Explaining the Performance of Supervised and Semi-Supervised Methods for Automated Sparse Matrix Format Selection

Sunidhi Dhandhania
Indian Institute of Technology Kanpur
India
sunidhi@cse.iitk.ac.in

Akshay Deodhar
College of Engineering, Pune
India
deodharar17.comp@coep.ac.in

Konstantin Pogorelov
Simula Research Laboratory
Norway
konstantin@simula.no

Swarnendu Biswas
Indian Institute of Technology Kanpur
India
swarnendu@cse.iitk.ac.in

Johannes Langguth
Simula Research Laboratory
Norway
langguth@simula.no

ABSTRACT

The performance of sparse matrix-vector multiplication kernels (SpMV) depends on the sparse matrix storage format and the architecture and the memory hierarchy of the target processor. Many sparse matrix storage formats along with corresponding SpMV algorithms have been proposed for improved SpMV performance. Given a sparse matrix and a target architecture, supervised Machine Learning techniques automate selecting the best formats. However, existing supervised approaches suffer from several drawbacks. They depend on large representative datasets and are expensive to train. In addition, retraining to incorporate new classes of matrices or different processor architectures is just as costly since new training data must be generated by benchmarking many instances. Furthermore, it is hard to understand the results of many supervised systems.

We propose using semi-supervised machine learning techniques for format selection. We highlight the challenges in using the K-Means clustering for the sparse format selection problem and show how to adapt the algorithm to improve its performance. An empirical evaluation of our technique shows that the performance of our proposed semi-supervised learning approach is competitive with supervised methods, in addition to providing flexibility and explainability.

CCS CONCEPTS

• **Mathematics of computing** → **Mathematical software performance**; • **Computing methodologies** → *Learning paradigms.*

KEYWORDS

Sparse matrices, SpMV, ML, performance prediction, clustering

ACM Reference Format:
Sunidhi Dhandhania, Akshay Deodhar, Konstantin Pogorelov, Swarnendu Biswas, and Johannes Langguth. 2021. Explaining the Performance of Supervised and Semi-Supervised Methods for Automated Sparse Matrix Format Selection . In *50th International Conference on Parallel Processing Workshop (ICPP Workshops '21), August 9–12, 2021, Lemont, IL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3458744.3474049

1 INTRODUCTION

Sparse Matrix-Vector (SpMV) multiplication is a linear algebra kernel that is used in many application domains such as graph and data analytics, machine learning, and computational sciences. SpMV is a level-2 Basic Linear Algebra Subprograms operation of the form $\mathbf{y} = A\mathbf{x}$, where $\mathbf{x}$ and $\mathbf{y}$ are dense vectors and A is a sparse matrix. Real-world applications such as PageRank and sparse convolution neural networks invoke the SpMV kernel many times during execution. SpMV has little data reuse, which makes it memory bandwidth-bound, and its irregular data access patterns can further reduce performance. Optimizing an SpMV kernel is *challenging* because the performance depends on the combination of architectural details and the sparsity pattern of each input matrix.

There has been extensive research on improving SpMV performance [1–3, 5, 7, 11, 18, 19, 28, 30, 31, 36, 38, 40]. Several studies have focused on optimizing the memory requirements of sparse matrices by defining sophisticated storage formats that enable the use of advanced architectural features like vectorization [1, 4, 20, 22, 29, 35]. However, *no single format is optimal across all matrices and architectures.* Therefore, it is desirable to select the format that has the best SpMV performance on the target platform for each matrix individually. This is known as the *sparse format selection* problem.

The problem. Recent work has adopted automated techniques that use supervised Machine Learning (ML) models for predicting the best sparse format for a given matrix on a target architecture [3, 24, 28, 30, 38–40]. However, these supervised learning techniques suffer from several limitations in the context of SpMV.

- Supervised techniques assume the availability of *representative* training datasets. The ML models for sparse format selection are often trained on the SuiteSparse matrix collection[1] or similar libraries. Given that these datasets are much smaller than other ML datasets such as ImageNet [10], it is

[1] https://sparse.tamu.edu

not guaranteed that they include *all* possible sparse matrix types. Creating much larger training datasets is a conceivable option for the future, but it exacerbates challenges that we will discuss soon.
- We are currently observing an explosion in the number of hardware platforms, which has been labeled "a new golden age for computer architecture" [12]. Numerical computations for machine learning are performed no longer only by high-performance computing systems but also by a wide variety of low-power devices with vastly different hardware characteristics. Thus, it is important to support the *train once, deploy multiple times* workflow. It is challenging to implement this workflow efficiently for kernels like SpMV since the output of an ML model trained for format selection on one architecture may not be portable *across* architectures. Even small changes such as different algorithm implementations for the same format or even differences in compiler toolchains can affect performance and thus reduce the prediction accuracy of models trained under different parameters. Therefore, it becomes increasingly infeasible to pre-train models for the exact situation under which they are being used.
- Supervised techniques require benchmarking the dataset. Given a training set of M matrices, F sparse matrix formats, and N repetitions to reduce noise, it implies running $M \times F \times N$ experiments. Although each SpMV run may be inexpensive, the total time taken by all runs amount to several days given the overhead of reading from files and format conversion of sparse matrices.

Recent work has trained Deep Learning (DL) models like CNNs for improved classification accuracy [28, 38]. An undesirable side effect is that the DL models require *large* training data to achieve good accuracy and are *expensive* to train. These challenges are further exacerbated since new sparse formats are regularly proposed to optimize for new sparsity patterns and architectural features [4, 20, 22, 35]. This makes repeated benchmarking on large datasets across multiple architectures and retraining the DL models infeasible.

It is desirable to develop a sparse format selection method that addresses the above challenges. Such an automated approach should meet the following requirements:

(1) Given any sparse matrix, the accuracy of the approach should be competitive with existing supervised learning techniques.
(2) The proposed solution should be less tightly coupled to a particular architecture, and the model predictions should be easily portable to different hardware platforms.
(3) Given a matrix with a new sparsity pattern, the approach should be reasonably flexible to incorporate new data.

Existing supervised approaches do not meet the above requirements because they rely on large amounts of labeled data. For the format selection problem, labels denote which format is fastest for a given matrix. Thus, obtaining such a label requires benchmarking a matrix over all the available formats on all the available architectures. It is advisable to move away from black-box approaches toward a more explainable algorithm to overcome these limitations.

Our approach. We propose a semi-supervised approach for the sparse format selection problem based on clustering similar matrices. We extract statistical features from sparse matrices and use popular clustering algorithms such as K-Means for format selection. Preliminary experiments showed that a naïve application of the idea is not competitive with existing supervised models. We analyze the benchmark dataset and use the insights, such as applying logarithmic transformations, to refine the clustering-based format selection model. The performance of the resulting model is comparable with existing supervised approaches. The resulting model is semi-supervised since we still have to assign an optimal format to each cluster. The clusters are largely platform-independent, thereby enabling easy portability of the model across architectures. Furthermore, it is more efficient to merge and split clusters or change their optimal format when new sparse matrices are added to the dataset, especially compared to retraining large DL models.

