DyLaClass: Dynamic Labeling Based Classification for Optimal Sparse Matrix Format Selection in Accelerating SpMV

Zheng Shi, *Student Member, IEEE,* Yi Zou*, *Senior Member, IEEE,* Xianfeng Song, *Student Member, IEEE,* Shupeng Li, *Student Member, IEEE,* Fangming Liu, *Senior Member, IEEE,* and Quan Xue *Fellow, IEEE*

***Abstract*—Sparse matrix-vector multiplication (SpMV) is crucial in many scientific and engineering applications, particularly concerning the effectiveness of different sparse matrix storage formats for various architectures, no single format excels across all hardware. Previous research has focused on trying different algorithms to build predictors for the best format, yet it overlooked how to address the issue of the best format changing in the same hardware environment and how to reduce prediction overhead rather than merely considering the overhead in building predictors. This paper proposes a novel classification algorithm for optimizing sparse matrix storage formats, DyLaClass, based on dynamic labeling and flexible feature selection. Particularly, we introduce mixed labels and features with strong correlations, allowing us to achieve ultra-high prediction accuracy with minimal feature inputs, significantly reducing feature extraction overhead. For the first time, we propose the concept of the most suitable storage format rather than the best storage format, which can stably predict changes in the best format for the same matrix across multiple SpMV executions. We further demonstrate the proposed method on the University of Florida's public sparse matrix collection dataset. Experimental results show that compared to existing work, our method achieves up to 91% classification accuracy. Using two different hardware platforms for verification, the proposed method outperforms existing methods by 1.26 to 1.43 times. Most importantly, the stability of the proposed prediction model is 25.5% higher than previous methods, greatly increasing the feasibility of the model in practical field applications.**

***Index Terms*—Sparse Matrix, SpMV, High Performance Computing, Sparse Matrix Format, Machine Learning, Classification.**

I. Introduction

SPARSE Matrix Vector Multiplication (SpMV) is an indispensable computational core in various scientific computations and engineering applications, occupying a considerable computational cost in fields such as electronic design simulation [1], machine learning [2], [3], [4], graph computation [5], [6], [7] and more. With the iterative updates of CPUs and the advent of GPU-accelerated computing [8], [9], [10], [11], [12] research into optimizing the performance of SpMV has become more in-depth.

The characteristics of sparse matrices necessitate their storage in a more compact and efficient format in memory, employing various sparse matrix storage formats to enhance the SpMV kernel under different architectures.

*Corresponding author

Fig. 1 illustrates several common sparse matrix storage formats. Over the past decade, there has been considerable research innovating or expanding storage formats [12], [13], [14], [15], [16], [17], [18],recombining classic storage formats or proposing new ones tailored to specific hardware architectures or specific SpMV kernels. Numerous studies [13], [19], [20], [21], [22] have observed that different storage formats of the same matrix can significantly impact SpMV performance, sometimes by several folds. Selecting the most suitable storage format before executing SpMV can significantly improve computational efficiency.

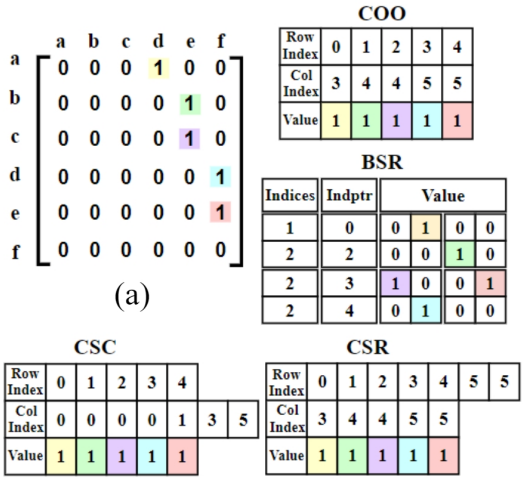


Fig. 1: Illustration of four major sparse matrix formats, where (a) is an example matrix. COO, CSR, CSC, and BSR are the corresponding sparse matrix formats.

There has already been a substantial amount of research [20], [22], [23], [24], [25] aimed at predicting the optimal format for matrices, which has employed different machine learning models to model, analyze, and predict based on the sparse characteristics of matrices and various hardware architectures, achieving good prediction accuracy. Fig. 2 illustrates the overall process of matrix format conversion prediction in an actual SpMV program. Typically, the input is unified into a default format, after which features are extracted, and a predictor predicts the best format, subsequently converting the matrix into a more efficient new format.

For a better understanding of the optimization problem for SpMV in the context of proper storage format selection, let S be the set of all supported matrix storage formats, i.e. S={COO, CSR, CSC, BSR}. Previous research has focused

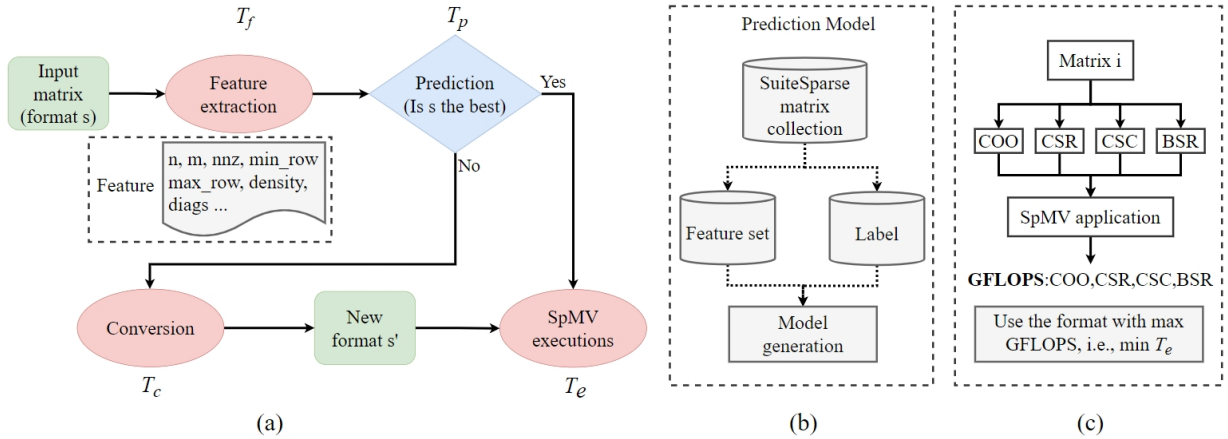


Fig. 2: (a) Illustration of the typical classic matrix storage format selection based SpMV application working flow. Note that some prior work [20], [23], [24], [25], [26], [27] is focusing on minimizing T_e over the storage format selection $s \in S$, i.e. $\arg\min T_e(s)$, while others [28] focus on minimizing T_{total} but only in the context of storage format selection $s \in S$,i.e. $\arg\min T_{total}(s)$. Whereas in this paper, our work is taking into consideration of both format selection and feature selection, i.e., $\arg\min T_{total}(s, f)$. (b) Illustration of the prediction model in the working flow; (c) Illustration for minimizing the SpMV execution time T_e based on format of max GFLOPS in the working flow.

on finding and constructing different features or using various machine learning models to predict the format that minimizes *Execution Time* denoted as T_e, thereby neglecting the time spent on *Conversion* denoted as T_c, *Feature Extraction* denoted as T_f, and *Prediction* denoted as T_p, which together may exceed the total execution time of SpMV T_e. Combining T_f, T_p, T_c, and T_e is the overall *Total Time* overhead, denoted as T_{total}, which can then be calculated as Eq.(1) below.

$$T_{total} = T_f + T_p + T_c + T_e, \tag{1}$$

The T_{total} is related to two major variables, i.e., the selected matrix storage format and the selected matrix features such as row and column size in Table III. Assume F is set of all possible features listed in Table III, thus to minimize T_{total}, we refine the definition of T_{total} as a function of $s \in S$ and $f \in F$ as below, denoted as T_{total},

$$\underset{\forall s \in S, \forall f \in F}{T_{total}(s, f)} = T_f(f) + T_p(f) + T_c(s) + T_e(s), \tag{2}$$

where T_f and T_p are functions of f, T_c and T_e are functions of $s \in S$. Furthermore, for a given $s \in S$, the execution time $T_e(s)$ over a given matrix dataset D is generally given by

$$T_e(s) = \sum_{i}^{D} T_{spmv}(i, s) \times N_i, \tag{3}$$

where $T_{spmv}(i, s)$ is the time cost for a single run of SpMV, N_i is the number of iterations of SpMV such as *PageRank* [29] in Table XIII. Zhou et al. [28] is the first to mention that the conversion cost after prediction imposes significant limitations on format selection. Therefore, it proposes assessing the impact of format conversion before predicting the best format using $T_{total}(s, f)$ directly, achieving overall better results than considering $T_e(s)$ alone. However, this method fundamentally still considers the total time as a whole, predicting the format corresponding to the minimum T_{total} time, i.e.,

$$\underset{\forall s \in S}{\arg\min}\, T_{total}(s) = \{s | T_{total}(s) \leq T_{total}(x), \forall x \in S \setminus \{s\}\}. \tag{4}$$

From Eq. (4) where we can see f is not considered, i.e., individual time cost factors of $T_f(f)$, $T_p(f)$, $T_c(s)$, and $T_e(s)$ still objectively exist and remain unchanged.

Different from previous work, in this paper we establish a holistic view of the overall time overhead $T_{total}(s, f)$, as given by Eq.(5) below as,

$$\underset{\forall s \in S, \forall f \in F}{\arg\min}\, T_{total}(s, f) = \{(s, f) | T_{total}(s, f) \leq T_{total}(x, y), \forall x \in S \setminus \{s\}, \forall y \in F \setminus \{f\}\}, \tag{5}$$

where we can see that the goal is not only to predict the optimal storage format, but also more importantly to optimize and reduce each key time cost factor separately and individually in the entire process, achieving genuine acceleration both accurately and reliably. Note that for convenience of discussion, throughout this paper, we use T_f, T_p, T_c, T_e, and T_{total} in place of $T_f(f)$, $T_p(f)$, $T_c(s)$, $T_e(s)$, and $T_{total}(s, f)$.

Additionally, the optimal format for matrices can vary significantly across different hardware platforms. However, our research has found that even when the same SpMV is executed multiple times on the same matrix on the same platform, the performance can vary with each execution, as shown in Fig. 3. We measure the efficiency of a matrix in the current SpMV application using GFLOPS and select the format corresponding to the highest GFLOPS as the label for the matrix to participate in training. This process is illustrated in Fig. 2, referred to as "Prediction Model". However, we argue that this approach poses a serious prolem where training and building a predictor via minimizing T_e or T_{total} based on the premise that there is a single best format for the matrix on the current hardware platform and application. However, in a

classification prediction model, if the categories of samples to be trained are inherently ambiguous, the constructed predictor will no longer be accurate. This is akin to classifying images of cats and dogs when many images in the training samples are both cats and dogs. Such variations may be related to numerous factors, but it also means that a predictor generated from a single experiment, even after training to achieve very high prediction accuracy, cannot stably predict subsequent matrix inputs, an issue not mentioned in previous articles. Therefore, our constructed predictor needs to be able to make stable judgments on subsequent matrix inputs across different platforms.

