Enhancing Programs Efficiency through a Machine Learning-Based Model for Tile Size Selection

NoorUlhuda S. Ahmed*[1,2], *Esraa H.* Alwan[1] and *Ahmed B. M.* Fanfakh[1]

[1] Department of Computer Science, College of science for women, University of Babylon, Babil, Iraq
[2] College of Medicine, University of Al-Ameed, Karbala PO Box 198, Iraq

Abstract. This work presents an innovative machine-learning approach to improve loop tiling in computational programming. Loop tiling is a crucial strategy for boosting speed by promoting data locality and reducing cache misses. Conventional methods frequently have difficulties in accurately calculating the most suitable tile size, which is a crucial component impacting the performance of the program. The combination of Multi-Output Generalized Regression Neural Networks (MOGRNN) and Linear Regression in researchers' techniques allows for precise prediction of optimal tile sizes for various computing workloads. The study entails an extensive gathering of data from 22 benchmark programs, which encompass a diverse set of computational patterns and issue sizes. This data collection is further enhanced by incorporating both static and dynamic program aspects. By employing meticulous preprocessing and doing dual-model analysis, the researchers' approach effectively captures both linear and intricate non-linear correlations present in the data. The method's usefulness in boosting prediction accuracy for ideal tile sizes and enhancing overall program performance has been demonstrated through extensive testing on an Intel Core i7 CPU. This novel approach provides a viable avenue for advanced study in code optimization approaches.

1 Introduction

Loop tiling, sometimes referred to as code tiling or loop transformation, is a well-established method that plays a crucial role in optimizing code. The importance of this resides in improving the proximity of data and promoting large-scale parallelism, significantly increasing the performance of the code [1]. The core concept of loop tiling is breaking down a loop's range of iterations into smaller, rectangular parts known as tiles. The careful choice of tile size is crucial, as it has a significant impact on the performance of the code that uses tiles. Loop tiling is especially advantageous in scientific computing as it enhances data localization, reduces memory access expenses, and boosts the

performance of numerical computations by leveraging hierarchical memory architectures [2, 3]. It utilizes the reuse of unchanging elements in outer loops and the reuse of elements in close proximity in inner loops, therefore optimizing the total execution of the program. Significantly, it excels in reducing cache misses and improving temporal cache locality, making it particularly useful for compute-intensive processes like matrix-matrix multiplication. However, the pursuit of the most suitable tile size is filled with difficulties and complexities. The efficiency of tiled code is significantly influenced by the choice of tile size. Tiles that are too tiny can cause additional processing and reduce the efficiency of other optimizations, whereas tiles that are overly big can result in conflicts in cache mapping and interference misses.

Furthermore, the effectiveness of tiling depends on how arrays are structured, making the sensitivity of tile factors an important element to consider in array organization [4]. Therefore, it is crucial to handle the challenge of selecting the tile size in order to achieve maximum performance optimization through tiling. This requires determining the optimal tile sizes that align with the parameters of the program, the memory hierarchy, and the number of processors. Nevertheless, the process of creating accurate models for selecting tile sizes is laborious and time-consuming, particularly when it comes to adapting to different architectures or keeping up with advancing compilers. These problems highlight the necessity for advanced research and the creation of sophisticated techniques to automate the process of selecting tile sizes. This will guarantee optimized code performance across computing systems that are diverse in nature [5] [6].

The primary focus of research in tile size selection (TSS) has been on analytical models that use static analysis of loop kernels and hardware features to identify the appropriate tile size. Although these models offer vital insights into the interaction between hardware and software, they demand significant human effort and may not consistently perform well across different applications and current CPUs. Researchers suggested technique utilizes machine learning (ML) to accurately forecast the most suitable tile dimensions. The methodology involves the utilization of two machine learning modules, specifically multi-output generalized regression neural networks (MOGRNN) and linear regression models.

* Corresponding author: noor.ahmed.gsci115@student.uobabylon.edu.iq

This approach utilizes a dual-module technique to capitalize on the intrinsic capabilities of each model, with the objective of producing tile size forecasts that are both more precise and resilient.

The subsequent sections of this paper are structured as follows: Section 2 explores the existing research and approaches in the field of TSS, showing various methods employed. Section 3 describes the optimal tile size method, providing in-depth insights into its underlying principles and algorithms. Section 4 shows the results of comprehensive experiments conducted to assess the effectiveness and efficiency of the researchers' proposed method. Finally, in Section 5, concludes by summarizing the contributions and benefits of the proposed approach and suggesting future enhancements in this domain.

2 Related Work

Extensive study has focused on selecting the most optimal tile size in loop kernels to improve performance by optimizing parallelism and data location. This section presents a comprehensive overview of the current research, encompassing both analytical models and machine learning-based approaches in this domain.

Analytical models have played a crucial role in forecasting the duration of execution and comprehending the cache behaviors of tiling loop kernels. An influential study outlined in [6] offered the first analytical framework for loop-tiling transformations. This model took into account factors such as loop limits, data dependencies, and cache dimensions in order to determine the most efficient tile size that reduces execution time. Later, [7] introduced an approach that focuses on analyzing loop iteration spaces and data access patterns, with the goal of reducing the working set size in tiled loop nests. These models provide deep insights into the impact of tile size on performance measures.