Contributions. The contributions of this work are as follows.
- To the best of our knowledge, we present the first semi-supervised approach for the format selection problem that is also more explainable than most supervised models.
- We perform an extensive empirical evaluation of our approach and compare it to different supervised models. Unlike previous work in the area, we assess performance using measures that reflect the multi-class nature of the problem. Our results show that the semi-supervised approach delivers competitive performance while avoiding the problems of the supervised models.

2 BACKGROUND AND MOTIVATION

In this section, we discuss sparse matrix formats, the associated format selection problem, and the challenges with existing work.

2.1 Sparse Matrix Storage Formats

Sparse matrices for many applications can be very large. Sparse matrix storage formats save space by not storing zero elements. A number of sparse storage formats have been proposed over the years [1, 2, 4, 7, 20, 22, 29, 35], and several high-performance libraries like Intel MKL [14], and CUSP [25] and cuSPARSE [26] from NVIDIA provide support for the more popular storage formats.

The *coordinate* (COO) format stores the matrix in three dense arrays of length *NNZ* (number of nonzeros) called row, column, and value. The position of every nonzero value in the matrix is given explicitly. The *compressed sparse row* (CSR) format, which is the most popular format, compresses the row array to store the start positions of all rows in the corresponding column and value arrays. Formats like CSR and COO are *general*, i.e., they require $O(n)$ space for matrices with n nonzeros. The *ELLPACK* (ELL) format stores a sparse matrix A as a dense rectangular matrix by shifting the nonzeros in each row to the left and zero-padding all rows that have fewer nonzeros than the maximum. The storage size of ELL thus depends on the maximum number of nonzeros in a row of A, which is problematic for matrices with a large deviation in the number of nonzeros per row. The *hybrid* (HYB) format alleviates this problem by using ELL for storing most of the matrix A and COO to store additional entries in rows with many nonzeros. This reduces the required amount of padding while maintaining some advantages of

Feature	Description
$nrows$	Number of rows
$ncols$	Number of columns
nnz	Number of nonzeros
nnz_frac	Fraction of nonzeros
nnz_mu	Average number of nonzeros per row
nnz_min	Minimum number of nonzeros per row
nnz_max	Maximum number of nonzeros per row
nnz_sig	Standard deviation of nonzeros per row
max_mu	Difference between nnz_max and nnz_mu
mu_min	Difference between nnz_mu and nnz_min
csr_max	Maximum number of rows a particular warp will process in the CSR kernel
sig_lower	Root mean square (RMS) of difference between row nonzero counts which are less than nnz_mu
sig_higher	RMS of difference between row nonzero counts which are greater than nnz_mu
hyb_ell_size	Size of ELL structure in HYB representation of the matrix
hyb_coo	Number of nonzeros in COO part of the HYB representation
hyb_ell_frac	Number of nonzeros stored in ELL part of HYB representation
$diagonals$	Number of diagonals which are not empty
dia_size	Number of entries which will be stored in the DIA structure
dia_frac	Fraction of entries in DIA which will be true nonzeros
ell_frac	Fraction of true nonzero entries in the ELL structure
ell_size	Size of the ELL structure in ELL format

Table 1: Sparse matrix features popularly used for automated format selection.

ELL. Other formats, like *diagonal* (DIA), take advantage of specific sparsity patterns but can also take $O\left(n^2\right)$ space in the worst case. More recent proposals aim to exploit microarchitectural features like vectorization and the compute capabilities of GPUs [4, 20, 22, 35]. We refer the reader to existing literature for an in-depth discussion [1, 11, 29].

2.2 Automated Sparse Matrix Format Selection

The performance of an SpMV kernel is sensitive to the size and the sparsity pattern of the input matrix and the target architecture. Although CSR is general and is the most-used format, there is *no single format that suits all sparsity patterns and target architectures* [30]. Slowdowns amounting to two orders of magnitude from suboptimal sparse formats are often observed in practice. For example, we observe a maximum slowdown of 194.85X when using CSR with the *mawi_201512012345* matrix from the SuiteSparse Matrix Collection [9] on an NVIDIA Quadro RTX 8000 GPU, for which HYB is the optimal format. To avoid such slowdowns, a large body of work has focused on *automatically* predicting the best sparse storage format for a given sparse matrix [3, 18, 19, 24, 28, 30, 31, 38–40].

Automated sparse storage format selection requires identifying features that help differentiate classes of sparse matrices [3]. Some aspects of SpMV performance are easily understood by analyzing the suitability of the features to the properties of the different formats. For example, ELL can be efficient if all the matrix rows have a similar number of nonzeros. However, manually designing heuristics is error-prone, since it can be hard to generalize rules across a variety of inputs and target platforms. This fact, together with the recent advances in Machine Learning (ML), has spurred work on using sophisticated models to learn the performance characteristics of SpMV kernels. Format-specific SpMV kernels are benchmarked with many input matrices across different formats. Supervised ML models are trained offline on the benchmark results to predict the SpMV performance of input matrices for each format. Based on these predictions, the system recommends the best format to use for a given matrix. The ML models can be either regression or classification based, and the performance and accuracy of these techniques depend on the predictive power of the models. Prior work has used Decision Trees and Random Forests [30], Support Vector Machines [3], and XGBoost models for automated format selection. Convolutional Neural Networks (CNNs) are popular deep learning models used for image classification. Recent work employs CNN-based deep learning (DL) models for format selection by encoding the sparse matrix as an image and showcase good predictive accuracy [28, 38].

Table 1 shows statistical features that are commonly used by existing non-DL supervised-learning-based approaches. Note that feature identification in SpMV mostly ignores domain-specific knowledge, since that would restrict the approach's applicability.

2.3 Motivation

There are several challenges in optimizing the performance of an SpMV kernel, which depend on several factors, such as the sparsity pattern of the matrix and the target architecture. Since computing platforms are becoming increasingly heterogeneous, an application may be run on many architectures, each having characteristics that affect the performance of different SpMV formats. However, most existing work proposes techniques that focus on automating the format selection problem for a *single* architecture. For example, ML-based techniques usually run profiles, train, and expect the inference to be run on the same architecture. But the profile data and hence the labels in supervised learning *differ* significantly across architectures. Most prior techniques ignore the challenges in predicting optimal storage format for *any given* architecture. A naïve way to deal with this problem is to build models on every architecture, but the need for extensive training, possibly under different tuning parameters, is computationally expensive and can become intractable.

Given the overhead of benchmarking and training CNN models, prior work briefly discusses transfer learning strategies for CNNs [38]. They test a single transfer, i.e., Intel to AMD CPUs, and conclude that transfer learning can be used for SpMV problems. However, training a CNN model is very costly. In our test environment, we use 5-fold cross-validation for training. The training time on a GeForce GTX1080 GPU is around 3 hours for one split. With 5 splits, it takes ∼15 hours to get one set of results.

3 LEARNING SPMV PERFORMANCE

Prior work uses supervised models to improve SpMV performance where sparsity patterns are learned from training data that is annotated with ground truth labels. In contrast, unsupervised learning methods such as clustering can capture new sparsity patterns in an architecture-independent way. Hence, clustering provides a significant benefit toward model portability if we can ensure that most matrices in a cluster *actually* have the same optimal format. A clustering-based approach still requires cluster labels because it is not sufficient to determine that a cluster of matrices is similar. A cluster label is the optimal format for the cluster. Thus, the method is semi-supervised. The cluster labels might differ between the platforms, but due to the clustering, we only need to benchmark a *few* matrices to retrain the system for a new architecture (one matrix per cluster in an ideal case). Small clusters will be more accurate, but at the same time, clusters should be as large as possible to save on training time. The use of clustering for portability improves upon supervised learning models where the prediction accuracy might suffer when the target architecture changes. One can think of the supervised approaches as having a cluster size of one. We elaborate on the training of semi-supervised learning in the following section.