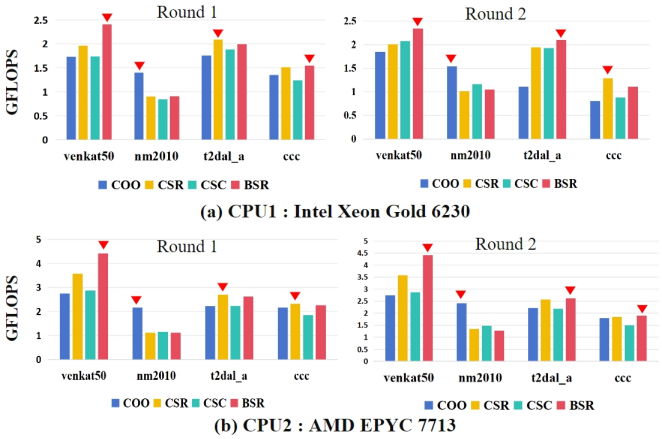


Fig. 3: On two CPUs, the GFLOPS achieved after running four real matrices with four different formats twice for SpMV application.

Overall, we summarize the major contributions from this paper as follows.

- First, we *analyze* the issue of changing the optimal formats for executing SpMV on the same matrix across different platforms.
- Secondly, we *propose* a novel method, DyLaClass, to solve the problem of the ambiguity in optimal format categories.
- In addition, we *introduce* a feature enhancing method in DyLaClass, achieving precise predictions with low computational complexity, significantly reducing the feature extraction time T_f at its core.
- Furthermore, we *conduct* a comprehensive comparative study of DyLaClass based on evaluations of 2617 matrices, using two different real-world applications, demonstrating its outstanding performance advantages.
- Moreover, we *establish* a mathematical model for evaluating the stability of the optimal format prediction model. To the best of our knowledge, this is the first effort aiming to analytically understand the stability problem in this context.
- Last, we *conduct* extensive stability tests under various loads on different hardware platforms, showcasing that the proposed DyLaClass achieves more stable outcomes than existing approaches, even in situations with inherent format category ambiguity.

The remainder of this paper is organized as the following. Section II introduces related work from other studies and provides a detailed discussion of the problems we identified. Section III proposes our solution. Section IV evaluates our approach and compares it with other solutions, also comparing our approach to others in terms of predictive stability. Finally, we conclude the paper and discuss future work in Section V.

II. Related work and current issue

There is a wealth of research aimed at enhancing SpMV performance, which can be specifically categorized into three types: (1) improvements on SpMV based on the CSR format [11], [13], [30], [31], [32], [33]; (2) designing new sparse matrix formats for different hardware architectures or applications [10], [12], [15], [18]; (3) selecting the optimal format based on the characteristics of the sparse matrix [20], [23], [24], [25], [26], [27].

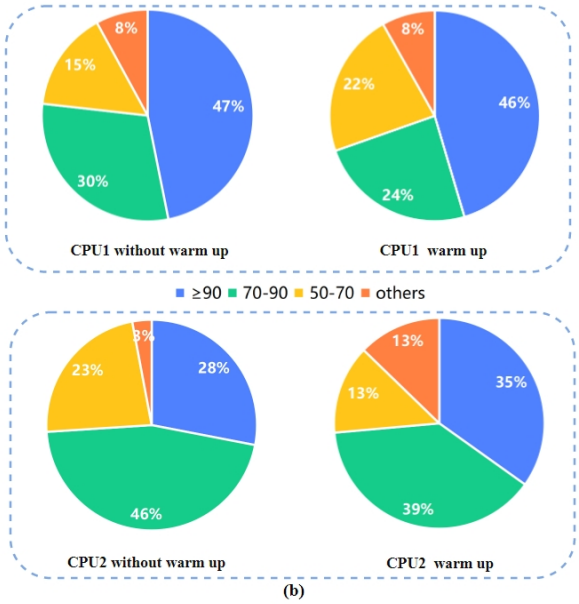
This paper focuses on research in category (3), where early studies on SpMV format selection have made significant progress. Researchers like Li et al. [20] have used decision tree models to choose the best storage format for sparse matrices, while Benatia et al. [23] have constructed new feature factors on support vector machines to achieve notable prediction accuracy. Zhao et al. [25] have attempted to use CNN models to expand the dataset through data augmentation for training on both CPU and GPU platforms, achieving high prediction accuracy, and Nisa et al. [24] also build a prediction model using XGBoost that performed well. However, these efforts largely revolve around using different machine learning models to construct predictors, remaining focused on optimizing prediction accuracy for T_e.

Yilmaz et al. [34] discuss the break-even point, which is the minimum number of SpMV calls required for the conversion benefits to outweigh costs. Although this concept was introduced, it was not integrated into the model, leaving the decision to the users. Zhou et al. [28] proposed a comprehensive cost-considering prediction model called the *Overhead Conscious Method* (OCM), divided in two stages. The first stage decides whether to perform format conversion based on predicted iteration numbers, and the second stage shifts the optimization objective of the predictor from T_e to T_{total}, achieving prediction throughout the process. However, this does not substantially reduce T_{total}; it merely reconstructs a more comprehensive predictor. Lin et al. [9] introduce an adaptive SpMV/SpMSpV framework capable of automatically selecting the appropriate GPU kernel for any sparse matrix and vector at runtime.

Although these works have yielded exceptional outcomes in certain aspects, they all fail to address a crucial aspect: determining the optimal format for a matrix. In classification problems, we first need to identify the category to which the object to be classified belongs, so that the model can learn through training and prediction. Then we consider whether to expand the dataset, think about costs, or use a larger model for optimized classification. Otherwise, the model cannot be effectively trained, nor can it make accurate predictions to achieve acceleration. However, almost all studies evaluate the best format of a matrix using the Top GFLOPS, referrted to as *TopG* throughout this paper, as shown in Fig. 4. Our

File_name	COO	CSR	CSC	BSR
CPU1 without Warm up				
CCC	0	38	0	62
t2dal_a	0	34	56	10
venkat50	0	0	9	91
nm2010	100	0	0	0
CPU1 Warm up				
CCC	0	42	18	40
t2dal_a	0	40	52	8
venkat50	0	0	6	94
nm2010	100	0	0	0
CPU2 without Warm up				
CCC	1	58	0	41
t2dal_a	0	71	0	29
venkat50	0	0	9	100
nm2010	100	0	0	0
CPU2 Warm up				
CCC	0	76	0	24
t2dal_a	0	83	0	17
venkat50	0	0	9	100
nm2010	100	0	0	0

(a)



(b)

Fig. 4: Analysis of the best format variation for matrix warm-up and without warm-up across different platforms, where (a) shows the format distribution corresponding to the best GFLOPS for four real matrix examples after running the SpMV program for 100 times, and (b) involves executing the SpMV program for 100 times on 2617 matrices, dividing them into four situations: $\geq$90 represents the proportion of matrices that consistently choose a certain format more than 90 times, 70-90 represents the proportion of matrices that consistently choose a certain format 70-90 times, along with 50-70 and other situations.

research finds that the maximum GFLOPS corresponding to different formats of the same matrix may change after multiple executions of SpMV.

Fig. 4 shows the optimal format selection for 2086 matrices from the Florida Sparse Matrix Collection and 531 matrices from our cooperation, using four formats on two different architecture CPUs after running SpMV. Among them, Fig. 4(a) represents several typical matrices, showing the distribution of the number of times each of the four formats is chosen after 100 rounds of SpMV. Fig. 4(b) shows the overall distribution more intuitively. Without SpMV warm-up, the number of matrices on CPU1 that consistently chose the same optimal format after 100 rounds of calculation is about 47%, while on CPU2 it is about 28% of the total. Considering the issue of cache preheating, we conduct another set of idealized experiments for comparison. The problem still exists: although the proportion of CPU2 stably choosing one optimal format rose to about 35%, CPU1 becomes worse.

This brings up a very important contradiction, i.e., since the optimal format is variable in actual field operation, how can we ensure that our predictions are correct, or at least as accurate as possible? On the other hand, assuming the accuracy of the labels in this round, the challenge lies in ensuring the trained model from this experiment can make accurate predictions during the next execution of SpMV. Our experiments have already shown that about 50% of matrices change their optimal format in the second round of SpMV execution. Ensuring the stability of the model obtained from a single experiment is a challenging difficulty.

Lastly, reducing prediction time remains a focus, with current research primarily aimed at balancing prediction accuracy with the overhead time. Although a prediction model has been constructed with the minimum total time T_{total} as the target, essentially high cost of feature extraction time T_f and conversion time T_c have not been significantly reduced. What we needed is a prediction model that uses fewer input features and lower computational complexity for prediction.

III. The Proposed DyLAClass

In the previous section, we observed that the highest GFLOPS value for a matrix during repeated SpMV execution corresponds to a change in its optimal storage format. This shift in the best storage format of the matrix becomes the crux of the current issue. The matrices exhibiting this phenomenon in the last experiment accounted for nearly half of the entire dataset. Relying solely on the highest GFLOPS value obtained from a single round of SpMV to determine the matrix's optimal format complicates the classification and selection process. This complexity is further illustrated in Fig. 5, which vividly displays the catastrophic distribution of features on a two-dimensional plane. Manual labeling in such cases tends to have a strong bias towards boundaries. Our challenge lies in finding a feasible solution to this group of complex points.

From Fig. 4(a), we can observe that in each round of results, the first and the second highest GFLOPS of the *CCC* matrix continually alternate between CSR and BSR formats. In *t2dal_a* on CPU1, there is a lower proportion of choosing the third format, but the top two GFLOPS predominantly alternating between CSR and CSC. This indicates that the best format for unstable matrices exists in two or more options, mainly revolving around the formats corresponding to the top two GFLOPS. As shown in Fig. 4(b) under "others", the proportion of matrices choosing among three formats is extremely low, primarily fluctuating around the formats

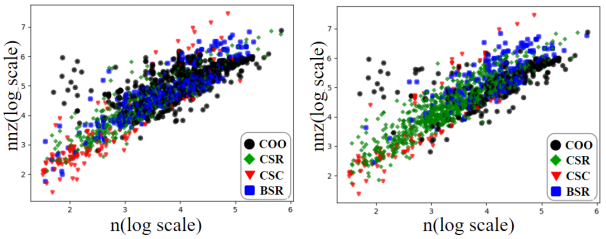


Fig. 5: (a) Simplified view of format selection in existing literatures. (b) Illustration non-optimal selection of CSC based on GLOPS. Illustration of label distribution on a 2D feature plane n-nnz respectively for (c) CPU1, and (d) CPU2.

corresponding to the top two GFLOPS. Our approach is to build a reasonable model that, while ensuring accuracy, can accommodate GFLOPS fluctuations. That is, we can predict the top two GFLOPS formats, and the model trained with this data can reliably predict during subsequent SpMV executions, while minimizing input features and reducing prediction time.

To address this, we propose *DyLaClass*, a novel algorithm for selecting the optimal format, which is highly effective in optimizing the sparse matrix format selection model. This method consists of two parts. The first part is *dynamic label based partitioning*, where we redefine the method for determining the best storage format by dividing matrix labels into two types of matrices, namely *Fixed Label Matrix* (*FLM*) and *Mixed Label Matrix* (*MLM*). Through a clustering method constrained by self-selected input features, MLM is clustered towards the adjacent FLM, achieving dynamic partitioning of matrix labels. The second part is the *flexible feature selection*, which allows users to independently choose features as constraints for clustering. Through our evaluation, the relevance of autonomously selected features in subsequent model predictions is enhanced, thereby enabling users to selectively choose features with lower computational complexity, reducing the overhead of feature extraction.

A. Dynamic Labeling Based Classification

Our input data utilizes 2086 matrices from the Florida Sparse Matrix Dataset. For each matrix, we run a round of PageRank in four different sparse formats, recording the GFLOPS value for each format.

1) Fixed and mixed labels: After completing the aforementioned process, we have the GFLOPS values for each matrix in four formats. We then compare the formats corresponding to the top two GFLOPS values. Fig. 6 illustrates our method of defining fixed and mixed labels. For each matrix, we use θ to assess the size of the difference between the highest GFLOPS value and the second GFLOPS value. If the highest GFLOPS value significantly exceeds the second, we designate the format corresponding to this highest GFLOPS as the matrix's label and call it FLM. If the difference is small, we consider the formats corresponding to the top two GFLOPS as labels of the matrix, meaning that it has two optimal formats, and we refer to it as MLM. Through our experiments, we find that σ=0.2 offers the best performance, as matrices with σ greater than 0.2 demonstrated stronger stability in our tests. While matrices with σ between 0.1 and 0.2 also showed considerable stability, there is a lower probability of their optimal format changing.

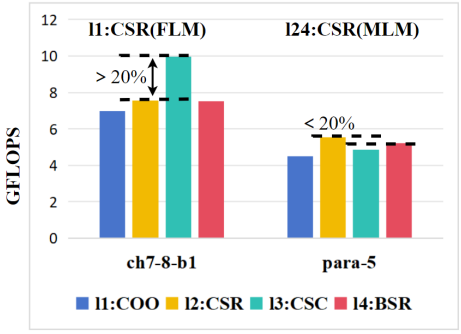


Fig. 6: Example of producing FLM and MLM based on GFLOPS for matrices `ch7-8-b1` and `para-5` with $\theta = 0.2$.

TABLE I: Distribution of FLM and MLM as output from two different CPU platforms.

CPU	FLM				MLM					
	l_1	l_2	l_3	l_4	l_{12}	l_{13}	l_{14}	l_{23}	l_{24}	l_{34}
CPU1	237	11	12	13	345	284	176	311	611	88
CPU2	111	15	1	26	154	92	48	310	1312	18

After the above process, we obtain a dataset with FLM and MLM labels. Table I illustrates the FLM and MLM distribution with $\theta = 0.2$, where l_1, l_2, l_3 and l_4 correspond to FLM labels of COO, CSR, CSC, and BSR and $l_{12}, l_{13}, l_{14}, l_{23}, l_{24}$, and l_{34} correspond to MLM of COO&CSR, COO&CSC, COO&BSR, CSR&CSC, CSR&BSR, and CSC&BSR. For example, the MLM label l_{12} indicates that the GFLOPS of COO and CSR differ by less than θ. Fixed labels like l_1 and l_4 represent the highest GFLOPS exceeding the second highest GFLOPS by more than θ. Effectively, we extend the set of neighbors in classification of sparse matrix types for a given feature vector data point based on FLM and MLM, improving classification performance.

2) Classification using FLM and MLM: We next apply FLM and MLM labels into the distance function for classification. For example, for a point A on a given feature plane n-nnz in Fig. 7, corresponding to a given feature set F, the distance function for A is then defined as:

$$dist(A, F) = \arg\min_{\forall l \in l_A} dist_l(A, F), \tag{6}$$

where l_A indicates labels for A, e.g., $l_A = l_{14} = \{l_1, l_4\}$ in Fig. 7, and $dist_l(A, F)$ is the average distance between A and all points with label l, defined below as:

$$dist_l(A, F) = \frac{\sum_{\forall B \in N_l}(d_{AB})}{\|N_l\|}, \tag{7}$$

where N_l is set of points with label l, $\|N_l\|$ is the cardinality of the set N_l, and $d_{AB} = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$ is the standard Euclidean distance between A and B over the 2D feature plane n-nnz, as shown in the example in Fig. 7.

Note that Eq. (6) and Eq. (7) effectively takes the shortest distance of all labels as the optimal label for the target matrix. Intuitively, we consider all possible labels for classification

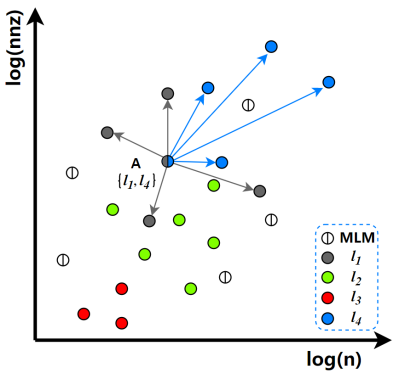


Fig. 7: An illustration of FLM and MLM based classification on feature plane n-nnz.

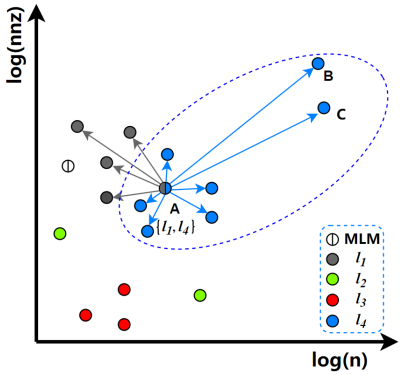


Fig. 8: An illustration of impact on classification from dispersed points B and C.

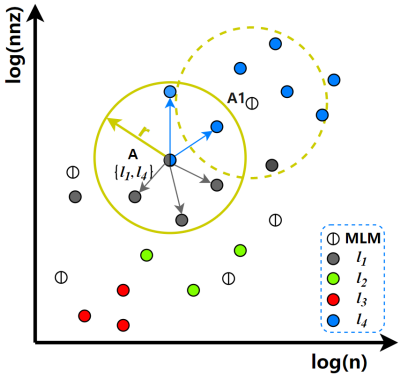


Fig. 9: An illustration of the local radius filtering algorithm for removing dispersed points.

for the given input sparse matrix. Essentially, this means that the classification is smart enough to avoid the problem of false labeling that is almost hard-bound to the one with top GFLOPS, as shown in Fig. 6. This approach is simple with little overhead, e.g., calculation over MLM of size $\|l_A\|$, but very effective, particularly for uniformly distributed labels.

3) Solving dispersed points: However, we also note that there exist biases in distance calculations adversely affecting the clustering results. As illustrated in Fig. 8 where A and $l_A = l_{14} = \{l_1, l_4\}$, ideally A is expected to be classified to be with cluster of l_4. However, the existence of the two dispersed points, denoted as B and C, which lead to $dist_{l_4}(A, F) > dist_{l_1}(A, F)$, resulting A being incorrectly classified to be with cluster of l_1.

To mitigate the impact from disperse points, we introduce a filtering method using a local radius r calculated from the diagonal distance of the hypercube formed by all data points over the feature vector space defined by the given feature set F as follows:

$$r = \alpha \sqrt{\sum_{f \in F} (\arg\max_{\forall B \in N} f_B - \arg\min_{\forall B \in N} f_B)^2}, \quad (8)$$

where f_B is the feature value corresponding to a given data point B along the axis of f in the feature plane, N is the input set of data points in the feature vector space, and α is a scaling factor. In our experiments, we find that $\alpha = \frac{1}{15}$ offers best results.

TABLE II: Label distribution on n-nnz using the proposed DyLaClass, as compared to TopG.

CPU	Method	COO	CSR	CSC	BSR	Loss
CPU1	DyLa	537	809	343	97	3.1%
	TopG	671	628	142	345	/
CPU2	DyLa	206	1193	303	84	1.2%
	TopG	266	1228	100	192	/

Using the aforementioned method, we obtain a new set of four class labels. Table II shows the label distribution on the n-nnz plane after applying DyLaClass. Compared to TopG, i.e., the traditional method of considering only the format corresponding to the highest GFLOPS as optimal matrix format [20], [23], [24], [25], [26], [27], the average performance loss with the format selected by DyLaClass is about 2%, where the performance loss is defined in Eq.(9) as,

$$Loss = \sum_{i \in D} \frac{TopG\_Best_i - DyLa\_Best_i}{TopG\_Best_i}, \quad (9)$$

where $TopG\_Best_i$ denotes the GFLOPS corresponding to the optimal format for the ith matrix, and $DyLa\_Best_i$ represents the GFLOPS for the most suitable format selected by DyLaClass, D is the entire test dataset. The performance loss is due to DyLaClass potentially selecting the format corresponding to the second highest GFLOPS. However, it is important to note that, as verified in Fig. 4, GFLOPS can vary across multiple experiments. Therefore, this 2% loss is only the difference between this round of SpMV and the optimal performance.

B. Flexible Feature Selection

Table. III shows the list of features that we explore in this paper. Unlike previous studies that attempt to identify unique features or use deep learning methods to score and select from a large number of features, the features we have chosen are of very low computational complexity. Moreover, after clustering under the aforementioned specific constraints, the labels corresponding to these features are enhanced.

1) Feature distribution: From Fig. 10, we can observe that matrices exhibit different distributions on a two-dimensional plane composed of various features. Both n and $Ndiags$ generally show an increasing trend with the increase of nnz. There is no obvious correlation between mu and nnz, and the $density$ generally shows a decreasing trend as nnz increases. This indicates that different features carry distinct information.

2) Label distribution after DyLaClass: Fig. 11 shows the results of clustering using n and nnz as feature constraints within a two-dimensional plane using DyLaClass. We represent the newly generated labels from clustering with different colors. As can be seen in Fig. 11(a), both CPU1 and CPU2 exhibit more distinct boundaries compared to Fig. 5. Additionally, although the clustering is performed in the n-nnz plane, when these new labels are displayed in other feature planes, such as n-mu and n-cv, the boundaries remain significant. This suggests that we can choose a wider array of other

TABLE III: Features and their corresponding complexity.

Features	Description	Complexity
n,m	row value, column value	O(1)
nnz	number of non-zeroes	O(1)
Ndiags*	number of diagonals $(n+m-1)$	O(1)
dens*	density of non-zero elements (nnz/nm)	O(1)
mu*	average number of non-zero values per line	O(n)
cv*	coefficient of variation of non-zero elements per row	O(n)
sd*	standard deviation of non-zero elements per row	O(2n)
max_RD*	maximum relative density	O(n)
min_RD*	minimum relative density	O(n)
max_Row*	maximum number of non-zero elements	O(m)
min_Row*	minimum number of non-zero elements	O(m)
relative_range*	$(max\_row - min\_row)/n$	O(n)

* These features are commonly seen in other work [20], [23], [24], [25], [26], [27]. but we are not using them in this paper for a more computationally efficient design

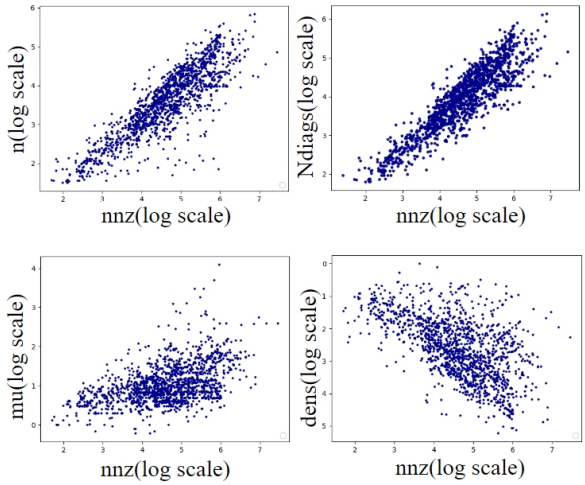


Fig. 10: Feature distribution on the sparse matrices sorted by increasing values of $log(nnz)$.

feature combinations for multi-dimensional training to achieve optimal results.

3) Flexible feature selection: Table. IV presents our scoring of various features under several feature constraints using the F-Score. Our tests show that using different feature combinations for constraints enhances the relevance of the feature, as directly reflected in the increased correlation scores provided by the F-Score. For example, clustering in the n-nnz plane significantly increases the relevance of nnz compared to other features. Similarly, clustering in the n-m plane raises the relevance of m. This not only strengthens the boundaries between different categories in our method, but also enhances the correlation of input features. We can flexibly choose features with low computational complexity for training according to our needs.

C. Stability

As we observe in previous section, there exists a certain degree of variability in the optimal matrix format across multiple SpMV executions. This variability implies that the

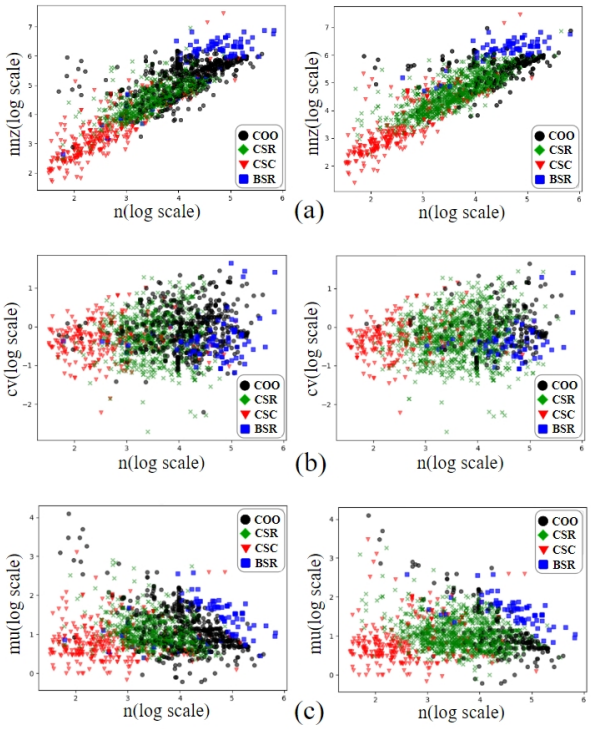


Fig. 11: After applying the DyLaClass on the n-nnz plane, the colored projections of matrix label categories on the (n-nnz), (n-cv), and (n-mu) planes across two CPUs.

TABLE IV: F-Score of 8 features

plane	n	m	nnz	$diags$	mu	sd	$dens$	cv
n-nnz	154	174	98	174	9	61	94	24
n-m	193	234	57	226	2	47	101	35
n-dens	180	196	45	199	3	43	102	27

prediction model has inherent instability when the actual available computation resources vary, such as memory and CPU cycles. To assess such stability, we consider the following two specific criteria,

1) For TopG, we verify whether the format chosen for each matrix in a single experiment remains the highest GFLOPS corresponding format in the controlled experiment.
2) For DyLaClass, we check if the format chosen for each matrix in a single experiment continues to be one of the top two GFLOPS corresponding formats in the controlled experiment.

Mathematically, we describe the prediction model stability, denoted δ, using labels generated from two distinct experimental setups, i.e., the existing Top GFLOPS or the proposed DyLaClass experiment and the correspoding controlled experiment. δ is defined as,

$$\delta = \frac{1}{|D|} \sum_{\forall i \in D} I(L_i^a, L_i^b), \tag{10}$$

where D denotes the matrix dataset, the L_i^a is the label set produced from a single run of the DyLaClass experiment for matrix i, and L_i^b is the label set from the corresponding

controlled experiment for matrix i. Given that DyLaClass aims to identify the top two highest GFLOPS formats, specifically $L_i^b = \{l_{\text{top1}}, l_{\text{top2}}\}$. In Eq.(10), we employ the indicator function $I(L_i^a, L_i^b)$ to ascertain the frequency at which L_i^a is included within L_i^b. $I(L_i^a, L_i^b)$ is defined below as,

$$I(L_i^a, L_i^b) = \begin{cases} 1 & \text{if } L_i^a \subseteq L_i^b \\ 0 & \text{if } L_i^a \not\subseteq L_i^b \end{cases} \tag{11}$$

As we can see, the stability δ varies between 0 and 1. when all produced label set L_i^a from DyLaClass is always a subset of the corresponding L_i^b produced by the controlled experiment, i.e. when $L_i^a \subseteq L_i^b, \forall i \in D$, δ reaches is upper bound as 1, proving the prediction model is accurate. On the other hand, for the worst case when DyLaClass is giving all wrong labels, i.e., $L_i^a \not\subseteq L_i^b, \forall i \in D$, then we have the lower bound of $\delta = 0$.

IV. Evaluation

In this section, we provide evaluations of the proposed DyLaClass classification algorithm for sparse matrix format selection. We provide detailed performance analysis and share our thoughts for future explorations.

A. Lab Setup

Considering hardware platform impacts, we use two mainstream CPU architectures, one from Intel and one from AMD, as the computation hardware platforms for our evaluations, where details of key parameters of the platforms are shown in Table V.

TABLE V: Key parameters of computation platforms.

CPU	CPU1	CPU2
CPU model	Intel (R) Xeon (R) Gold 6230	AMD EPYC 7713
Architecture	x86_64	x86_64
Frequency (MHz)	1262. 035	1700
Ll/L2/L3 cache (Mib)	1.3/40/55	4/64/512
Cores per socket/Threads per core	20/2	64/2

B. Evaluation on Classification Accuracy

1) ML models and datasets: In our experiments, the input sparse matrix is first processed through a SpMV application to obtain GFLOPS in four formats. Here, we employ PageRank as the practical application for this experiment and tailor a model specifically for PageRank. We leverage an extensive range of machine learning algorithms for format selection and performance modeling to assess our algorithm, following the recommendations of Benatia et al. [23] and Sedaghati et al. [22]. We use models based on *SVM* and *Decision Tree* (*DT*), supplemented by *MLP* and *XGBoost* as adopted by Nisa et al. [24]. We also compare the *Random Forest* (*RF*) model as an additional benchmark. We randomly select 80% of each category to form the training set, with the remaining 20% of each category comprising the test set. Clustering is performed once on the n-nnz plane with DyLaClass. We tune the model parameters using *GridSearchCV*. We utilize a dataset comprising 2086 matrices from the University of Florida Sparse Matrix Collection [35] and 531 matrices from our cooperation, These two datasets were used for evaluation with the PageRank and *BiCGSTAB* [36] algorithms, respectively. The experiments are based on NumPy library.

2) Classification Accuracy: We utilize the eight features listed in Table III as inputs and use the labels clustered by DyLaClass as the most suitable storage format for the matrix. The matrix is then trained and predicted using the aforementioned machine learning models. It is important to note that we do not optimize the machine learning models themselves; the five models are merely used to evaluate the performance of DyLaClass. As shown in Table VI, among the five machine learning models, CPU1 achieves a stable accuracy rate of over 81%, with *XGBoost* achieving the highest accuracy. Similarly, our prediction accuracy on CPU2 can reach over 87%, with the best-performing model being XGBOOST. In contrast to merely using the highest GFLOPS as the best format for prediction, we still choose the aforementioned eight features for training. The results, as shown in Table VI, are quite poor.

TABLE VI: Using features $\{n, m, nnz, diags, dens, cv, sd, mu\}$.

Method	CPU	SVM	XGBoost	MLP	RF	DT
DyLaClass	**CPU1**	80.22%	**85.21%**	81.89%	84.12%	83.29%
	CPU2	86.91%	**91.09%**	89.14%	90.81%	88.86%
TopG	**CPU1**	57.94%	62.95%	59.33%	**63.51%**	61.28%
	CPU2	76.04%	**79.67%**	77.44%	79.11%	77.71%

From Fig. 11, we can clearly see that the labels under each feature plane after being processed by DyLaClass have distinct boundaries. Consequently, we attempted to reduce the number of input features, adjusting the model to accept 3 and 5 features as input. The resulting prediction accuracy, as shown in Table VII and Table VIII, indicates that even with fewer input features, we are still able to maintain a very high accuracy. Although there is a slight decrease in accuracy, having fewer input features means faster prediction time and reduced feature extraction time. This lays a solid foundation for further acceleration.

TABLE VII: Using features $\{n, m, nnz, diags, dens\}$.

Method	CPU	SVM	XGBoost	MLP	RF	DT
DyLaClass	**CPU1**	79.67%	**85.14%**	82.73%	84.12%	83%
	CPU2	84.4%	90.25%	87.47%	**90.81%**	88.86%
TopG	**CPU1**	54.04%	59.61%	57.1%	**59.89%**	56.82%
	CPU2	75.97%	78.32%	77.44%	**78.55%**	77.72%

TABLE VIII: Using features $\{n, m, nnz\}$.

Method	CPU	SVM	XGBoost	MLP	RF	DT
DyLaClass	**CPU1**	79.67%	83%	82.17%	**83.84%**	82.45%
	CPU2	87.47%	**90.25%**	88.86%	89.97%	89.97%
TopG	**CPU1**	53.6%	57.05%	55.27%	**57.61%**	55.1%
	CPU2	76.32%	**76.27%**	74.16%	76.27%	73.16%

On the other hand, through F-Score analysis, we discover that the labels obtained after clustering by DyLaClass enhance the features on their constrained feature planes. From Table. IV, we observe clustering on three two-dimensional planes

of relatively low computational complexity: n-nnz, n-m, and n-$dens$, where the corresponding feature scores for nnz, m, and $dens$ have all been enhanced. Based on this, we conduct additional training to identify the optimal set of three feature inputs for prediction accuracy. Fig.12 illustrates the prediction accuracy after training with different feature inputs obtained from clustering on three feature planes, where we only utilize *XGBOOST* as the model.

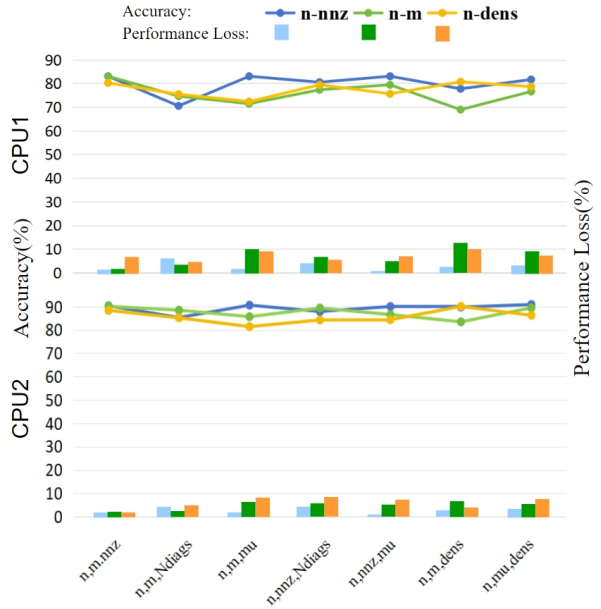


Fig. 12: After applying the DyLa algorithm on three two-dimensional planes (n-nnz, n-m, n-$dens$), an analysis of the prediction accuracy for any given input of three low computational complexity features is conducted, along with an evaluation of the losses incurred due to prediction errors.

From Fig.12, it can be observed that the prediction accuracy of labels clustered on various two-dimensional planes changes with different combinations of input features. The average accuracy remains around 80% on CPU1 and above 85% on CPU2. Although the combination of features n, nnz, and mu achieves the same accuracy as n, m, and nnz, we still choose the feature combination with the lowest computational complexity as the input for this experiment.

C. Evaluation on Cost in Time

We verify the series of acceleration effects brought about by DyLaClass based on three aspects, time for single round SpMV execution, i.e. T_{spmv}, time for feature extraction and prediction, i.e., T_f and T_p, and the format conversion time T_c. The comparative study using these metrics collectively demonstrate the acceleration effects of the proposed DyLa-Class. Recall from Eq. (1)-(5), where minimizing the overall time overhead T_{total} would essentially imply to reduce T_{spmv}, T_p, T_f, and T_c as much as possible. We discuss in detail below on how DyLaClass is able to reduce these major cost factors to improve the overall T_{total}.

1) SpMV execution time T_{spmv}: We calculate the time it takes to execute a single SpMV on a given CPU for the input set of 2086 matrices. As Table IX indicates, on CPU1, the average execution time is approximately 0.00041 seconds, while on CPU2 it is 0.00025 seconds. However, as we note that there exists a small numbe of matrices with an excessively large number of non-zero values, without loss of generality, we also calculate the median of T_{spmv} in Table IX, which are on the order of e^{-5} seconds.

TABLE IX: Comparison of T_{spmv} across different CPUs.

Format	T_{spmv} on CPU1		T_{spmv} on CPU2	
	Average (s)	Median (s)	Average (s)	Median (s)
COO	$4.1e^{-4}$	$5.7e^{-5}$	$2.5e^{-4}$	$4.3e^{-5}$
CSR	$4.1e^{-4}$	$6e^{-5}$	$2.5e^{-4}$	$3.9e^{-5}$
CSC	$4.4e^{-4}$	$7e^{-5}$	$2.7e^{-4}$	$4.5e^{-5}$
BSR	$4.2e^{-4}$	$6.2e^{-5}$	$2.5e^{-4}$	$4.1e^{-5}$

2) Feature prediction time T_p and extraction time T_f: We conduct a comparative analysis of prediction time for models with inputs of 3, 5, and 8 features, as illustrated in Table X. We can see that the prediction time T_p is less than that for a single SpMV operation. In addition, T_p cost for 3 feature inputs is approximately 8% of that for 8 feature inputs. Moreover, although some studies use CNNs [25] to predict the best format, in our experiments, the time taken to convert matrices into images is at the order of e^{-1}s, far exceeding both SpMV execution time and prediction time. Hence we do not consider using CNNs or other deep learning models in this paper.

TABLE X: T_p and T_p/T_{spmv} with 3, 5, and 8 features.

Format	T_p on CPU1		T_p on CPU2	
	Average (s)	T_p / T_{spmv}	Average (s)	T_p / T_{spmv}
3	$1.4e^{-5}$	0.1	$0.9e^{-5}$	0.1
5	$3.9e^{-5}$	0.6	$2.5e^{-5}$	0.6
8	$7.8e^{-5}$	1.3	$5.9e^{-5}$	1.4

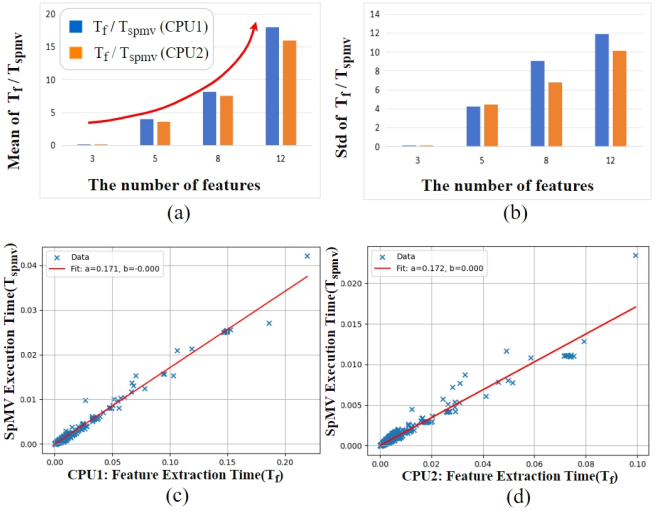


Fig. 13: Correlation between T_f and T_{spmv}.

We also highlight the impact from feature extraction time T_f in relationship to single SpMV execution time T_{spmv}, as illustrated in Fig. 13 and Table XI. In Fig. 13(a)-(b), we can

intuitively observe the relationship between T_f and T_{spmv} when number of input features are varying from 3, 5, 8 to 12, where we analyze the ratio T_f/T_{spmv} as an indicator to the overhead of the feature extraction. Note that for low computational complexity features as input, e.g., 3 and 5, the corresponding T_f/T_{spmv} is approximately 0.1 and 2.5 on both CPUs. However, when incorporating more features, particularly with high computational complexity features such as max_RD and max_Row, which are commonly used in existing studies [9], [28], T_f/T_{spmv} begins to increase exponentially. As shown in Fig. 13(a), T_f/T_{spmv} reaches approximately 6 for input of 8 features, more than double of input of 5 features, and quickly exceeds 18 for input of 12 features. Another interesting aspect for discussion is that for a given number of features as input, we observe a linear relationship between T_f and T_{spmv} for all matrices in the dataset. This illustrated by Fig. 13(c)-(d), where the linear relationship of T_f/T_{spmv} is consistent across different CPUs. In contrast in DyLaClass, we only need three extremely low computational complexity features to achieve high-precision prediction, with very low feature extraction overhead consistently for all matrices.

TABLE XI: Correlation between T_f/T_{spmv} and number of input features for an example matrix $venkat50.mtx$.

Feature (venkat50.mtx*)	T_f on CPU1		T_f on CPU2	
	T_f (s)	T_f/T_{spmv}	T_f (s)	T_f/T_{spmv}
n	6.91e^{-6}	x	1.67e^{-6}	x
m	9.54e^{-7}	x	4.77e^{-7}	x
nnz	5.48e^{-6}	x	1.91e^{-6}	x
mu	1.4e^{-3}	1	7.15e^{-4}	1
cv	9.36e^{-3}	5	4.13e^{-3}	4
sd	1.2e^{-2}	7	5.42e^{-3}	5
Ndiags	1.78e^{-5}	x	1.68e^{-5}	x
dens	1.19e^{-6}	x	8.54e^{-7}	x
max_RD	1.61e^{-2}	9	1.21e^{-2}	12
min_RD	1.61e^{-2}	9	1.28e^{-2}	13
max_Row	7.73e^{-3}	4	6.37e^{-3}	6
min_Row	7.17e^{-3}	4	6.96e^{-3}	7

* For $venkat50$, the corresponding T_{spmv} on CPU1 is 1.8e^{-3}s, and on CPU2 it is 1e^{-3}s. "x" means T_f/T_{spmv} is too small to be considered.

Table XI demonstrates the other important observation, where the feature extraction time T_f varies significantly due to difference in matrix dimension and number of non-zero values in each matrix. Taking the $venkat50$ matrix as an example, with CSR as input format and eight features from Table III, the single SpMV execution time T_{spmv} for $venkat50$ is $1.1e^{-3}$s and $0.9e^{-3}$s, respectively on CPU1 and CPU2. From Table XI, it is evident that even when we select 8 features with low computational complexity, it still leads to a performance degradation since the high ratio of T_f/T_{spmv} results in significant time loss.

3) Conversion time: Next, we consider the overhead from format conversion. After model prediction, the matrix needs to be converted into a different format. Taking $venkat50$ as an example again, Table XII lists the corresponding conversion time for four different formats. Here, we continue to use CSR as the default input format. The conversion overhead from CSR to other formats is generally above 6. If the prediction result remains CSR, then the conversion overhead is negligible.

TABLE XII: T_c and T_c/T_{spmv} from CSR to other formats.

Format (venkat50.mtx)	T_c on CPU1		T_c on CPU2	
	T_c (s)	T_c/T_{spmv}	T_c (s)	T_c/T_{spmv}
COO	1.7e^{-2}	9	9.6e^{-3}	10
CSC	1.4e^{-2}	8	6.4e^{-3}	6
BSR	3.7e^{-2}	20	2.1e^{-2}	21

Fig. 14 illustrates the relationship between the conversion time T_c from CSR to COO, CSC, and BSR formats. Particularly, Fig. 14(a) shows how T_c/T_{spmv} varies for these formats with input of 3, 5, 8, and 12 features, respectively. As we can see that the ratio of T_f/T_{spmv} is approaching 1 when numnber of features is 8, implying that T_c is approaching T_{spmv}. Moreover, when considering other commonly used and more complex features, T_c/T_{spmv} increases exponentially. In a different perspective, Fig. 14(b)-(d) also demonstrate a very high linear correlation between T_c and T_{spmv} for all matrices in the dataset. The similar linear trend in Fig. 13

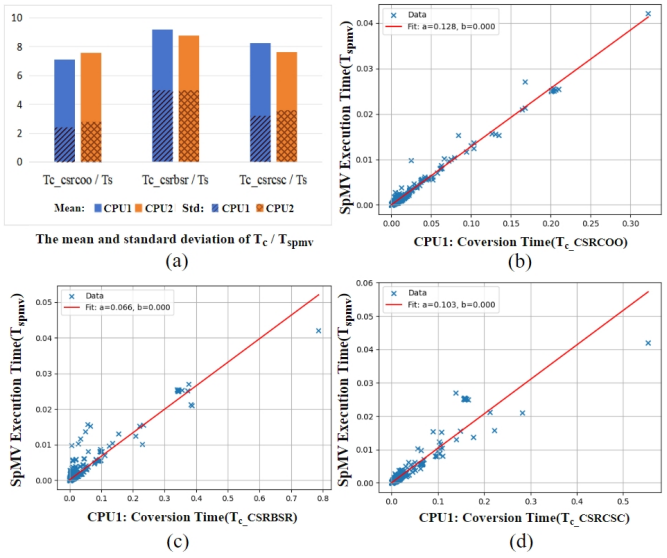


Fig. 14: Correlation between T_c and T_{spmv} from CSR to COO, CSC, BSR with 3, 5, 8, and 12 features.

and Fig. 14 indicates that as matrix complexity increases, both feature extraction time T_f and conversion time T_c almost synchronously increase, maintaining a stable relationship.

As Fig. 13 and Fig. 14 both show similar linear relationships between T_f and T_c to T_{spmv}. We further explore the the correlation between T_c and T_f in Fig. 15, which without much surprise but interestingly also demonstrates a very high linear correlation. This also highlights the significant advantage of DyLaClass in terms of feature extraction overhead compared to other methods.

4) Overall performance evaluation: Recall that our goal is to optimize the overall performance of T_{total} given by Eq. (1).

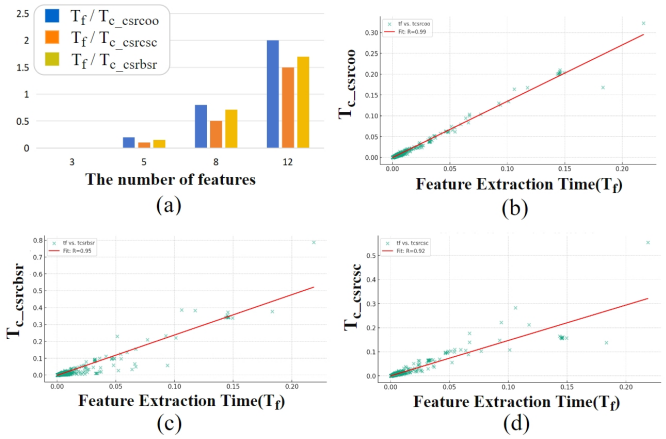


Fig. 15: Correlation between T_c and T_f from CSR to COO, CSC, BSR with 3, 5, 8, and 12 features.

With the above analysis for the four major time-consuming processes, i.e., T_{spmv}, T_f, T_p, and T_c, we present the evaluation of the entire acceleration process in the following. As Table X reveals, the prediction time T_p from the proposed DyLaClass is much less than a single SpMV execution time T_{spmv}. However, the prediction accuracy is much higher than that of the TopG approach, where higher accuracy directly leads to a higher probability of choosing the most appropriate format. Furthermore, with DyLaClass, we only need three input features, namely n, m, and nnz, to achieve very high accuracy. As Table XI indicates, the feature extraction time of these three low complexity features is only about $1/10$ of T_{spmv}, significantly reducing the overhead from T_f, which is not possible in previous methods. Also, for the main contributing factors in T_{total}, i.e., the conversion time T_c and the required iterated SpMV execution time $\sum_i T_{spmv}(i) \times N_i$, Since DyLaClass is trained beforehand, it does not incur any other time overhead on T_{total}.

In Fig. 16, We compare the acceleration ratios from DyLaClass with various existing methods, with CSR format as the baseline, including TopG representing Top GFLOPS [20], [22]–[24], OCM representing the Overhead Conscious Method [28], and DyLa representing the proposed DyLaClass method. For completeness, we also include TopG_Best and DyLa_Best, which respectively represent the acceleration upper limits from Top GFLOPS and DyLaClass, assuming the ideal scenario where all matrices are in the best format with no overhead in conversion or prediction.

From Fig. 16, it is evident that at low iteration counts, the overhead of feature extraction and format conversion outweighs the benefits of selecting the appropriate formats. Therefore, TopG may incur significant negative returns, particularly at small iterations. On the other hand, After evaluating the overhead of the first stage prediction, OCM retains the default CSR format without performing format prediction and conversion, resulting in no acceleration effect. However, its prediction evaluation also has significant errors, leading to negative returns due to format conversion and feature extraction overhead. In contrast, DyLaClass only incurs overhead from format conversion, resulting in lower negative returns.

In matrices with a higher number of iterations, DyLaClass is able to fully utilize its advantage of very low feature

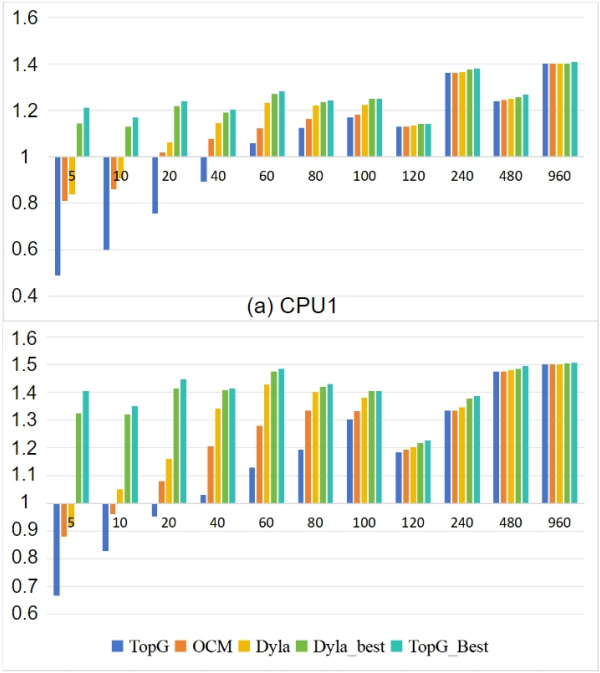


Fig. 16: Comparison of acceleration effects of from TopG, OCM, and DyLaClass for PageRank on (a) CPU1 and (b) CPU2, with CSR as the default input format. The data sorted by the number of iterations for each matrix in the dataset.

extraction time, which is unavoidable to both OCM and TopG. When OCM predicts to keep the default CSR format to avoid deceleration, it outperforms TopG after 40 iterations. However, note that when OCM determines that a format conversion is necessary, feature extraction time is still inevitable, where DyLaClass still outperforms OCM. Also note that for DyLa_Best, we do not always choose the highest GFLOPS value, so compared to TopG_Best, there is a slight loss, with the loss assessment being about 2% as analyzed in Table II.

As the number of matrix iterations continues to increase, compared to using only the CSR format, TopG, OCM, and DyLaClass all show better acceleration effects. As Eq. (3) shows, the iterations of T_{spmv} would make the majority of T_{total}. Still, in all cases, DyLaClass outperforms TopG thanks to its capability to reduce T_f to the minimum. Requiring only three input features, DyLaClass achieves very high prediction accuracy, making T_f almost negligible. In contrast, TopG requires a large number of features to participate in the prediction. Examples in Table. III also illustrate that even low-complexity feature extractions, such as *cv*, *sd*, *diags*, amount to an astonishing 10 times of the SpMV time T_{spmv}. Additionally, the higher prediction accuracy from DyLaClass allows us to accurately choose the top two GFLOPS corresponding formats. Compared to OCM, although it avoids unnecessary format conversions when the iteration number is low, T_c and T_f are still unavoidable as iterations increase.

In addition to PageRank, we also use BiCGSTAB [36], a commonly used solver, to evaluate the performance of DyLaClass on 531 matrices from a collaborated company, where similar acceleration benefits are observed. In Table. XIII, we

list the number of matrices in each iteration interval and the overall acceleration effect. Overall, DyLaClass consistently outperforms TopG and OCM on average on different CPUs.

D. Stability Analysis

In this section, we focus on analyzing the impact of the previously mentioned issue of fluctuating GFLOPS and assess the stability of the model established by DyLaClass. Note that DyLaClass is designed to solve the problem of varying GFLOPS in matrices with each repetition of SpMV execution, where our research suggests that there is only a most suitable format, not a best format. Therefore, our model aims to accurately hit one of the top two GFLOPS.

1) Stability: To verify whether the model generated by DyLaClass from a single experiment remains effective for subsequent input matrices, we additionally conduct two sets of control experiments on two CPUs with different CPU utilization rates to validate the stability of DyLaClass.

Table XIV shows the mixed and fixed label at CPU utilization of 0-30%, 30%-65%, and 65%-100%. We use Eq.(10) for stability evaluation, the size of δ will intuitively reflect the stability of each method in various control experiments.

Our evaluation shows that in CPU1, labels generated in a single experiment under three types of CPU utilization rates are still among the top two in control experiments over 86% of the time. In CPU2, this ratio reaches an average of 94%. This demonstrates that our proposed method maintains very high stability across multiple experiments. Conversely, using the highest GFLOPS as a criterion for selecting the optimal format, we apply the same evaluation method. We compare the label generated by the highest GFLOPS for a single use with whether it matches the label corresponding to the highest GFLOPS of the matrix in control experiments. The results are obvious: in control experiments, the highest GFLOPS varies significantly, much lower than DyLaClass's stability.

Fig.17 shows the scatter distribution after using DyLaClass on the n-nnz plane uniquely in control experiments. The distribution effect does not always remain unchanged, but through our method, the distribution obtained by merely using the TopG approach become more linear. This is also the reason why DyLaClass can achieve high prediction accuracy by only using few features with low computational complexity. In CPU2 (30%-65%), due to the absence of the fixed label l_3 after DyLaClass's clustering, a category is lost in classification, but this has a very minimal impact on performance because DyLaClass's clustering targets mixed labels. Mixed labels themselves represent the top two formats of the matrix where GFLOPS differ slightly. If this mixed label includes l_3, it means that its format does not significantly differ from another label. Therefore, we can see that using the model from the previous section's experiment for prediction in control experiments, the speedup ratio can still remains stable. However, models using the highest GFLOPS method experience a further decline in the speedup ratio in control experiments.

2) Speedup: In the controlled experiments, we compared the overall performance of DyLaClass with two other methods on two CPUs. Fig.18 displays the speedup ratios for some real matrices in three sets of controlled environments, the average speedup ratios for the three methods in two experiments, and the proportion of speedup distribution for Pagerank under each controlled trial. DyLaClass consistently achieves a stable speedup effect of 1.17 to 1.43 times across various operating environments, with minimal variation from the speedup seen in individual experiments, demonstrating that DyLaClass can achieve very stable acceleration effects in real-world scenarios. However, TopG and OCM showed a declining trend in both prediction accuracy and overall acceleration effects in subsequent controlled experiments. The test results indicate that DyLaClass maintains very satisfactory performance under both types of CPUs.

V. Conclusion

This paper introduces a unique yet comprehensive analysis of the most effective matrix storage format selection for SpMV applications on sparse matrices, and suggests an efficient and dependable solution with optimal accuracy. Building upon traditional approaches, we directly optimize and reduce the overhead of feature extraction, greatly enhancing the efficiency in SpMV applications. In real-world applications, hardware environment changes have a significant impact on the performance of SpMV. We believe that the proposed DyLaClass help many important applications that heavily rely on the efficiency of SpMV in a variety of hardware platforms. Additionally, the unique capability of feature selection enhancement from DyLaClass may shine even brighter in certain application scenarios with prominent features. This paper has only conducted a preliminary study on different CPU platforms, and we plan to explore further in the future on a more complicated mixture of hardware platforms.

Acknowledgment

This research was supported in part by the South China University of Technology Research Start-up Fund No. K3200890. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

[1] Xiaoming Chen, Yu Wang, and Huazhong Yang. *Parallel sparse direct solver for integrated circuit simulation*. Springer, 2017.

TABLE XIII: Overall acceleration effects using TopG, OCM, and DyLaClass in two real-world applications.

App	Matrices	Number of Matrices per Iterations (%)						Speedup in CPU1				Speedup in CPU2			
		0-20	20-40	40-80	80-120	120-480	480+	TopG	OCM	DyLa	Best	TopG	OCM	DyLa	Best
PageRank	2086	11%	17%	41%	13%	10%	8%	1.01	1.17	1.24	1.3	1.15	1.27	1.39	1.44
BiCGSTAB	531	2%	6%	14%	28%	21%	29%	1.07	1.15	1.25	1.33	1.14	1.3	1.45	1.52

TABLE XIV: Control experiments under three CPU utilization rates, along with stability and prediction accuracy analysis.

CPU Usage	No.		FLM				MLM						DyLa		TopG	
			l_1	l_2	l_3	l_4	l_{12}	l_{13}	l_{14}	l_{23}	l_{24}	l_{34}	Stability	Accuracy(%)	Stability	Accuracy(%)
CPU1	$<30\%$	1	244	7	15	15	354	304	196	312	559	79	0.855	81.7	0.603	41.3
		2	220	5	14	13	318	284	178	298	661	97	0.832	80.2	0.525	40.8
	30%-65%	1	242	8	11	19	361	291	158	322	592	83	0.864	82	0.635	41.1
		2	244	13	15	13	350	298	189	283	583	98	0.876	82.5	0.612	40.5
	$>65\%$	1	274	8	48	16	196	524	123	458	190	248	0.674	77.6	0.405	34.2
		2	284	8	27	7	174	691	114	451	171	159	0.65	78.2	0.326	35.7
CPU2	$<30\%$	1	114	14	1	22	187	91	56	271	1305	25	0.931	88.2	0.899	77.6
		2	114	15	1	22	162	90	67	324	1264	27	0.937	89.5	0.903	76.2
	30%-65%	1	112	13	0	25	155	95	50	284	1336	16	0.932	90	0.789	75.4
		2	113	12	0	23	135	95	46	318	1329	15	0.945	89.8	0.786	76.9
	$>65\%$	1	111	88	1	74	353	25	91	181	1147	16	0.829	83.1	0.436	51.2
		2	107	76	0	71	389	25	114	183	1094	26	0.817	81.4	0.283	56.7

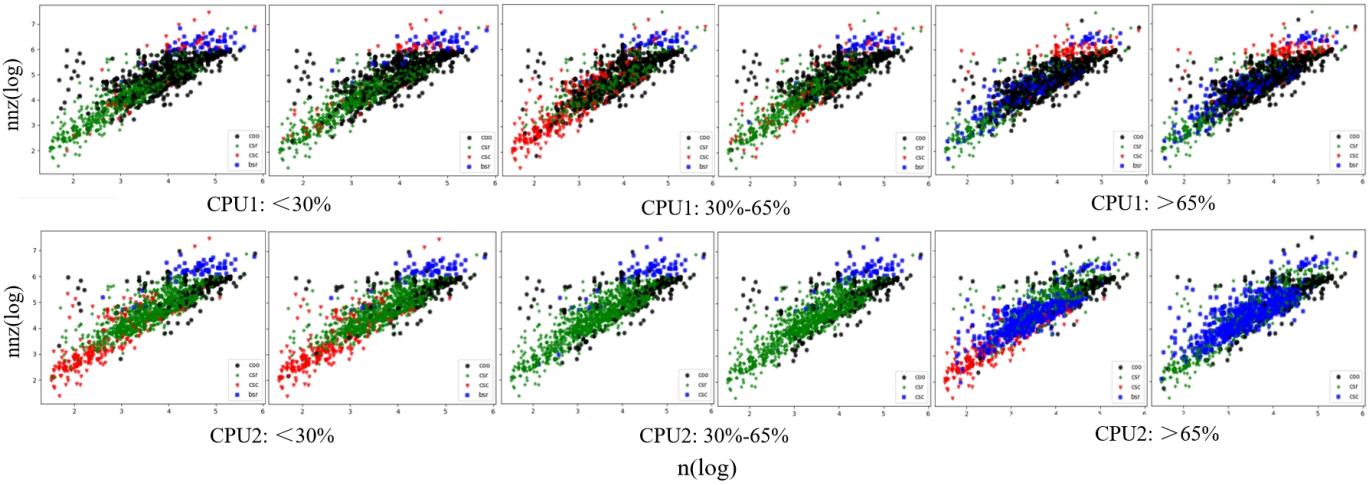


Fig. 17: The scatter distribution of control experiment labels in Table XIV.

[2] Hui Li, Xi Yang, Yang Li, Li-Ying Hao, and Tian-Lun Zhang. Evolutionary extreme learning machine with sparse cost matrix for imbalanced learning. *ISA Transactions*, 100:198–209, 2020.
[3] Ernesto Dufrechou, Pablo Ezzatti, Manuel Freire, and Enrique S. Quintana-Ortí. Machine learning for optimal selection of sparse triangular system solvers on gpus. *Journal of Parallel and Distributed Computing*, 158:47–55, 2021.
[4] Xin Luo, Mengchu Zhou, Shuai Li, Lun Hu, and Mingsheng Shang. Non-negativity constrained missing data estimation for high-dimensional and sparse matrices from industrial applications. *IEEE Transactions on Cybernetics*, 50(5):1844–1855, 2020.
[5] Ruibo Fan, Wei Wang, and Xiaowen Chu. Fast sparse gpu kernels for accelerated training of graph neural networks. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 501–511, 2023.
[6] Yuke Wang, Boyuan Feng, Zheng Wang, Guyue Huang, and Yufei Ding. TC-GNN: Bridging sparse GNN computation and dense tensor cores on GPUs. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 149–164, Boston, MA, July 2023. USENIX Association.
[7] W. Wu, X. Shi, L. He, and H. Jin. Turbomgnn: Improving concurrent gnn training tasks on gpu with fine-grained kernel fusion. *IEEE Transactions on Parallel amp; Distributed Systems*, 34(06):1968–1981, jun 2023.
[8] Hao Li, Kenli Li, Jiyao An, and Keqin Li. An online and scalable model for generalized sparse nonnegative matrix factorization in industrial applications on multi-gpu. *IEEE Transactions on Industrial Informatics*, 18(1):437–447, 2022.
[9] Min Li, Yulong Ao, and Chao Yang. Adaptive spmv/spmspv on gpus for input vectors of varied sparsity. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1842–1853, 2021.
[10] Biwei Xie, Jianfeng Zhan, Xu Liu, Wanling Gao, Zhen Jia, Xiwen He, and Lixin Zhang. Cvr: efficient vectorization of spmv on x86 processors. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018, page 149–162, New York, NY, USA, 2018. Association for Computing Machinery.
[11] Markus Steinberger, Rhaleb Zayer, and Hans-Peter Seidel. Globally homogeneous, locally adaptive sparse matrix-vector multiplication on the gpu. In *Proceedings of the International Conference on Supercomputing*, ICS '17, New York, NY, USA, 2017. Association for Computing Machinery.
[12] Weifeng Liu and Brian Vinter. Csr5: An efficient storage format for cross-platform sparse matrix-vector multiplication. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, ICS '15, page 339–350, New York, NY, USA, 2015. Association for Computing Machinery.
[13] Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11, 2009.
[14] Daniel Langr and Pavel Tvrdík. Evaluation criteria for sparse matrix storage formats. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):428–440, 2016.
[15] Kornilios Kourtis, Vasileios Karakasis, Georgios Goumas, and Nectarios Koziris. Csx: an extended compression format for spmv on shared memory systems. *SIGPLAN Not.*, 46(8):247–256, feb 2011.
[16] Zhen Xie, Guangming Tan, Weifeng Liu, and Ninghui Sun. Ia-spgemm: an input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication. In *Proceedings of the ACM International Conference on Supercomputing*, ICS '19, page 94–105, New York, NY, USA, 2019. Association for Computing Machinery.
[17] Weifeng Liu and Brian Vinter. Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors. *Parallel Computing*, 49:179–193, 2015.

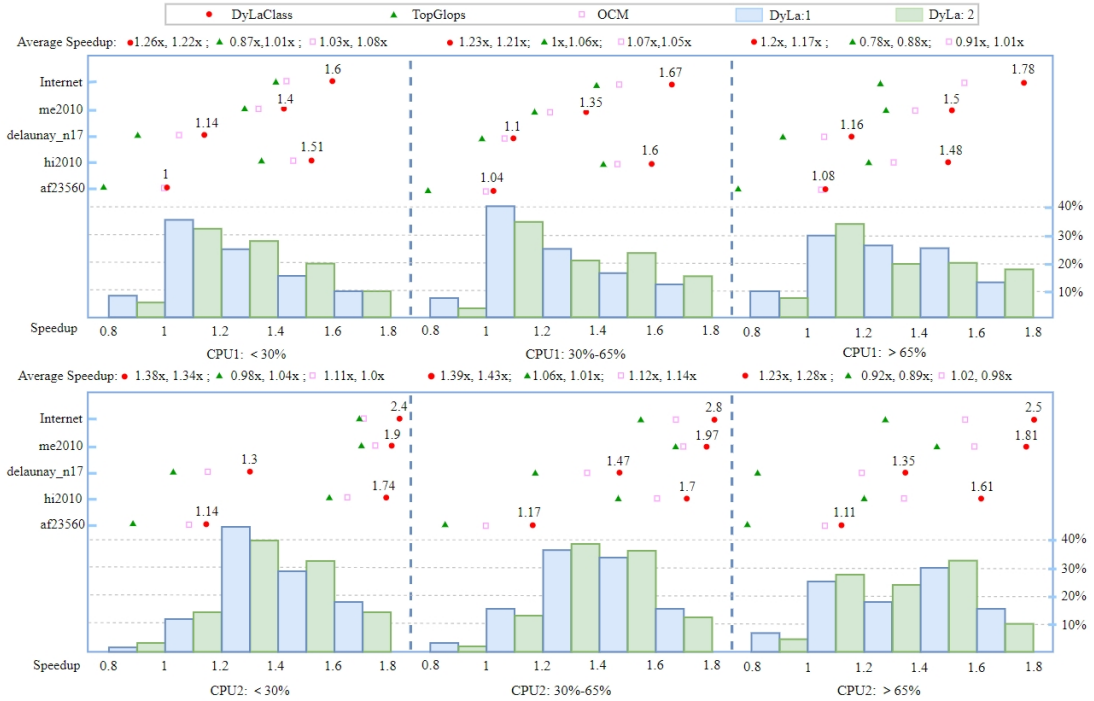


Fig. 18: The overall acceleration effects of TopG, OCM, and DyLaClass on Pagerank across two different CPUs under three CPU utilization rates, DyLa:1 and 2 show the distribution proportion of matrix quantities within each speedup ratio interval in two sets of controlled trials.

[18] Xing Liu, Mikhail Smelyanskiy, Edmond Chow, and Pradeep Dubey. Efficient sparse matrix-vector multiplication on x86-based many-core processors. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, page 273–282, New York, NY, USA, 2013. Association for Computing Machinery.

[19] Richard W. Vuduc and Hyun-Jin Moon. *Fast Sparse Matrix-Vector Multiplication by Exploiting Variable Block Structure*, pages 807–816. HPCC. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[20] Jiajia Li, Guangming Tan, Mingyu Chen, and Ninghui Sun. Smat: an input adaptive auto-tuner for sparse matrix-vector multiplication. *SIGPLAN Not.*, 48(6):117–126, jun 2013.

[21] Duane Merrill and Michael Garland. Merge-based sparse matrix-vector multiplication (spmv) using the csr storage format. PPoPP '16, New York, NY, USA, 2016. Association for Computing Machinery.

[22] Naser Sedaghati, Te Mu, Louis-Noel Pouchet, Srinivasan Parthasarathy, and P Sadayappan. Automatic selection of sparse matrix representation on gpus. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, pages 99–108, 2015.

[23] Akrem Benatia, Weixing Ji, Yizhuo Wang, and Feng Shi. Sparse matrix format selection with multiclass svm for spmv on gpu. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 496–505. IEEE, 2016.

[24] Israt Nisa, Charles Siegel, Aravind Sukumaran Rajam, Abhinav Vishnu, and P Sadayappan. Effective machine learning based format selection and performance modeling for spmv on gpus. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1056–1065. IEEE, 2018.

[25] Yue Zhao, Jiajia Li, Chunhua Liao, and Xipeng Shen. Bridging the gap between deep learning and sparse matrix format selection. In *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, pages 94–108, 2018.

[26] Hang Cui, Shoichi Hirasawa, Hiroyuki Takizawa, and Hiroaki Kobayashi. A code selection mechanism using deep learning. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 385–392, 2016.

[27] Yue Zhao, Weijie Zhou, Xipeng Shen, and Graham Yiu. Overhead-conscious format selection for spmv-based applications. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 950–959, 2018.

[28] Weijie Zhou, Yue Zhao, Xipeng Shen, and Wang Chen. Enabling runtime spmv format selection through an overhead conscious method. *IEEE Transactions on Parallel and Distributed Systems*, 31(1):80–93, 2020.

[29] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[30] The api reference guide for cusparse, the cuda sparse matrix library, 2024.

[31] Arash Ashari, Naser Sedaghati, John Eisenlohr, Srinivasan Parthasarath, and P. Sadayappan. Fast sparse matrix-vector multiplication on gpus for graph applications. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 781–792, 2014.

[32] Joseph L. Greathouse and Mayank Daga. Efficient sparse matrix-vector multiplication on gpus using the csr storage format. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 769–780, 2014.

[33] Markus Steinberger, Andreas Derlery, Rhaleb Zayer, and Hans-Peter Seidel. How naive is naive spmv on the gpu? In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8, 2016.

[34] Buse Yilmaz, Bariş Aktemur, MaríA J. Garzarán, Sam Kamin, and Furkan Kiraç. Autotuning runtime specialization for sparse matrix-vector multiplication. 13(1), mar 2016.

[35] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), dec 2011.

[36] H. A. van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.

Zheng Shi is currently pursuing the Ph.D. degree with College of Microelectronics, South China University of Technology(SCUT), Guangdong, China.

His research interest includes distributed Computing, Calculate acceleration and EDA optimization.

Yi Zou (Senior Member, IEEE) is currently a Chair Professor at the South China University of Technology (SCUT), China. He received the M.Eng. degree from Nanyang Technological University, Singapore in 2002 and Ph. D. in Computer Engineering from Duke University, USA in 2004. Before joining SCUT, he was a postdoctoral fellow at Duke University, USA and a Senior Staff Research Scientist at Research Labs, Intel Corp., USA. He has published more than 70 publications including book chapters, technical papers, and patents. He serves as a frequent program committee member and reviewer for IEEE transactions and conferences. Most recently, he is a TPC member at IEEE NAS'21, ACM SEC'20/21, IEEE DataComp'19, etc. His current research areas of interest are intelligent compute architectures and systems, scale-out memory and storage, data and sensor fusion, edge computing and AI, etc.

Xianfeng Song (Student Member, IEEE) was born in Huzhou, Zhejiang, China. He received the B.S. degree from Nanchang University, Jiangxi, China, in 2015. He is currently pursuing the Ph.D. degree with the College of Microelectronics, South China University of Technology, Guangdong, China.

His research interest includes *Distributed Computing*, *Graph Neural Networks*, and *Distributed Storage*.

Shupeng Li is currently working toward the Ph.D. degree with the School of Microelectronics, South China University of Technology, Guangzhou, China. His research interests include optical wireless communications and deep learning.

Fangming Liu (S' 08, M' 11, SM' 16) received the B.Eng. degree from the Tsinghua University, Beijing, and the Ph.D. degree from the Hong Kong University of Science and Technology, Hong Kong. He is currently a Full Professor at the Huazhong University of Science and Technology, Wuhan, China. His research interests include cloud computing and edge computing, data center and green computing, SDN/NFV/5G, and applied ML/AI. He received the National Natural Science Fund (NSFC) for Ex cellent Young Scholars and the National Program Special Support for Top-Notch Young Professionals. He is a recipient of the Best Paper Award of IEEE/ACM IWQoS 2019, ACM e-Energy 2018 and IEEE GLOBECOM 2011, the First Class Prize of Natural Science of the Ministry of Education in China, as well as the Second Class Prize of National Natural Science Award in China.

Quan Xue School of Electronic and Information Engineering, South China University of Technology, Guangzhou, China. Quan Xue (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees in electronic engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 1988, 1990, and 1993, respectively. In 1993, he joined UESTC as a Lecturer, where he became a Professor in 1997. From 1997 to 1998, he was a Research Associate and then a Research Fellow with The Chinese University of Hong Kong, Shatin, Hong Kong. In 1999, he joined the City University of Hong Kong (CityU), Kowloon, Hong Kong, where he was a Chair Professor of microwave engineering from 2013 to 2017. He was the Associate Vice President of CityU, Innovation Advancement and China Office, from 2011 to 2015, the Director of the Information and Communication Technology Center (ICTC), and the Deputy Director of the State Key Laboratory of Millimeter Waves. He is currently the Dean of the School of Electronic and Information Engineering, South China University of Technology, Guangzhou, China. He has authored or coauthored over 350 internationally refereed journal articles and over 170 international conference articles. He is a coinventor of five granted Chinese patents and 27 granted U.S. patents (five of them have been licensed), in addition to 29 filed patents. His current research interests include microwave/millimeterwave/terahertz passive components, active components, antenna, microwave monolithic integrated circuits, and radio frequency integrated circuits. Dr. Xue served as an AdCom Member for the IEEE MTT-S from 2011 to 2013. He was a recipient of the 2017 H. A. Wheeler Paper Award of the IEEE Antenna and Propagation Society. He was an Associate Editor of the IEEE Transactions on Microwave Theory and Techniques from 2010 to 2013, IEEE Transactions on Industrial Electronics from 2010 to 2015, and IEEE Transactions on Antennas and Propagation from 2016 to 2017 and an Editor of the International Journal of Antennas an