Researchers have customized analytical models to incorporate unique architectural intricacies in order to improve their accuracy. For example, [4] established the theoretical limits on the best tile sizes and examined how they affect the complexity of the cache. In addition, [7] explored the field of tensor contraction workloads and developed a model that effectively captures data reuse patterns and cache performance. This model helps in selecting optimal tile sizes. In addition, [8] developed sophisticated analytical models specifically designed for stencil calculations on GPUs. These models optimize the size of computation tiles by carefully analyzing the dynamics of computation and memory access. These techniques emphasize the need to incorporate hardware-specific characteristics into the process of selecting tile sizes.

Although analytical models have greatly enhanced researchers' comprehension, their dependence on simple assumptions often made them insufficient for capturing the complex behaviour of actual systems. To address this constraint, machine learning methods have emerged as an appealing substitute, providing more sophisticated and precise cost models. Recent advancements in program optimization have utilized deep learning techniques to understand the intricacies involved in adjusting the size of tiles, which is a critical factor in determining program success [5]. This innovative approach utilizes large datasets consisting of program instances to train deep learning models, thereby enabling machine learning to decipher the intricate relationship between program attributes and their ideal tile sizes.

Researchers have made substantial advancements in the domain of tile size selection by introducing machine learning models that can predict program execution durations based on different tile sizes and hardware combinations [9]. By utilizing past performance data, these models have demonstrated exceptional prediction accuracy, making the process of selecting the best tile sizes more efficient and enhancing the overall effectiveness of the optimization effort. Moreover, the task of choosing the size of tiles has been reconceptualized as an optimization issue, with machine learning techniques being skillfully utilized to address it [1]. These projects, based on machine learning principles, highlight the significant powers of data-driven methods in the context of optimizing tile size.

Nevertheless, although these traditional approaches may be efficient in specific situations, they often lack the necessary adaptability and accuracy, especially when confronted with the complexities inherent in distinct computing environments and different program architectures. The main limitation of these conventional methods is their fixed character, which makes them less able to adjust dynamically to the changing behaviours of programs and the intricacies of various hardware setups. As mentioned in reference [18], conventional loop tiling methods focus on enhancing data locality and cache efficiency, but they typically overlook the dynamic features of program execution. This lack of attention might result in less-than-ideal performance in some situations. In addition, many solutions that rely on heuristics need a significant level of skill and can be time-consuming, which restricts their practicality in computer systems that are constantly evolving.

Recent research has thoroughly proven the efficacy of ensemble machine-learning algorithms in improving prediction accuracy and resilience. Exemplary research conducted by [115] illustrates this pattern. This study utilized an enhanced ensemble model, which merged Logistic Regression and Extra Tree classifier, to address the limitations of individual machine learning methods, resulting in a notable advancement in software fault prediction. Another significant work [17] investigates the use of ensemble methods in software defect prediction. It emphasizes the effectiveness of these techniques in improving prediction performance, which aligns with the goals of our methodology.

Furthermore, a multitude of academic endeavors has concentrated on tailoring the choice of tile size to fit certain applications or specific algorithmic domains. [10] investigated the extent to which iterative polyhedral compilation techniques may be used to achieve scalable tile size optimization. This technology has demonstrated its adaptability across many architectural environments through an iterative refinement process of the tile size parameter. Simultaneously, a separate investigation conducted by [11] thoroughly analyzed the impact of tile size on the performance metrics of sparse

matrix multiplication, recommending customized approaches to optimize tile sizes within this specific domain. This area of investigation emphasizes the many factors that must be taken into account when choosing the size of tiles to suit unique problems in different circumstances.

[11] conducted a quantitative assessment of analytical methods for selecting tile sizes, with the goal of determining their usefulness. This comparison research serves as a great tool for identifying the comparative strengths of different algorithms and explaining how they might be used in certain computing situations. Together, these study efforts emphasize the crucial significance of choosing the appropriate tile size as a fundamental aspect of program optimization. They shed more light on the potential of machine-learning approaches and data-driven tactics in improving and optimizing this crucial process. Furthermore, doing focused research that concentrates on certain application areas provides valuable knowledge on how to optimize the dimensions of tiles specifically designed for particular issue scenarios.

3 Proposed Method

This paper introduces a novel approach to improve loop tiling in computer systems through the utilization of ensemble machine-learning techniques. The objective of this technique is to accurately predict the optimal tile size for loop structures in programming, with the goal of enhancing cache efficiency and overall system performance. The advancement of this paper's approach comprises several crucial phases, each important for achieving the objective. A comprehensive understanding of these stages can be attained by consulting a full flowchart diagram shown later in this document. The flowchart picture below provides a visual representation of the sequential phases and step-by-step techniques of this paper's systematic approach.