Enabling porting of SpMV model results. Sparse format prediction consists of two stages: training an offline model and inference of the predicted optimal format. Training a format predictor involves benchmarking a training set of sparse matrices across different sparse formats for a given platform. Given the sensitivity of SpMV performance to the architecture, the optimal format for the same sparse matrix is *often* different on different architectures. Thus, existing models for sparse matrix format selection cannot be assumed to be portable across architectures. An ML model may perform sub-optimally when deployed on a platform different from the training platform. For example, we trained an XGBoost classifier using profile information on the NVIDIA GeForce GTX 1080 platform, where the classifier gives high accuracy (90.65%) (Table 6). However, when the same classifier is used on the NVIDIA Volta V100 platform, accuracy drops to 71.03% (Table 7). The speedup (compared to running all matrices in the CSR format) drops from 1.07X to 0.97X, which means that using the model in that setting has no practical value.

For a pre-trained model to be accurate on a new architecture, it needs to *learn* some characteristics of the new architecture and predict formats accordingly. For example, ELL provides memory access coalescing at the cost of a larger memory footprint. It stores k entries for all rows in the sparse matrix, where k is the maximum number of nonzeros among all rows of the matrix. This can result in many padding entries for matrices where there is a significant variation in the number of nonzeros in a row. On the Quadro RTX 8000 (see Section 5.1), there is sufficient memory to accommodate ELL structures for large matrices. On GPUs with smaller memory, such as GeForce GTX 1080, storing a large ELL matrix may not be possible. Furthermore, implementations like CUSP use a single thread to process a particular row of each matrix. Thus, the relative efficiency of row-based formats such as CSR and ELL versus COO may depend on the number of threads that are available on the GPU.

4 SEMI-SUPERVISED LEARNING

Unlike prior work, we explore semi-supervised learning to solve the portability problem of SpMV format selection. The idea is to cluster matrices such that *the matrices in each group will have the same optimal format irrespective of the architecture.* Given such a clustering, a transfer learning scheme would have to ideally benchmark only one matrix from each cluster on the target platform to provide full prediction accuracy. The clusters are formed using the matrices from the training set. For predicting the format of matrix M, the format assigned to the cluster whose centroid is closest to the data point associated to M will be assigned to M when using centroid-based clustering. Unlike in the supervised methods, a fraction of matrices from each cluster can be benchmarked *after* the clusters have been formed. Clusters are assumed to be invariant across platforms, while the assignment of labels is platform-specific. Naturally, this requires a relatively fine-grained clustering. Note, the strategy is not limited to using any particular clustering algorithm.

Creating clusters. The main challenge in creating clusters is to define a distance metric that quantifies the similarity of two sparse matrices with respect to performance on any given architecture. To compare different metrics, we first define the *purity* of a cluster c as

$$purity\,(c) = \frac{\max_f \; count\,(c, f)}{|c|}$$

where $count\,(c, f)$ is the number of matrices in cluster c having f as the optimal format, and $|c|$ represents the total number of matrices in c. For effectively using clustering for format selection, we need to create clusters with high purity.

The K-Means clustering algorithm is suitable for creating these clusters, with the statistical features forming the feature space for K-Means. As there are no *inherent* clusters in the space of statistical features, a centroid-based clustering algorithm is appropriate.

Computing the values for the features listed in Table 1 requires traversing the entire matrix, thus taking time proportional to the number of nonzeros. If the features related to diagonal count are dropped, for a matrix in CSR format, the features can be computed in time proportional to the number of rows. We have chosen only features that can be computed in time proportional to the number of nonzeros, so calculating these for a sparse matrix dataset is inexpensive. Moreover, these features are completely *invariant* across architectures, so they have to be computed only once. However, a naïve application of a clustering algorithm with the features shown in Table 1 does not work well. This is due to the distribution of the statistical features—due to the varying sizes and nonzero distributions of sparse matrices, there will be matrices that have unusually high values for some features such as *nnz*, *nnz_max*, or *nnz_mu*. Since basic clustering algorithms such as K-Means exclusively use Euclidean distance as the similarity metric, this results in the formation of small clusters containing outliers and *impure* clusters containing matrices that are not similar. An examination of the distribution of feature values reveals that some features follow a power-law distribution. Applying the *log* transformation to these features before clustering gave clusters with fairly uniform sizes and high purity.

In our approach, a *log* transform or a *square root* transform is applied to all features which have a sparse distribution (irrespective of whether they have a power-law distribution). Afterward, min-max scaling is used to scale each feature to a range of $[0, 1]$. This is essential when performing classification based on a distance metric, unlike tree-based classifiers, which do not depend on such scaling. We then use Principal Component Analysis (PCA) to decompose the features to a feature vector of size 8. We have thus created a feature space where the Euclidean distance between two matrix datapoints is correlated with their similarity. The position of each matrix in the feature space is then used as an input to the clustering algorithm. Many clustering algorithms such as K-Means require giving a number of clusters K, which presents a tradeoff. Having more small clusters will increase accuracy, while having fewer large clusters reduces training time and limits the risk of overfitting. We

μ-architecture	Pascal	Volta	Turing
Model	GTX 1080	V100 SXM3	RTX 8000
# of SMs	20	80	72
L1 cache per SM	48 KiB	128 KiB	64 KiB
L2 cache	2048 KiB	6144 KiB	6144 KiB
Memory (GB)	8 (GDDR5)	32 (HBM2)	48 (GDDR6)
Memory bandwidth	320 GB/s	897 GB/s	672 GB/s

Table 2: Different NVIDIA GPUs used in our experiments.

	Pascal	Volta	Turing	Common Subset		
				Pascal	Volta	Turing
COO	92	4	415	79	4	15
CSR	6019	4417	6629	4008	4138	4671
ELL	2796	2126	1721	1868	1968	1399
HYB	217	3	40	158	3	28
Total	**9124**	**6550**	**8805**		**6113**	

Table 3: Distribution of the best sparse formats across GPUs.

ran a series of preliminary experiments to determine a good choice of K for each clustering algorithm and architecture.

The fact that K-Means and other clustering algorithms use Euclidean distance as a similarity metric suggests that a KNN predictor which uses the same feature set and the same preprocessing transformations should also be competitive. We test this idea in Section 5.

Of course, we cannot assume the clustering to be perfectly pure, i.e. result in all matrices in a cluster having the same best format. To deal with impure clusters, it is beneficial to benchmark multiple matrices from each cluster and apply a decision rule such as *majority voting* to select the format label for each cluster.

Example. Suppose there are 10 matrices in a particular cluster, of which the ELL format is optimal for 9 on a Turing GPU, while CSR is optimal for the remaining. On a Pascal GPU, the CSR format might be optimal for 8 of the matrices in the same cluster and HYB for the remaining 2. We benchmark a single matrix from the cluster when setting up the predictor for the Turing GPU. With 90% likelihood, it will vote for ELL as its format, and it will then classify 9 out of 10 matrices correctly. If it votes for CSR instead, it will only classify 1 out of 10 matrices correctly, for a total prediction accuracy of 82%. For the Pascal GPU, the same cluster would have a 68% accuracy, which clearly illustrates the importance of high-purity clusters. If two matrices are benchmarked in the latter case, the likelihood of picking the correct label, i.e., CSR, rises to 96% and the accuracy to 78%, which is close to the upper bound set by the purity of the cluster.

Clustering methods. As we have seen, the semi-supervised approach depends on clustering in combination with a classification algorithm, and its performance can be sensitive to the chosen method. For that reason, we implement and test our approach with a variety of clustering algorithms, including the well-known **K-Means** [16], as well as **Mean-Shift** [8] and **Birch** clustering [37]. For each clustering algorithm, we test three different classification algorithms: **Majority Vote** (VOTE), **Logistic Regression** (LR), and **Random Forest** (RF). This gives us a total of nine combinations that are used in the experiments. As these techniques in themselves are well-established, we do not explain their details here and refer the reader to their respective sources.

5 EVALUATION

In this section, we present a comprehensive evaluation of our proposed semi-supervised approach and compare it to state-of-the-art supervised techniques for automated sparse format selection.

5.1 Experimental Setup

Platform. We run our experiments on three different NVIDIA GPU platforms, namely the older Pascal and the more recent Turing and Volta architectures. The NVIDIA GeForce GTX 1080 (Pascal) is a desktop card intended for gaming. NVIDIA Quadro RTX 8000 (Turing) is a workstation card, and NVIDIA Volta V100 (Volta) is a GPU intended for high-performance computing. Table 2 shows technical details of the three GPUs. We will refer to the GPUs by their architecture names.

Implementation. We have reimplemented several existing automated sparse matrix format selection approaches that use supervised learning [3, 30, 38]. We use the scikit-learn library [27] to implement classifiers based on Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and XGBoost models. Each supervised algorithm uses an optimized subset of the features from Table 1. The input features are selected based on the best performance for that method. We mostly use the default hyper-parameters suggested by the library, excepting the following changes that improve the performance of the models. For RF, we use 100 estimators with a maximum depth of 6. For XGBoost, we set a learning rate of 0.1 and the number of rounds to 100. We have reimplemented the publicly available convolutional neural network model (CNN) from prior work [38] using the TensorFlow v2 framework. We use the NVIDIA CUDA toolkit v9.2 and the NVIDIA CUSP [25] libraries for obtaining the SpMV benchmark results.

Benchmarks. We test all models using sparse matrices from the SuiteSparse Matrix Collection [9]. Due to limited memory, very large matrices cannot be run on some GPUs, and they are omitted. We also omit matrices where the CUSP library failed to generate the ELL variant because of restrictions on the size (noted by prior work [3]). This leaves a total of 1929 matrices from the collection, which execute successfully across all the GPU platforms. To effectively train the CNN model, we derived additional instances from the SuiteSparse matrices by performing simple row and column permutations similar to prior work [28, 38]. We thus generated an augmented dataset combining the original SuiteSparse and the permuted matrices.

We limit benchmarking to four sparse formats, namely CSR, COO, ELL, and HYB since the implementations are readily available as part of the CUSP library [25]. Prior work also use these formats because of their popularity and generality and the availability of high-performance libraries. Evaluating sparse format implementations from other sources is not a fair comparison because

	Pascal				Volta				Turing			
Algorithm:	NC	MCC	ACC	F1	NC	MCC	ACC	F1	NC	MCC	ACC	F1
K-Means-VOTE	400	0.422	0.749	0.726	100	**0.448**	0.770	0.740	300	0.629	**0.882**	**0.877**
K-Means-LR	200	0.312	0.712	0.677	100	0.388	0.750	0.707	150	0.537	0.86	0.845
K-Means-RF	400	0.404	0.735	0.719	100	0.45	**0.771**	**0.742**	200	**0.631**	0.875	0.873
Mean-Shift-VOTE	32	0.154	0.673	0.554	30	0.169	0.687	0.574	30	0.137	0.792	0.710
Mean-Shift-LR	32	0.128	0.671	0.553	30	0.152	0.685	0.570	30	0.111	0.790	0.705
Mean-Shift-RF	32	0.145	0.672	0.554	30	0.170	0.687	0.575	30	0.145	0.793	0.713
Birch-VOTE	400	**0.435**	**0.753**	**0.732**	400	0.44	0.767	0.736	150	0.622	0.881	0.874
Birch-LR	150	0.289	0.709	0.643	50	0.332	0.727	0.659	100	0.354	0.822	0.777
Birch-RF	400	0.404	0.738	0.719	150	0.441	0.768	0.74	200	0.628	0.879	0.874

Table 4: Performance of the semi-supervised approach using different clustering algorithms on different GPUs. The best performance for each architecture is marked in bold. NC is the number of clusters generated by the algorithms.

of implementation differences. The time measured for each kernel and input matrix is the average over 100 trials. The benchmark results serve as labels (i.e., ground truth) for training the ML models. Columns 2–4 in Table 3 show the final number of matrices used for each GPU architecture. The *Common Subset* columns indicate the overlapping set of matrices that executed successfully on all three GPUs and formed the basis of our transfer learning experiments.

Training the Classifiers. We perform a series of experiments that compare the different sparse format classification strategies. The classifiers are trained and evaluated on the *same augmented* benchmark data (Table 3). All experiments are performed with 5-fold cross-validation to reduce the possibility of overfitting, and average results are reported for the different metrics. As shown in Table 3, the classes obtained are highly unbalanced, with the majority of matrices having CSR as the best format.

5.2 Testing the Semi-Supervised Approach

We evaluate the semi-supervised method by training and testing on each of the three GPU architectures using all 9 clustering algorithms. Table 4 shows the **MCC** score, the **ACC**uracy, and the **F1** score. Prior work mostly use ACC and F1, we also use *Matthew's correlation coefficient* (MCC) since the classes are highly unbalanced. MCC is a statistical rate that produces a high score only if the predictions obtained good results in all the cells of the confusion matrix, proportional to the number of elements in each class of the dataset [23]. MCC is widely regarded as a superior metric, especially for multiclass problems [6]. In addition, **NC** gives the number of clusters used. Note that *Mean-Shift* does not take NC as input but determines the number of clusters automatically. We observe that all variants of the *Mean-Shift* algorithm perform poorly while *Birch-VOTE*, *K-Means-VOTE*, and *K-Means-RF* perform well. The main reason for this seems to be the fact that *Mean-Shift* finds many clusters which are too small to capture meaningful differences in performance, while the other algorithms are given a sufficient number of clusters. Interestingly, the quality of prediction is much better for the Turing GPU compared to the other architectures.

Our next experiment evaluates the accuracies of the different alternatives in the *transfer* setting, which means that the training and test platforms differ. Since we have three GPUs, this gives us six different combinations of transfer source and target architecture. Furthermore, we show results for three different amounts of retraining. In the first case, no retraining on the test architecture has been performed, while in the second and third cases, 25 and 50% of the training data were used, respectively. Table 5 shows the results of this portability experiment. Here, the difference between the clustering methods is even more pronounced, with either *K-Means-VOTE* or *K-Means-RF* being the best alternative in all cases.

Among the different architectures, the predictions for the Turing GPU are much more accurate than those for the other GPUs, even when the system was only trained on a different GPU and no retraining occurred. The higher accuracy compared to the single device results seems to be because the test set is smaller. Based on the numbers in Table 3, the most likely reason for this behavior is the fact that Turing has the highest number of CSR instances, and the system tends to overpredict CSR. When using the better clustering algorithm, the classification performance in the transfer case is relatively high without retraining. On the other hand, additional retraining only provides a moderate increase in performance.

5.3 Comparison to Supervised Classifiers

We study the performance of all supervised classifiers in the *local* setting (i.e., training and inference on the same platform). Table 6 summarizes the results. Columns 2–4 show the *accuracy* (ACC), *F1 score* (F1), and MCC metrics. In addition, the *GT* column shows the speedup from the model predictions compared to an oracle scheme, which always makes the correct prediction. Consequently, all entries are 1 or lower. The *CSR* column shows the speedup achieved over the strategy of always using the CSR format as the default. Values in both columns represent the geometric mean over all the matrices. The column *Threshold* shows the number of matrices that experience a significant slowdown of ≥1.5X over the CSR baseline due to mispredictions; lower values indicate better classification performance. All the floating-point values are rounded to two points after the decimal. The best result in each column is emphasized, and we break ties using a higher precision not shown in the table.

Based on the MCC score, we observe that both Random Forest (RF) and XGBoost perform well, while KNN, Decision Tree (DT), SVM, and CNN models show somewhat weaker results. The CNN model [38] has good results for the Turing platform, but its

			0% Training Data			**25% Training Data**			**50% Training Data**		
	Algorithm	NC	MCC	ACC	F1	MCC	ACC	F1	MCC	ACC	F1
Pascal to Turing	K-Means-VOTE	1250	0.605	0.866	0.870	**0.638**	**0.881**	**0.880**	**0.645**	**0.882**	**0.886**
	K-Means-LR	125	0.582	0.872	0.861	0.592	0.875	0.863	0.576	0.870	0.859
	K-Means-RF	200	**0.630**	**0.873**	**0.872**	0.642	0.873	0.874	0.645	0.875	0.875
	Mean-Shift-VOTE	32	0.197	0.798	0.724	0.210	0.799	0.726	0.185	0.797	0.723
	Mean-Shift-LR	32	0.188	0.798	0.725	0.201	0.799	0.727	0.103	0.790	0.707
	Mean-Shift-RF	32	0.194	0.799	0.727	0.206	0.799	0.729	0.180	0.797	0.724
	Birch-VOTE	175	0.593	0.864	0.866	0.610	0.878	0.871	0.632	0.880	0.879
	Birch-LR	100	0.482	0.849	0.825	0.544	0.862	0.847	0.520	0.857	0.839
	Birch-RF	200	0.611	0.872	0.869	0.613	0.879	0.870	0.643	0.855	0.862
Pascal to Volta	K-Means-VOTE	125	0.365	0.726	0.724	0.426	0.760	0.731	0.432	0.766	0.739
	K-Means-LR	125	0.392	0.753	0.724	0.393	0.751	0.714	0.403	0.758	0.724
	K-Means-RF	125	**0.432**	**0.767**	**0.743**	**0.445**	**0.770**	**0.740**	**0.449**	**0.774**	**0.747**
	Mean-Shift-VOTE	32	0.185	0.691	0.582	0.201	0.692	0.587	0.197	0.696	0.591
	Mean-Shift-LR	32	0.197	0.694	0.588	0.209	0.694	0.591	0.202	0.697	0.593
	Mean-Shift-RF	32	0.205	0.695	0.591	0.216	0.695	0.593	0.209	0.699	0.596
	Birch-VOTE	125	0.361	0.732	0.720	0.401	0.751	0.726	0.418	0.761	0.725
	Birch-LR	80	0.370	0.744	0.698	0.402	0.754	0.715	0.368	0.745	0.696
	Birch-RF	100	0.388	0.748	0.733	0.430	0.765	**0.740**	0.413	0.760	0.722
Turing to Pascal	K-Means-VOTE	1750	**0.440**	**0.759**	**0.730**	**0.460**	**0.766**	**0.736**	**0.462**	**0.769**	**0.739**
	K-Means-LR	150	0.324	0.719	0.675	0.339	0.724	0.672	0.344	0.728	0.677
	K-Means-RF	300	0.395	0.722	0.712	0.402	0.733	0.718	0.403	0.735	0.720
	Mean-Shift-VOTE	30	0.123	0.669	0.545	0.118	0.667	0.543	0.130	0.672	0.551
	Mean-Shift-LR	30	0.083	0.664	0.535	0.089	0.664	0.535	0.090	0.668	0.540
	Mean-Shift-RF	30	0.109	0.668	0.543	0.107	0.666	0.541	0.114	0.671	0.548
	Birch-VOTE	1750	0.428	0.755	0.721	0.380	0.737	0.696	0.393	0.744	0.704
	Birch-LR	100	0.250	0.698	0.628	0.238	0.693	0.605	0.344	0.728	0.671
	Birch-RF	100	0.300	0.708	0.670	0.332	0.720	0.679	0.375	0.729	0.709
Turing to Volta	K-Means-VOTE	2000	**0.472**	**0.780**	**0.765**	**0.461**	**0.774**	**0.759**	**0.467**	**0.779**	**0.764**
	K-Means-LR	100	0.361	0.742	0.697	0.399	0.754	0.718	0.379	0.749	0.706
	K-Means-RF	100	0.419	0.761	0.726	0.443	0.769	0.740	0.438	0.769	0.736
	Mean-Shift-VOTE	30	0.158	0.686	0.571	0.161	0.685	0.569	0.171	0.691	0.577
	Mean-Shift-LR	30	0.113	0.681	0.559	0.132	0.681	0.560	0.135	0.686	0.567
	Mean-Shift-RF	30	0.163	0.688	0.575	0.168	0.686	0.573	0.175	0.692	0.580
	Birch-VOTE	2000	0.410	0.758	0.733	0.416	0.758	0.721	0.408	0.758	0.717
	Birch-LR	80	0.295	0.722	0.671	0.364	0.740	0.691	0.361	0.742	0.687
	Birch-RF	100	0.375	0.746	0.701	0.398	0.752	0.711	0.403	0.756	0.713
Volta to Pascal	K-Means-VOTE	1750	**0.402**	**0.746**	0.700	**0.425**	**0.753**	0.712	**0.449**	**0.765**	**0.730**
	K-Means-LR	150	0.349	0.728	0.682	0.355	0.729	0.679	0.328	0.723	0.673
	K-Means-RF	250	0.390	0.739	**0.710**	0.403	0.730	**0.717**	0.408	0.733	0.721
	Mean-Shift-VOTE	30	0.115	0.668	0.545	0.116	0.667	0.543	0.119	0.672	0.549
	Mean-Shift-LR	30	0.097	0.666	0.540	0.095	0.664	0.537	0.114	0.671	0.548
	Mean-Shift-RF	30	0.110	0.668	0.544	0.106	0.666	0.541	0.114	0.671	0.548
	Birch-VOTE	1750	0.352	0.729	0.675	0.371	0.734	0.684	0.371	0.737	0.689
	Birch-LR	100	0.272	0.704	0.631	0.276	0.704	0.636	0.300	0.715	0.655
	Birch-RF	175	0.370	0.725	0.705	0.402	0.735	0.718	0.372	0.728	0.707
Volta to Turing	K-Means-VOTE	1750	**0.670**	**0.896**	**0.889**	**0.708**	**0.906**	**0.901**	**0.724**	**0.912**	**0.907**
	K-Means-LR	150	0.577	0.869	0.859	0.570	0.868	0.856	0.583	0.870	0.861
	K-Means-RF	175	0.631	0.876	0.874	0.646	0.882	0.879	0.651	0.886	0.882
	Mean-Shift-VOTE	30	0.145	0.794	0.714	0.141	0.793	0.713	0.120	0.792	0.710
	Mean-Shift-LR	30	0.121	0.793	0.710	0.114	0.791	0.707	0.095	0.791	0.705
	Mean-Shift-RF	30	0.141	0.794	0.714	0.138	0.793	0.712	0.117	0.792	0.710
	Birch-VOTE	1750	0.575	0.871	0.857	0.607	0.879	0.867	0.614	0.881	0.871
	Birch-LR	100	0.460	0.844	0.816	0.494	0.850	0.829	0.549	0.864	0.849
	Birch-RF	200	0.569	0.835	0.841	0.632	0.849	0.856	0.602	0.869	0.866

Table 5: Comparison of the effectiveness in sparse matrix format selection of different automated techniques using transfer learning across different GPUs. The best values in each scenario are marked in bold. NC is the number of clusters used.

	MLM	ACC	F1	MCC	GT	CSR	Thresh.
Pascal	DT	84.86	0.84	0.67	0.97	1.06	172
	RF	85.94	0.85	0.69	0.98	1.06	124
	SVM	81.75	0.81	0.59	0.97	1.05	161
	KNN	89.47	0.90	0.78	0.98	1.07	86
	XGBoost	**90.65**	**0.91**	**0.80**	**0.99**	**1.07**	**79**
	CNN	77.79	0.85	0.52	0.90	1.02	466
Volta	DT	79.66	0.78	0.52	0.96	1.04	**199**
	RF	**80.58**	**0.79**	**0.54**	**0.96**	**1.04**	201
	SVM	79.66	0.77	0.52	0.96	1.04	210
	KNN	72.95	0.73	0.39	0.95	1.02	260
	XGBoost	80.23	0.78	0.53	0.96	1.04	200
	CNN	66.32	0.76	0.20	0.91	1.02	248
Turing	DT	94.36	0.94	0.83	0.99	1.05	17
	RF	95.04	0.95	0.85	1	1.05	**11**
	SVM	93.85	0.94	0.81	0.99	1.04	21
	KNN	94.81	0.95	0.85	0.99	1.05	15
	XGBoost	**95.62**	**0.96**	**0.87**	**1**	**1.05**	**11**
	CNN	90.45	0.94	0.72	0.98	1.04	14

Table 6: Performance of ML models on different GPUs.

performance drops significantly for the Pascal and Volta architectures , which we believe is due to the composition of the training dataset. Among the three architectures, we observe that the MCC scores for Volta are far lower than for Pascal or Turing. Furthermore, the MCC scores for the CNN are weak. The CNN model attains 90.45% accuracy for the Turing GPU, which is close to prior work [38], while the results for the other two architectures are much poorer. We believe these differences arise due to the differences in datasets and due to the known difficulty CNNs face with unbalanced training sets.

Table 7 shows the results for the *transfer* case for the supervised classifiers. All supervised models are trained on their full training set. Note that we use a subset of matrices that is common across all the architectures (Table 3). We evaluate them when directly transferred to the target architecture, i.e., with 0% retraining, as well as 25 and 50% retraining on the target architecture. The 0/25/50% retraining results show the attainable prediction performance after retraining with part of the training data of the target architecture. We omit evaluating the CNN model since it has poor performance on the Pascal and Volta architectures, and each experiment takes $\sim$ 15 hours to complete. Following the structure of Table 5, there are six possible combinations of the *transfer* scenario with three GPUs. However, due to lack of space, we omit Volta to Pascal as it is very similar to the Turing to Pascal case. Across the architectures and classifiers, we observe a performance improvement when going from 0 to 25%, and in some cases also from 25 to 50%. The improvement is greater than in the semi-supervised case, indicating that the supervised methods *depend more* on retraining. While the accuracy scores are reasonably high in this scenario, the MCC scores are noticeably lower than those presented in Table 6. This indicates that in the transfer case, matrices belonging to the smaller classes are often misclassified.

The classification performance on Volta is far lower than on the other two architectures in the non-transfer scenario. This is still true in the transfer case, although the difference is much less pronounced. However, this number is very sensitive to the composition of the dataset and should not be generalized.

Unlike in the non-transfer case, there is no clear winner among the different classifiers. Thus, *without retraining, K-Means is comparable to a given supervised classifier.* However, with substantial retraining, the supervised classifiers improve more than the semi-supervised approach. Thus, overall, K-Means attains classification performance that is *comparable* to the other classifiers. Its advantage lies in the fact that it depends less on retraining, obtains comparable results more efficiently, and it is easy to explain its classification.

5.4 Comparing time

We compare the performance of the classifiers according to the time taken for benchmarking and training. The main steps involved in benchmarking are (i) reading matrices from `.mtx` files into memory, (ii) converting matrices from the default CSR format to other formats, (iii) iteratively performing the SpMV multiplication kernel to reduce the impact of noise and average the final result. On average, the first two steps contribute more to the time taken for benchmarking.

Table 8 shows the relative cost in format conversion, normalized to the cost of performing SpMV with the default CSR format (adapted from prior work [39]). The table shows the time taken to benchmark the matrices on each platform (Table 3), assuming an average time of 5 seconds for reading the `.mtx` files from disk. The estimates are a lower bound because it ignores the overheads of other function calls; profiling on each GPU platform takes close to two days.

Table 9 compares the different ML models according to the time taken to train. The results are intuitive; the training times for the non-DL models are reasonable and depend on the amount of training data. The CNN model comparatively takes much longer to train and presents a poor choice in the context of frequent retraining for model portability and to deal with new sparse matrices. The time taken for training in the local setting is similar to transfer with 0% additional data. Please note that the absolute numbers depend on the implementation, the amount of training data, and the training platform.

6 RELATED WORK

Given the importance of the SpMV computation, there has been extensive research to optimize the performance of SpMV kernels [21, 29]. In the following, we briefly discuss prior work on devising sparse matrix formats and techniques for format selection.

Devising sparse matrix formats. Prior work has studied the performance of SpMV on multicore CPUs [34] and GPUs [2] and defined some basic formats. Since then, many additional formats such as yaSpMV [36], generalized sliced ELL [15], CSR5 [22], CVR [35], CSR2 [4], and VBSF [20] have been proposed. Sparse matrix storage formats can be divided into two types: single format (for e.g., CSR, DIA, COO, CSR5) and mixed format (HYB, ELLPACK-RP). Such formats aim at reducing the data size by storing sub-blocks or by increasing the opportunities for vectorization.

Selecting sparse matrix formats. The multitude of formats naturally leads to the question of selecting the best format for a given matrix. Recent work has focused on using ML models [3, 11, 28, 30, 38]. Of particular interest is the question of overhead-conscious format selection which requires quantitative rather than qualitative

		0% Training Data					**25% Training Data**					**50% Training Data**				
	MLM	ACC	F1	MCC	GT	CSR	ACC	F1	MCC	GT	CSR	ACC	F1	MCC	GT	CSR
Turing to Volta	DT	78.98	0.78	0.50	0.96	1.03	80.90	0.80	0.52	0.96	1.03	81.01	0.8	0.52	0.96	1.03
	RF	**79.34**	**0.78**	**0.51**	**0.96**	**1.03**	82.32	0.82	0.55	**0.96**	**1.03**	81.92	0.81	0.55	0.96	1.03
	SVM	79.03	0.78	0.5	0.95	1.03	82.01	0.81	0.54	0.96	1.03	81.64	0.8	0.54	**0.96**	**1.03**
	KNN	77.56	0.77	0.46	0.95	1.02	81.44	0.81	0.53	0.96	1.03	**82.5**	**0.82**	**0.57**	0.96	1.03
	XGBoost	79.11	0.78	0.5	0.95	1.03	**82.33**	**0.82**	**0.55**	0.96	1.03	81.78	0.81	0.54	0.96	1.03
Pascal to Volta	DT	72.70	0.73	0.38	0.92	1	78.64	0.78	0.47	0.95	1.01	78.09	0.78	0.46	0.95	1.01
	RF	**74.91**	**0.75**	**0.42**	0.93	1	**79.08**	**0.79**	**0.48**	0.94	1.01	**79.44**	**0.79**	**0.49**	0.95	1.02
	SVM	74.68	0.75	0.41	**0.94**	**1.01**	78.15	0.78	0.46	**0.95**	**1.01**	78.93	0.78	0.48	**0.95**	**1.02**
	KNN	69.25	0.71	0.33	0.89	0.96	75.24	0.76	0.43	0.91	0.98	77.25	0.78	0.48	0.93	0.99
	XGBoost	71.03	0.72	0.37	0.9	0.97	76.3	0.77	0.44	0.92	0.99	76.98	0.77	0.45	0.93	1
Turing to Pascal	DT	77.67	0.75	0.49	0.95	1.04	80.76	0.79	0.55	0.96	1.04	80.63	0.79	0.55	0.96	1.05
	RF	78.47	0.76	0.51	0.95	1.05	81.29	0.79	0.56	0.96	1.05	80.51	0.78	0.55	0.96	1.05
	SVM	**79.47**	**0.77**	**0.54**	**0.96**	**1.05**	**81.88**	0.80	**0.57**	**0.97**	**1.06**	81.36	0.79	0.57	**0.96**	**1.06**
	KNN	77.23	0.75	0.48	0.95	1.04	81.48	**0.80**	0.57	0.96	1.05	**83.65**	**0.83**	**0.63**	0.96	1.06
	XGBoost	77.8	0.76	0.5	0.95	1.04	81.21	0.79	0.56	0.96	1.05	81.41	0.8	0.57	0.96	1.05
Pascal to Turing	DT	81.06	0.82	0.55	0.97	1.03	86.99	0.87	0.65	0.98	1.04	89.61	0.90	0.71	0.99	1.05
	RF	84.85	0.86	0.63	0.98	1.04	**88.94**	**0.89**	**0.70**	**0.96**	**1.05**	**91.50**	**0.92**	**0.76**	**0.99**	**1.05**
	SVM	**85.49**	**0.86**	**0.64**	**0.98**	**1.04**	88.04	0.88	0.68	0.98	1.04	90.02	0.90	0.73	0.99	1.05
	KNN	76.23	0.78	0.46	0.95	1.01	81.08	0.83	0.54	0.96	1.02	84.11	0.85	0.61	0.97	1.03
	XGBoost	77.47	0.79	0.49	0.96	1.02	83.58	0.85	0.60	0.97	1.03	86.83	0.88	0.66	0.98	1.04
Volta to Turing	DT	89.55	0.89	0.69	0.98	1.05	91.85	0.91	0.76	0.99	1.05	92.46	0.92	0.78	0.99	1.05
	RF	**90.77**	**0.90**	**0.73**	**0.99**	**1.05**	**92.70**	**0.92**	**0.78**	**0.99**	**1.05**	93.23	0.93	0.80	**0.99**	**1.05**
	SVM	89.69	0.89	0.70	0.98	1.04	90.87	0.90	0.72	0.99	1.05	90.98	0.90	0.73	0.99	1.05
	KNN	77.54	0.78	0.45	0.96	1.02	80.92	0.82	0.52	0.97	1.03	84.85	0.85	0.61	0.98	1.04
	XGBoost	90.18	0.90	0.72	0.99	1.05	91.87	0.91	0.76	0.99	1.05	**93.30**	**0.93**	**0.80**	0.99	1.05

Table 7: Comparison of the effectiveness in sparse matrix format selection of different automated techniques using transfer learning across different GPUs.

Format	**Conversion Cost**
COO	9
ELL	102
HYB	147
Platform	**Time (Hours)**
Pascal	27
Quadro	24
Volta	19

Table 8: Relative cost of format conversion (adapted from [39]) and the time (rounded to the nearest hour) for benchmarking.

	Transfer data		
	0%	25%	50%
DT	32	39	49
RF	75	94	115
SVM	59	76	91
KNN	60	77	89
XGBoost	26	31	36
CNN	$\sim$ 27,000	$\geq$ 30,000	$\geq$ 30,000
K-Means-VOTE	5	6	7
K-Means-LR	11	15	20
K-Means-RF	7	9	10

Table 9: Average training times (rounded to seconds) of the models in the local and the transfer setting.

predictions [39, 40]. Other approaches focus on auto-tuning of optimization parameters [7, 18, 28, 31, 32]. In some cases the SpMV format is one of the tunable parameters by applying some heuristics [13, 33]. A completely different method of optimizing SpMV performance is the use of probabilistic modeling [19].

SpMV performance can also be enhanced by increasing the reuse of the vector data in cache [17]. This requires reordering the matrix, which may be combined with optimizing the format. However, formats such as sliced ELL, which reorder the rows, may reduce cache reuse, thus causing a performance tradeoff [15].

7 CONCLUSION

Due to the importance of the SpMV kernel, combined with its high performance sensitivity to sparsity patterns and the architecture, the problem is an interesting candidate for the use of ML techniques. However, so far, only supervised methods have been used. In this work, we have extended the toolbox by presenting a semi-supervised approach based on clustering. Its main advantage is that it separates determining the similarity between matrices from the selection of the optimal format and exposes these aspects to the user. This has the advantage of providing explainable classifications and is well-suited for the transfer case. Furthermore, unlike the supervised methods, the semi-supervised approach would also be suitable for an *online* learning scenario where new matrices are added, and new clusters are formed continuously. However, this would require an incremental clustering algorithm, which is beyond the scope of this work.

We have performed a range of experiments to find the best implementation of the semi-supervised approach, which is characterized by the clustering and the classification algorithms and the chosen number of clusters k, both in the *local* and in the *transfer* scenario. We compared the classification performance of our method to state-of-the-art and show that our method attains comparable performance. In contrast to earlier studies, we have used MCC scores which better reflect performance in this highly unbalanced multiclass problem. We have found that Random Forest and XGBoost still have somewhat better classification performance, while the CNN approach does not provide a performance that would offset its high computational cost. For all models, we observed that the classification performance varies widely among the different GPU architectures. The reason for this is the large variance in the number of matrices in the COO and HYB class. This is a general problem that occurs when using the SuiteSparse dataset for this purpose, and it affects the newer GPU architectures more than older GPUs and CPUs. In the future, we will target improvements to the semi-supervised model to match the classification performance of all supervised models. We will build upon this work to create an online classification system that makes full use of the clustering-based approach by being able to learn from SpMV operations while they are being performed.

REFERENCES

[1] Nathan Bell and Michael Garland. 2008. *Efficient Sparse Matrix-Vector Multiplication on CUDA*. Technical Report NVR-2008-004. NVIDIA.

[2] Nathan Bell and Michael Garland. 2009. Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors. In *SC*. Article 18, 11 pages.

[3] A. Benatia, W. Ji, Y. Wang, and F. Shi. 2016. Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU. In *ICPP*. 496–505.

[4] Bian Bian, Jianqiang Huang, Runting Dong, Lingbin Liu, and Xiaoying Wang. 2020. CSR2: A New Format for SIMD-accelerated SpMV. In *CCGRID*. 350–359.

[5] Daniele Buono, Fabrizio Petrini, Fabio Checconi, Xing Liu, Xinyu Que, Chris Long, and Tai-Ching Tuan. 2016. Optimizing Sparse Matrix-Vector Multiplication for Large-Scale Data Analytics. In *ICS*. Article 37, 12 pages.

[6] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics* 21, 1 (2020), 1–13.

[7] Jee W. Choi, Amik Singh, and Richard W. Vuduc. 2010. Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs. In *PPoPP*. 115–126.

[8] Dorin Comaniciu and Peter Meer. 2002. Mean shift: A robust approach toward feature space analysis. *IEEE TPAMI* 24, 5 (2002), 603–619.

[9] Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM TOMS* 38, 1, Article 1 (Dec. 2011), 25 pages.

[10] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE CVPR*. 248–255.

[11] S. Filippone, V. Cardellini, D. Barbieri, and A. Fanfarillo. 2017. Sparse Matrix-Vector Multiplication on GPGPUs. *ACM TOMS* 43, 4, Article 30 (Jan. 2017), 49 pages.

[12] John L. Hennessy and David A. Patterson. 2019. A New Golden Age for Computer Architecture. *CACM* 62, 2 (Jan. 2019), 48–60.

[13] Eun-Jin Im, Katherine Yelick, and Richard Vuduc. 2004. Sparsity: Optimization Framework for Sparse Matrix Kernels. *The International Journal of High Performance Computing Applications* 18, 1 (2004), 135–158.

[14] Intel Software. 2021. Intel oneAPI Math Kernel Library. Online.

[15] Moritz Kreutzer, Georg Hager, Gerhard Wellein, Holger Fehske, and Alan R Bishop. 2014. A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units. *SIAM Journal on Scientific Computing* 36, 5 (2014), C401–C423.

[16] K Krishna and M Narasimha Murty. 1999. Genetic K-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29, 3 (1999), 433–439.

[17] J. Langguth, N. Wu, J. Chai, and X. Cai. 2015. Parallel Performance Modeling of Irregular Applications in Cell-Centered Finite Volume Methods over Unstructured Tetrahedral Meshes. *J. Parallel and Distrib. Comput.* 76, C (Feb. 2015), 120–131.

[18] Jiajia Li, Guangming Tan, Mingyu Chen, and Ninghui Sun. 2013. SMAT: An Input Adaptive Auto-tuner for Sparse Matrix-vector Multiplication. In *PLDI*. 117–126.

[19] K. Li, W. Yang, and K. Li. 2014. Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling. *IEEE TPDS* 26, 1 (2014), 196–205.

[20] Yishui Li, Peizhen Xie, Xinhai Chen, Jie Liu, Bo Yang, Shengguo Li, Chunye Gong, Xinbiao Gan, and Han Xu. 2019. VBSF: a new storage format for SIMD sparse matrix–vector multiplication on modern processors. *The Journal of Supercomputing* 76 (apr 2019), 2063–2081.

[21] Changxi Liu, Biwei Xie, Xin Liu, Wei Xue, Hailong Yang, and Xu Liu. 2018. Towards Efficient SpMV on Sunway Manycore Architectures. In *ICS*. 363–373.

[22] Weifeng Liu and Brian Vinter. 2015. CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication. In *ICS*. 339–350.

[23] Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405, 2 (1975), 442–451.

[24] I. Nisa, C. Siegel, A. S. Rajam, A. Vishnu, and P. Sadayappan. 2018. Effective Machine Learning Based Format Selection and Performance Modeling for SpMV on GPUs. In *International Workshop on Automatic Performance Tuning*. 1056–1065.

[25] NVIDIA Developer. 2021. CUSP. Online.

[26] NVIDIA Developer. 2021. cuSPARSE. Online.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *JMLR* 12 (2011), 2825–2830.

[28] J. C. Pichel and B. Pateiro-López. 2018. A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques. In *IEEE Cluster*. 46–54.

[29] Y. Saad. 1990. *SPARSKIT: a basic tool kit for sparse matrix computations*. Technical Report RIACS-TR-90-20. Research Institute for Advanced Computer Science.

[30] N. Sedaghati, T. Mu, L. Pouchet, S. Parthasarathy, and P. Sadayappan. 2015. Automatic Selection of Sparse Matrix Representation on GPUs. In *ICS*. 99–108.

[31] Guangming Tan, Junhong Liu, and Jiajia Li. 2018. Design and Implementation of Adaptive SpMV Library for Multicore and Many-Core Architecture. *ACM TOMS* 44, 4, Article 46 (Aug. 2018), 25 pages.

[32] Richard Vuduc, James W Demmel, and Katherine A Yelick. 2005. OSKI: A library of automatically tuned sparse matrix kernels. *Journal of Physics: Conference Series* 16, 1 (Jan. 2005), 521–530.

[33] R. Vuduc, J. W. Demmel, K. A. Yelick, S. Kamil, R. Nishtala, and B. Lee. 2002. Performance Optimizations and Bounds for Sparse Matrix-Vector Multiply. In *ICS*. IEEE Computer Society Press, 1–35.

[34] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. 2009. Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms. *Parallel Comput.* 35, 178–194.

[35] B. Xie, J. Zhan, X. Liu, W. Gao, Z. Jia, X. He, and L. Zhang. 2018. CVR: Efficient Vectorization of SpMV on x86 Processors. In *CGO*. 149–162.

[36] Shengen Yan, Chao Li, Yunquan Zhang, and Huiyang Zhou. 2014. yaSpMV: Yet Another SpMV Framework on GPUs. In *PPoPP*. 107–118.

[37] T. Zhang, R. Ramakrishnan, and M. Livny. 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM Sigmod Record* 25, 2 (1996), 103–114.

[38] Yue Zhao, Jiajia Li, