5.1.2 Random Forest classifier results

We employed a Random Forest classifier for this research. The model was configured with a fixed random state (random_state=18) to ensure reproducibility of results.To evaluate the model, we used Stratified K-Fold Cross-Validation with 15 folds. This method ensures that each fold has the same proportion of class labels as the entire dataset, which is crucial for maintaining the balance in class distribution across folds. Mean accuracy with 15-fold cross-validation: 99.56% while the mean log loss with 15-fold cross-validation: 0.1755 as shown with Figure 4.

5.1.3 XGBoost classifier results

XGBClassifier use the default parameters for XGBClassifier as of the latest version of the XGBoost library. XGBoost Classifier is built with **(n_estimators=36)** which is the number of gradient boosted trees. The performance of an XGBoost classifier on a dataset using 15-fold cross-validation is evaluated. Mean accuracy with 15-fold cross-validation: 99.567% while the mean log loss with 15-fold cross-validation is 0.0453 as shown in Figure 5.

Log loss is determined for each of three classifiers to determine how closely the projected probabilities match the true labels, with smaller log loss indicating higher predictive

accuracy. Additionally, precision, recall, F1 score, and standard deviation metrics are computed using 15-fold cross-validation. Table 4 presents the results for all metrics.

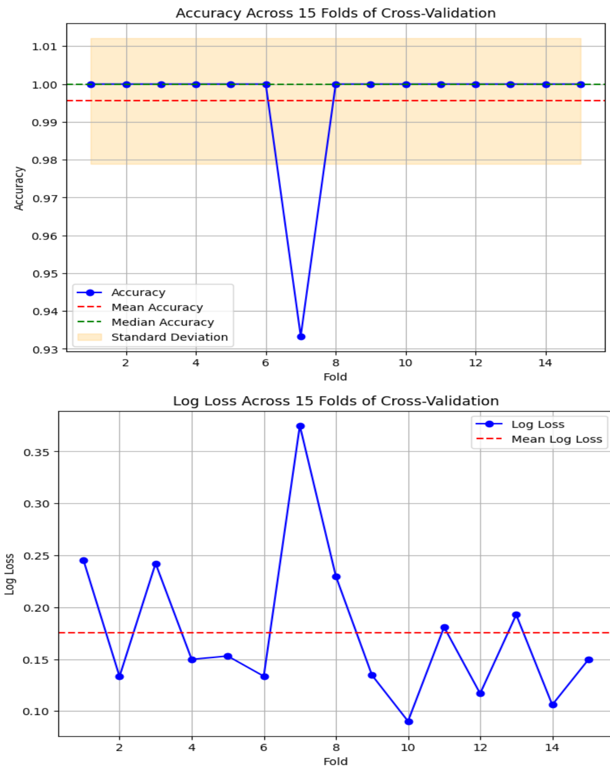

Figure 4. The accuracy of 15 folds of cross validation and loss function of Random Forest classifier

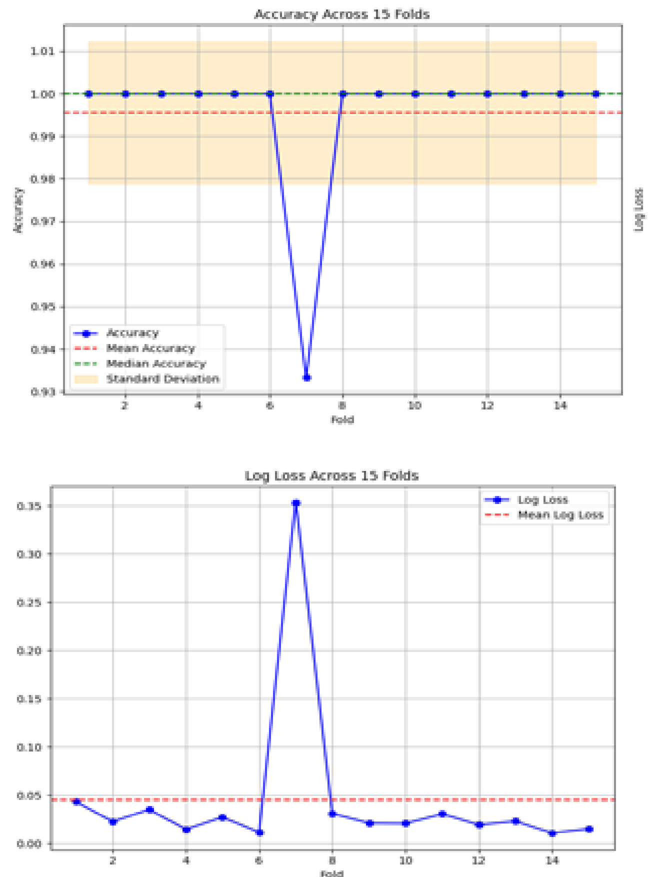

Figure 5. The accuracy of 15 folds of cross validation and loss function of XGBoost classifier

Table 4. Various evaluation metrics

Classifier	Mean Precision	Mean Recall	Mean F1 Score	Mean Accuracy	Mean Log Loss	SD
Bagging	99.29%	98.40%	98.60%	98.63%	0.2960	0.027
Random Forest	99.78%	99.33%	99.43%	99.56%	0.1755	0.016
XGBoost	99.56%	99.33%	99.29%	99.56%	0.0453	0.016

Moreover, we experimented with three different k-fold cross-validation settings: 5-fold, 10-fold, and 15-fold. The best results were obtained with 15-fold cross-validation as present in Table 5.

Table 5. Different k-fold cross validation

Classifier	5-fold	10-fold	15-fold
Bagging	97.73%	98.18%	98.63%
Random Forest	99.55%	99.55%	99.56%
XGBoost	99.55%	99.55%	99.56%

5.2 Analysis the findings

While building the dataset, we noted several key observations. First, the investigation revealed that, through using LLVM opt with the loop unroll optimization step, around half of the benchmark loops could not be unrolled Because of one or more of the following factors:

i. a low initial value for the loop induction variable and ii. the utilization of conditional control inside the loop.

We also found that an unroll factor equal to 8 is the highest value that effectively speeds up program execution time. Values greater than eight may maintain the same execution time or worsen program performance.

The experiments also show that the loops with an efficient unroll factor, for instance 8, often perform satisfactorily with a smaller one, like 2, though the reverse isn't always accurate. Furthermore, no single unroll factor is constantly dominant and performing well across all loops.

6. CONCLUSIONS

This research addressed the difficulty of recognizing the ideal unroll factor for a set of programs in order to increase its performance. We provided a method for identifying the most effective loop unroll factor by using multiple ensemble learning models such as Bagging, Random Forest, and XGBoost. We evaluated these models on the construction dataset. The dataset was comprised of a number of dynamic features of a set of programs with best loop unroll factors. XGBoost and RF outperformed the third model that trained on the dataset, predicting the most appropriate unroll factor with a precision rate of 99.56%.

A key limitation of the study is the lack of a sufficiently large dataset, which can introduce biases and affect the generalizability of the results. Small datasets may not fully

capture the variability or diversity present in the population. This can also result in biased performance estimates, as the model may learn noise or specific patterns that are not representative of the broader problem. In the future, extending the dataset and training classification models on it will help increase the model's ability for generalization.

REFERENCES

[1] Singh, I., Singh, S.K., Singh, R., Kumar, S. (2022). Efficient loop unrolling factor prediction algorithm using machine learning models. In 2022 3rd International Conference for Emerging Technology (INCET), pp. 1-8. https://doi.org/10.1109/INCET54531.2022.9825092

[2] Almohammed, M.H., Fanfakh, A.B., Alwan, E.H. (2020). Parallel genetic algorithm for optimizing compiler sequences ordering. In New Trends in Information and Communications Technology Applications: 4th International Conference, NTICT 2020, Baghdad, Iraq, June 15, 2020, Proceedings 4, pp. 128-138. https://doi.org/10.1007/978-3-030-55340-1_9

[3] Alwan, E.H. (2023). Predicting loop vectorization through machine learning algorithms. Journal of Fusion: Practice and Applications, 15(2): 36-45. https://doi.org/10.54216/FPA.150203

[4] Ahmed, N.S., Alwan, E.H., Fanfakh, A.B. (2024). Optimizing loop tiling in computing systems through ensemble machine learning techniques. Full Length Article, 15(1): 214-14. https://doi.org/10.54216/FPA.150117

[5] Zacharopoulos, G., Barbon, A., Ansaloni, G., Pozzi, L. (2018). Machine learning approach for loop unrolling factor prediction in high level synthesis. In 2018 International Conference on High Performance Computing & Simulation (HPCS), pp. 91-97. https://doi.org/10.1109/HPCS.2018.00030

[6] Panda, P.R., Sharma, N., Kurra, S., Bhartia, K.A., Singh, N.K. (2018). Exploration of loop unroll factors in high level synthesis. In 2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), pp. 465-466. https://doi.org/10.1109/VLSID.2018.115

[7] Huang, J.C., Leng, T. (1999). Generalized loop-unrolling: A method for program speedup. In Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No. PR00122), pp. 244-248. https://doi.org/10.1109/ASSET.1999.756775

[8] Domagała, Ł., van Amstel, D., Rastello, F., Sadayappan, P. (2016). Register allocation and promotion through combined instruction scheduling and loop unrolling. In Proceedings of the 25th International Conference on Compiler Construction, pp. 143-151. https://doi.org/10.1145/2892208.289221

[9] Stephenson, M., Amarasinghe, S. (2005). Predicting unroll factors using supervised classification. In International symposium on code generation and optimization, pp. 123-134. https://doi.org/10.1109/CGO.2005.29

[10] Murtovi, A., Georgakoudis, G., Parasyris, K., Liao, C., Laguna, I., Steffen, B. (2024). Enhancing performance through control-flow unmerging and loop unrolling on GPUs. In 2024 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pp. 106-118. https://doi.org/10.1109/CGO57630.2024.10444819

[11] Liu, H., Guo, Z. (2018). A loop unrolling method based on machine learning. Vibroengineering Procedia, 18: 215-221. https://doi.org/10.21595/vp.2018.19928

[12] Hall, M., Chame, J., Chen, C., Shin, J., Rudy, G., Khan, M.M. (2010). Loop transformation recipes for code generation and auto-tuning. In Languages and Compilers for Parallel Computing: 22nd International Workshop, LCPC 2009, Newark, DE, USA, Revised Selected Papers 22, pp. 50-64. https://doi.org/10.1007/978-3-642-13374-9_4

[13] Carminati, A., Starke, R.A., de Oliveira, R.S. (2017). Combining loop unrolling strategies and code predication to reduce the worst-case execution time of real-time software. Applied Computing and Informatics, 13(2): 184-193. https://doi.org/10.1016/j.aci.2017.03.002

[14] Booshehri, M., Malekpour, A., Luksch, P. (2013). An improving method for loop unrolling. arXiv preprint arXiv:1308.0698. https://doi.org/10.48550/arXiv.1308.0698

[15] Huang, D., Hu, D., He, J., Xiong, Y. (2018). Structure damage detection based on ensemble learning. In 2018 9th International Conference on Mechanical and Aerospace Engineering (ICMAE), pp. 219-224. https://doi.org/10.1109/ICMAE.2018.8467650

[16] Ho, T.K. (1995). Random decision forests. In Proceedings of 3rd International Conference on Document Analysis and Recognition, pp. 278-282. https://doi.org/10.1109/ICDAR.1995.598994

[17] Amit, Y., Geman, D. (1994). Randomized inquiries about shape; an application to handwritten digit recognition. Dept. Statistics, Univ. of Chicago, Technical Report, 401.

[18] Mienye, I.D., Sun, Y. (2022). A survey of ensemble learning: Concepts, algorithms, applications, and prospects. IEEE Access, 10: 99129-99149. https://doi.org/10.1109/ACCESS.2022.3207287.

[19] Chen, T., Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785-794. https://doi.org/10.1145/2939672.2939785

[20] Zhen, J., Mao, D., Shen, Z., Zhao, D., Xu, Y., Wang, J., Jia, M., Wang, Z., Ren, C. (2024). Performance of XGBoost ensemble learning algorithm for mangrove species classification with multisource spaceborne remote sensing data. Journal of Remote Sensing, 4: 0146. https://doi.org/10.34133/remotesensing.0146

[21] Alhasnawy, L.H., Alwan, E.H., Fanfakh, A.B. (2020). Using machine learning to predict the sequences of optimization passes. In New Trends in Information and Communications Technology Applications: 4th International Conference, NTICT 2020, Baghdad, Iraq, June 15, 2020, Proceedings 4, pp. 139-156). https://doi.org/10.1007/978-3-030-55340-1_10

[22] Almohammed, M.H., Alwan, E.H., Fanfakh, A.B. (2020). Programs features clustering to find optimization sequence using genetic algorithm. In Intelligent Computing Paradigm and Cutting-edge Technologies: Proceedings of the First International Conference on Innovative Computing and Cutting-edge Technologies (ICICCT 2019), Istanbul, Turkey, pp. 40-50.

https://doi.org/10.1007/978-3-030-38501-9_4

NOMENCLATURE

EL	Ensemble Learning
RF	Random Forest
XGBoost	Extreme Gradient Boosting
LLVM	Low Level Virtual Machine
Bagging	Bootstrap aggregating
Perf	CPU performance counters
ILP	Instruction level parallelism