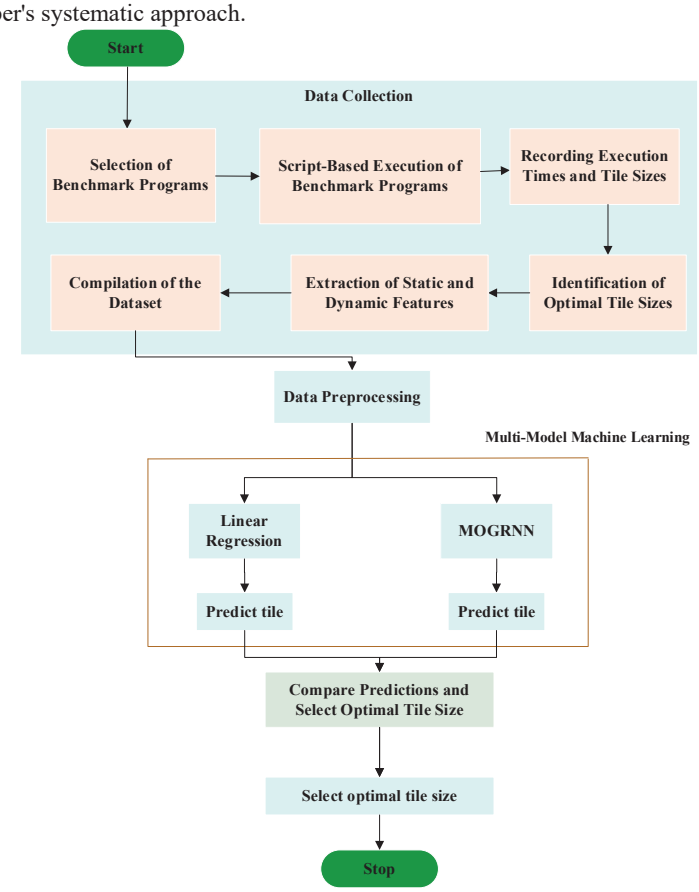

Fig.1 Propose method

Figure 1 illustrates the step-by-step progression of this paper's proposed technique. It visually demonstrates the process of collecting data and then applying advanced machine-learning techniques, which finally results in the practical assessment of these models. The flowchart offers a transparent and comprehensive depiction of the intricate procedures entailed in the investigation. This paper's emphasis lies in using a methodical methodology to spearhead advancements in computer system efficiency.

The suggested technique is organized into many essential stages:

3.1 Data Collection

• Benchmark Program Selection

First step involves the careful selection and arrangement of 22 benchmark programs from the PLUTO benchmark suite. These selections have been meticulously curated to include a wide range of computational patterns and issue sizes. This selection ensures that dataset [16] encompasses a wide range of loop-tiling options.

Table 1. PLUTO benchmark

Kernel	Problem Size	Description
Corcol	300;400;500;600;100;1500;1800;2000;2200;2500;3000	A program used for computing correlation coefficients, standard in statistical analysis applications.
Gemver	300;400;500;600;100;1500;1800;2000;2200;2500;3000	A kernel that performs vector multiplication and matrix addition, typical in linear algebra operations.
MVT	300;400;500;600;100;1500;1800;2000;2200;2500;3000	Matrix-vector product and Transposition are used in various numerical computations.
Doitgen	300;400;500;600;100;1500;1800;2000;2200;2500	A benchmark involved in the multi-dimensional array computation relating to tensor contraction.
FDTD-1D	300;400;500;600;100;1500;1800;2000;2200;2500	One-dimensional Finite-Difference Time-Domain simulation, used in electromagnetic wave propagation studies.
FDTD-2D	300;400;500;600;100;1500;1800;2000;2200;2500;3000	Two-dimensional extension of FDTD-1D for more complex electromagnetic simulations.
Floyd	300;400;500;600;100;1500;1800;2000;2200;2500;3000	Implements Floyd's algorithm for finding shortest paths in a graph, used in network analysis.
Seidel	300;400;500;600;100;1500;1800;2000;2200;2500;3000	A 2D stencil code for solving partial differential equations using the Seidel method.
Jacob_1D	300;400;500;600;100;1500;1800;2000;2200;2500;3000	The one-dimensional Jacobi method is used for solving systems of linear equations.
Jacob_2D	300;400;500;600;100;1500;1800;2000;2200;2500;3000	A two-dimensional extension of the Jacobi method for more complex systems.
Matmul_int	300;400;500;600;100;1500;1800;2000;2200;2500;3000	Integer matrix multiplication is a fundamental operation in many scientific computations.
Advect3d	300;400;500;600;100;1500;1800;2000;2200;2500;3000	A program for simulating three-dimensional advection, common in fluid dynamics.
Matmul	1000;3000;5000;8000;8500;8800;9000;9200;9500;9800;10000	Floating-point matrix multiplication is crucial in high-performance computing tasks.
Dsyrk	15,25,35,45,55,65,75,85,95,110,130,150	A kernel performing symmetric rank-k operations is used in various mathematical computations.
Dsyr2k	300;400;500;600;100;1500;1800;2000;2200;2500;3000	Similar to Dsyrk, it performs symmetric rank-2k operations.
TMM	300;400;500;600;100;1500;1800;2000;2200;2500;3000	Transpose matrix multiplication is used in graph algorithms and network analysis.
Trisolv	300;400;500;600;100;1500;1800;2000;2200;2500;3000	A program to solve triangular systems, a common problem in numerical methods.
SSYMM	300;400;500;600;100;1500;1800;2000;2200;2500;3000	Symmetric matrix-matrix multiplication is used in physics simulations and computer graphics.
STRMM	3000;4000;5000;6000;10000;15000;18000;20000;22000;25000;28000;30000	Triangular matrix-matrix multiplication is employed in solving linear algebra problems.
STRSM	300;400;500;600;100;1500;1800;2000;2200;2500;3000	Solves triangular matrix problems with multiple right-hand sides, used in advanced linear algebra solutions.
DCT	1024	Discrete Cosine Transform is widely used in image processing and signal compression.
Covcol	1024	A kernel for covariance computation, often used in data mining and pattern recognition.

The table provides a concise summary of each benchmark program selected for the study, highlighting their importance and extensive utilization in computational applications. These explanations provide useful insights into the breadth and intricacy of the difficulties addressed by the PLUTO benchmark set.

• Execution of benchmark programs using scripts

A tailored script is implemented for each chosen application to automate the program's execution across various tile sizes. This script is meticulously designed to systematically alter the dimensions of tiles within a predetermined range while recording the duration of each distinct tile configuration. A complete performance profile is created for each program by duplicating the execution technique for twelve distinct issue sizes corresponding to each program in order to analyze performance under various conditions.

• Collecting data on the duration of executions and the dimensions of tiles

The outcomes of every script execution are regularly documented in a dedicated file named "times." This file is essential since it consolidates the execution times associated with each tile size for every benchmark program. This resource is

crucial for conducting more studies, as it enables us to ascertain the tile size that yields the minimum execution time. Opting for the optimal tile size from this dataset is of utmost importance, as it guides the selection procedure for the target variable of the machine-learning model.

• Determining Optimal Tile Sizes

Considering various problem sizes, comprehensively analyse the "times" file to identify the most appropriate tile size for each benchmark program. The optimality criteria are established based on the lowest recorded execution time under particular conditions. The optimal dimensions of the tiles are of utmost importance since they serve as the desired variables for machine-learning models. The models will strive to forecast these sizes, providing a standard for evaluating the precision of the method.

• Extraction of Static and Dynamic Features

To extract the static features, the programs must be stored in bit-code file format (.bc). Then the LLVM "-inscount" pass is applied to extract them. Overall, these features describe the static behavior of the program. Table 1. illustrates the LLVM static features. The total number of them equals 39 features distributed among different programs. Each one has its own numerical value, which varies from one program to another, as shown in Table 1 [12] [13].

Table 2. List of static features of programs

Add instructions, FAdd instructions	GetElementPtr instructions, Ret instructions	SRem instructions	ZExt instructions basic blocks
Alloca instructions, FCmp instructions	ICmp instructions, SDiv instructions	Shl instructions	
And instructions, FDiv instructions	Load instructions, Sub instructions	Store instructions	Memory instructions non-external functions
AShr instructions, FMul instructions	Mul instructions, Switch instructions	Trunc instructions	
BitCast instructions, FPExt instructions	or instructions, SExt instructions	URem instructions	
Br instructions, FPToSI Instructions	PHI instructions, Select instructions	Unreachable instructions	
Call instructions, FSub instructions	PtrToInt instructions, SIToFP Instructions	Xor instructions	

To capture the dynamic behavior of the benchmark programs, The Perf tool collects hardware performance counter data throughout the execution of benchmark programs with varying tile sizes.. The dynamic features include cache misses, cache hits, and other relevant hardware performance metrics. These dynamic features enable us to understand the cache behavior and the effectiveness of different tile sizes in reducing cache misses. Table 2 below illustrates the dynamic features:

Table 3. List of dynamic features

Event	**Type**
Cpu-cycles or cycles, instructions, Cache- references, Cache-misses, Bus_cycles	Hardware event
Cpu_clock(msec), Task_clock(msec), Page- faults OR faults, Context-switches, Cpumigrations, Alignment-faults, Emulationfaults	Software event
L1-dcache-loads, L1-dcache-loads misses, L1-dcache-stores, L1-dcache-storesmisses, L1-icache-loads, L1-icache-loadsmisses, L1-icache-loads, L1-icache-loads misses, L1-icache- prefetches, L1-icacheprefetches –misses, LLC-load, LLC-loadsmisses, LLC-strores, LLC-strores misses, LLC-prefetch-misses, Dtlb-loads, Dtlb-loadsmisses, Dtlb-store, Dtlb-store-misses, Dtlb prefetches, Dtlb-prefetches -misses, Itlb -loads, Itlb –loads -misses, Branch-loads, Branch-loads-misses	Hardware cache event
Sched:sched-stat-runtime, Sched:sched-pi-setprio, Syscalls:sys_enter_socket, Kvmmmu:kvm_mmu_pagetable_walk	Tracepoint event
Stalled_cycles-frontend, Stalled_cycle- backend	Dynamic event

• Dataset	Sched:sched_process_exec, Sched:sched_process_frok, Sched:sched_process_wait,Sched:sched_process_wait_task, Sched:sched_process_exit	Tracepoint event

Compilation: The culmination of the data preparation process involves merging the meticulously recorded execution durations, establishing optimal tile sizes, and comprehensive collection of static and dynamic attributes for each program across all evaluated problem sizes. The dataset is structured to suit the unique requirements of the machine-learning system. The predictors are represented by the characteristics, while the optimal tile sizes determine the response variables. This comprehensive dataset includes the essential features of the computing environment and aligns with the main objectives of this study, providing a strong foundation for the subsequent phase of machine learning model development and training.

3.2 Data Preprocessing

Before deploying machine learning algorithms, the collected dataset undergoes a thorough preparation procedure to assure its quality and suitability. This phase is crucial for ensuring the preservation of data quality and relevance. Firstly, the dataset is normalized to ensure a uniform scale while preserving the distinctions in the value ranges. Subsequently, any absent values in the dataset are dealt with using suitable imputation techniques, thereby preventing potential biases or errors in the ensuing analysis. In addition, categorical variables are systematically encoded to convert qualitative data into a format that can be read by machines. These preprocessing measures are crucial for preparing the dataset for thorough analysis and the subsequent use of advanced feature selection techniques, such as Recursive Feature Elimination (RFE). RFE systematically removes less important features, leaving only the most significant predictors for the machine learning models.

3.3 Multi-Model Machine Learning

The essence of the suggested technique is based on a complex, dual-model machine learning architecture that functions simultaneously. Two specific prediction models used in this framework are Linear Regression and Multi-Output Generalized Regression Neural Networks (MOGRNN). Each model is assigned the goal of predicting the optimal tile size using the well-processed input. The Linear Regression model operates by detecting and applying linear correlations between the characteristics of the dataset and the tile size to produce its predictions. This model demonstrates exceptional proficiency in analyzing straightforward, proportional connections within the data, resulting in a solid foundation for making accurate predictions.

On the other hand, the MOGRNN is a sophisticated model specifically created to uncover complicated, non-linear relationships inside the dataset. The MOGRNN is a specialized version of the Generalized Regression Neural Network (GRNN) designed to handle situations with multiple outputs. By utilizing its neural network architecture, it is able to forecast several dependent variables, namely different dimensions of the tile size, by identifying and representing the non-linear relationships that exist among the input elements.

The combination of these two models enables a thorough analytical approach, where linear predictability is handled using Linear Regression, and the complexities of non-linear patterns are decoded using the MOGRNN. This results in a strong method for predicting the best tile sizes for program optimization.

Software and tools

The core of this methodology is built upon Python, which was chosen for its wide array of libraries and tools specifically developed for machine learning, data analytics, and scientific computation. Aside from a local development environment, use Kaggle, a renowned platform recognized for its vibrant data science competitions and collaborative community. Kaggle is an integrated development environment (IDE), facilitating seamless experimentation with diverse models and providing access to huge datasets and kernels. This access is especially beneficial for comparing methodologies with others and for future model enhancements. The data preparation technique is comprehensive, including normalization, imputation of missing values, and encoding of categorical variables to ensure the highest quality of the input data for the models. These necessary preprocessing methods are executed in a local Python environment and within Kaggle's robust online kernels. Using a dual-environment approach may enhance the processing of large datasets by leveraging the capabilities and adaptability of both local and cloud-based resources.

3.4 Determining the Optimal Tile Size

During the last stage, the predictions generated by both machine learning models are compared to determine the most appropriate tile size. This comparison is crucial for determining a tile size that combines the insights obtained from both linear and non-linear prediction approaches. The most suitable tile size is then determined, taking into account this

comparison study, to ensure that the selected size promotes the most effective execution of the program. The method ends with choosing an ideal tile size, which is anticipated to improve program performance by utilizing the combined knowledge gained from applying both machine learning models. This methodology offers a strong and data-driven approach to optimize the execution of loop kernels in computer programs.

4 Results and Discussion

The experimental findings were obtained by conducting tests on an Intel Core i7-8565U processor operating at a frequency of 1.80GHz with 8 cores. The tests were conducted on a system running Ubuntu 18.04.6 LTS. The CPU in question is equipped with three tiers of cache, namely L1 with a capacity of 64k, L2 with a capacity of 256k, and L3 with a capacity of 819k. The benchmarks utilized in this study are a collection of twenty-three widely recognized linear algebra loop kernels extracted from the PolyBench/C suite is referenced in Table 3 [14]. In the majority of programs, varying input sizes are employed, including but not limited to 300, 400, 500, 600, 1000, 1500, 1800, 2000, 2200, 2500, and 3000, in seqentail order. The dataset at this work has a total of 464 or more instances. The PluTo tool, specifically version 0.11.4, is utilized for the purpose of generating the tiled source code pertaining to the loop kernels under investigation. In general, Pluto accepts C language source code as input and produces an optimized C source code as output. The optimization techniques employed in PluTo encompass loop tiling, vectorization, loop interchange. This work exclusively incorporates the utilization of loop tiling optimization. The 'tile' option was employed to create a tile specifically designed for level 1 cache memory.

4.1 Prediction and evaluation stage for MOGRNN

During the predictive and evaluative phase of the Multi-Output Generalized Regression Neural Network (MOGRNN), the model uses a specific set of characteristics from each dataset to forecast the best tile sizes. These predictions are generated based on the subtle connections and patterns identified between the input characteristics and the multi-dimensional output variables that indicate the sizes of the tiles. After making predictions, the accuracy of the MOGRNN model's forecasts for each program in the test set is evaluated by comparing them to the actual values. The effectiveness of the MOGRNN model's predictions is measured using the Mean Squared Error (MSE) metric. The findings show that the model using dynamic features has a Mean Squared Error (MSE) of 5, whereas the model using static features has a somewhat lower MSE of 4. In this case, the numerical comparison suggests that the static characteristics provide more accurate predictions for the appropriate tile sizes than the dynamic features. The higher precision of the static features technique implies that it is more efficient in collecting the relevant attributes that impact the decision of tile size, even without considering the temporal changes between these attributes.

The upcoming tables, specifically Tables 4 and 5, will provide a clear comparison of the prediction results using dynamic and static features. The Mean Squared Error for each method will also be included. These tables will support the conclusion that the static features method demonstrates a higher level of predictive accuracy in this particular context.

Table 4. Comparison of Actual Values and Predictions with Dynamic Features

Program	Problem size	Tile1	Tile2	Tile3	Prediction1	Prediction2	Prediction3
covcol.c	300	125	183	20	131	186	37
floyd.c	1500	18	232	26	32	238	21
corcol.c	2500	222	254	14	217	245	16
strsm.c	400	256	171	108	240	180	87
doitgen.c	600	10	171	126	36	160	135
ssymm.c	1800	14	202	16	36	210	18
dsyr2k.c	400	175	46	24	158	50	21
ssymm.c	500	238	117	68	242	116	61
tmm.c	2500	256	228	30	244	236	24
jacobi-1d-imper.c	15000	75	14	256	48	12	228

Table 5. Comparison of Actual Values and Predictions with Static Features

Program	Problem size	Tile1	Tile2	Tile3	Prediction1	Prediction2	Prediction3
		202	218	52	200	214	54
corcol.c	600	63	206	18	55	222	11
fdtd-1d.c	2800	188	10	196	201	13	196
matmul-init.c	300	83	200	192	84	207	188
mvt.c	1000	97	222	196	93	221	192
seidel.c	2000	22	63	16	15	58	11
jacobi-1d-imper.c	5000	129	49	156	124	43	167
floyd.c	1800	4	57	14	12	53	17
matmul.c	2500	73	47	160	64	48	158
jacobi-1d-imper.c	15000	75	14	256	43	6	222

4.2 Prediction and evaluation stage for linear regression

Similar to the MOGRNN model, the linear regression model is also subjected to the prediction and evaluation stages for both static and dynamic datasets. The prediction stage for the trained linear regression model involves taking the selected set of features (obtained after performing recursive feature elimination, or RFE) as input and generating predictions for the optimal tile sizes. During the training stage, linear regression establishes a linear relationship between the input features and the target variables (tile sizes). This learned relationship is then utilized for making predictions on unseen data in the testing set. As with MOGRNN, the performance of the linear regression model is evaluated using the Mean Squared Error (MSE) metric. For the dataset with dynamic features, the calculated MSE is impressively low at 0.005, while for the dataset with static features, the MSE is slightly higher, measuring 0.049. This indicates that the linear regression model achieved significantly better accuracy in predicting the optimal tile sizes with the dataset containing dynamic features as compared to the static features. To gain deeper insights into the performance of the linear regression model, a comparison is made between the predicted optimal tile sizes and the actual target values from the testing set.

Similar to the MOGRNN comparison tables, these tables show the predicted and real tile sizes for various programs in the testing set, highlighting the model's accuracy in capturing the optimal tile sizes for dynamic and static feature datasets. The following tables (tables 6 and 7, respectively) show the results of the predictions with dynamic and static features, along with MSE for each method:

Table 6. Comparison of Actual Values and Predictions with Dynamic Features

Program	Problem size	Tile1	Tile2	Tile3	Prediction1	Prediction2	Prediction3
covcol.c	300	125	183	20	125	184	20
floyd.c	1500	18	232	26	18	232	26
corcol.c	2500	222	254	14	222	254	13
strsm.c	400	256	171	108	256	171	107
doitgen.c	600	10	171	126	10	171	127
ssymm.c	1800	14	202	16	15	201	16
dsyr2k.c	400	175	46	24	176	46	24
ssymm.c	500	238	117	68	238	116	68
tmm.c	2500	256	228	30	256	228	30
jacobi-1d-imper.c	15000	75	14	256	74	15	256

Table 7. Comparison of Actual Values and Predictions with Static Features

Program	Problem size	Tile1	Tile2	Tile3	Prediction1	Prediction2	Prediction3
		125	183	20	202	219	51
corcol.c	600	18	232	26	62	206	18
fdtd-1d.c	2800	222	254	14	188	256	196
matmul-init.c	300	256	171	108	82	199	189
mvt.c	1000	10	171	126	96	222	192
seidel.c	2000	14	202	16	23	162	19
jacobi-1d-imper.c	5000	175	46	24	129	48	156
floyd.c	1800	238	117	68	4	57	17
matmul.c	2500	256	228	30	172	147	61
jacobi-1d-imper.c	15000	75	14	256	68	9	250

4.3 The effect of the tile size on the cache miss

The effect of tile size on cache misses was investigated to assess the impact of tiling techniques on cache performance. As an example, to demonstrate the relationship between tile size and cache misses, use the benchmark program "dct.c" with a problem size of 3000, using the Perf tool to measure cache misses. In the original program without tiling, the cache miss rate was measured at 8.15%. However, by applying the optimal tile size, the cache misses were significantly reduced to 1.20%. This substantial reduction indicates that tiling optimizes data locality and minimizes cache conflicts, resulting in improved cache utilization. To further analyze the impact of tile size, a large tile size was also considered. The cache misses for the large tile size were measured at 2.96%. Although this value is higher than the optimal tile size, it is still lower than the cache misses observed in the original program without tiling. This indicates that even with a larger tile size, there is still an improvement in cache behavior compared to the previous program. The effect of different tile sizes on the execution time of the “dct.c” program, with a problem size of 3000, was investigated. The execution times were measured for various tile sizes, allowing us to analyze the impact of tiling on program performance.

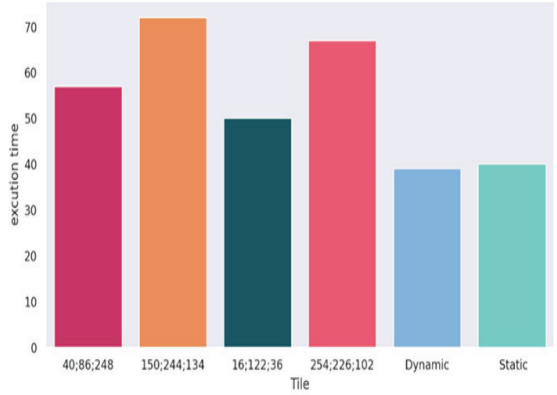

Fig. 2 Shows the execution time for “dct.c” program with different tile sizes with problem size 3000

Figure 2 presents the results of the execution time for different tile sizes, showcasing the effect of tiling on performance. The results are represented in a bar chart, visually comparing the execution times for each tile size. For the “dct.c” program, the execution time varied significantly depending on the tile size. Among the different tile sizes evaluated, the tile size (40, 86, 248) resulted in an execution time of 57 seconds. The tile size (150, 244, 134) led to an execution time of 72 seconds. The tile size (16, 122, 36) resulted in an execution time of 50 seconds. Lastly, the tile size (254, 226, 102) yielded an execution time of 67 seconds.

4.4 Comparing MOGRNN and linear regression results

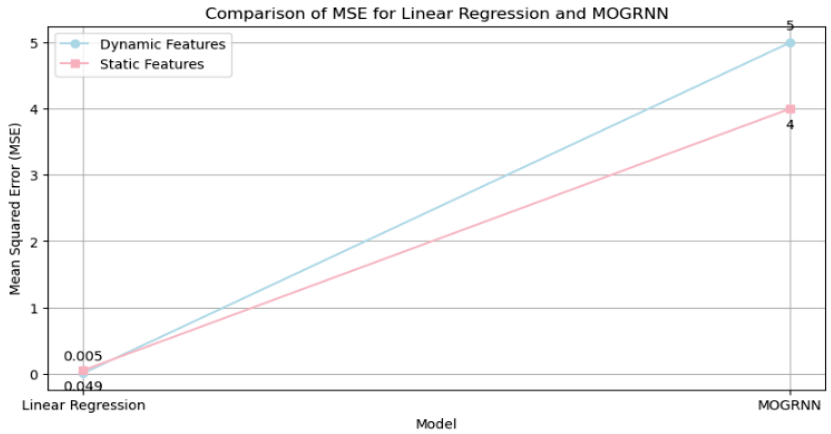

Fig. 3 Shows comparison between Linear regression and MOGRNN

Figure 3 graphically displays the comparison between the Linear Regression model and the Multi-Output Generalized Regression Neural Network (MOGRNN) in terms of Mean Squared Error (MSE) for both static and dynamic feature sets. During the predictive analysis, the Linear Regression model, which utilizes features that have been improved by Recursive Feature Elimination (RFE), shows a high level of accuracy when using dynamic features (mean squared error (MSE) of 0.005). However, it experiences a little loss in precision while using static features (MSE of 0.049).

In contrast, the MOGRNN model, specifically designed to capture non-linear interactions, exhibits higher Mean Squared Error (MSE) values. Specifically, it yields an MSE of 5 for dynamic features and a slightly improved MSE of 4 for static features. This implies that Linear Regression is effective in dealing with datasets that have changing linear relationships, but the MOGRNN model is susceptible to stable characteristics, perhaps catching intrinsic patterns within the information that linear regression would overlook.

The noticeable disparity in model performance, as seen in Figure 3, highlights the significance of selecting a model based on the characteristics of the feature set and the structure of the data. The consistent performance of static characteristics in predicting tile size, independent of the model used, provides useful assistance for selecting the best suitable predictive model to enhance computational program performance.

5 Conclusion

Conclusively, this study has methodically investigated the optimization of loop tiling, a critical approach for improving computational efficiency in scientific computing. Researchers have introduced a thorough methodology that combines machine learning models, notably Linear Regression and Multi-Output Generalized Regression Neural Networks (MOGRNN), to forecast the most suitable tile dimensions for various computing benchmarks. The methodology, presented in a well-organized flowchart, has been carefully developed to include a multi-step process: starting with the choice and implementation of benchmark programs, advancing through feature extraction and dataset creation, then proceeding to data preprocessing, and finally applying machine learning models to predict tile size.

The experimental results have shown that this strategy is beneficial. By combining both linear and non-linear predictive models, a strong framework is created to handle the difficulties of determining the ideal tile size. By employing Recursive Feature Elimination (RFE) during the preprocessing phase, the models have been enhanced to exclusively rely on the most important characteristics for accurate predictions.

Researchers models have demonstrated exceptional accuracy and possess the capability to adjust to the unique requirements of various architectures and applications, as confirmed by thorough examination. By exploiting the capabilities of both static and dynamic features, Researchers technique has produced useful insights into the interplay between program properties and their performance consequences. Furthermore, the empirical investigation has highlighted the crucial influence of tile size on cache performance, with appropriate tiling leading to a large reduction in cache misses and an improvement in total execution time. The finding is backed by quantitative evidence, which strengthens the idea that careful selection of tile size is crucial for maximizing cache use and, therefore, computing efficiency.

Looking ahead, this study's techniques and conclusions establish the foundation for future progress in the field of loop tiling. The integration of machine learning with classical analytical models creates new opportunities for research and development, offering ongoing enhancements in the optimization of computer processes. The study's utilization of a data-driven method showcases the capacity of machine learning to not only supplement but also greatly improve the process of performance improvement in scientific computing.

References

1. S. Liu, Y. Cui, Q. Jiang, Q. Wang, and W. Wu, "An efficient tile size selection model based on machine learning," *Journal of Parallel and Distributed Computing,* vol. 121, pp. 27-41, 2018, doi: 10.1016/j.jpdc.2018.06.005.
2. J. Zhao and A. Cohen, "Flextended Tiles," *ACM Transactions on Architecture and Code Optimization,* vol. 16, no. 4, pp. 1-25, 2019, doi: 10.1145/3369382.
3. J. Zhao and P. Di, "Optimizing the Memory Hierarchy by Compositing Automatic Transformations on Computations and Data," presented at the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020.
4. V. Kelefouras, K. Djemame, G. Keramidas, and N. Voros, "An Analytical Model for Loop Tiling Transformation," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, (Lecture Notes in Computer Science, 2022, ch. Chapter 7, pp. 95-107.
5. N. Prajapati, W. Ranasinghe, S. Rajopadhye, R. Andonov, H. Djidjev, and T. Grosser, "Simple, Accurate, Analytical Time Modeling and Optimal Tile Size Selection for GPGPU Stencils," presented at the Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2017.
6. V. Kelefouras, K. Djemame, G. Keramidas, and N. Voros, "A Methodology for Efficient Tile Size Selection for Affine Loop Kernels," *International Journal of Parallel Programming,* vol. 50, no. 3-4, pp. 405-432, 2022, doi: 10.1007/s10766-022-00734-5.
7. R. Li *et al.*, "Analytical cache modeling and tilesize optimization for tensor contractions," presented at the Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2019.
8. M. M. Riyadh Baghdadi, Mohamed-Hicham Leghettas,Kamel Abdous,Taha Arbaoui,Karima Benatchba,Saman Amarasinghe, "A DEEP LEARNING BASED COST MODEL FOR AUTOMATIC CODE OPTIMIZATION."
9. A. M. Malik, "Optimal Tile Size Selection Problem Using Machine Learning," presented at the 2012 11th International Conference on Machine Learning and Applications, 2012.
10. Y. Sato, T. Yuki, and T. Endo, "An Autotuning Framework for Scalable Execution of Tiled Code via Iterative Polyhedral Compilation," *ACM Transactions on Architecture and Code Optimization,* vol. 15, no. 4, pp. 1-23, 2019, doi: 10.1145/3293449.
11. A. S.-R. Süreyya Emre, Fabrice Rastello, Ponnuswamy Sadayyapan, "Efficient Tiled Sparse Matrix Multiplication through Matrix Signatures."
12. A. B. M. F. Manal H. Almohammed, d Esraa H. Alwan, "Parallel Genetic Algorithm for Optimizing
13. Compiler Sequences Ordering," pp. 128-138.
14. E. H. A. Laith H. Alhasnawy, and Ahmed B. M. Fanfakh, "Using Machine Learning to Predict the Sequences of Optimization Passes," 2020, pp. 139-156.
15. C.-H. Hsu and U. Kremer, “A Quantitative Analysis of Tile Size Selection Algorithms,” Kluwer Academic Publishers, 2004.
16. F. Johnson, O. Oluwatobi, O. Folorunso, A. Ojumu, and A. Quadri, "Optimized ensemble machine learning model for software bugs prediction," in Innovations in Systems and Software Engineering, vol. 19, pp. 1-11, 2022, doi: 10.1007/s11334-022-00506-x.
17. https://www.kaggle.com/datasets/noorasalam/tile-size-selection
18. T. Sharma, A. Jatain, S. Bhaskar, and K. Pabreja, “Ensemble Machine Learning Paradigms in Software Defect Prediction,” in Procedia Computer Science, Elsevier B.V., 2022, pp. 199–209. doi: 10.1016/j.procs.2023.01.002.
19. J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, "Chapter 5-Source code transformations and optimizations," in Embedded Computing for High Performance, pp. 137-183, 2017.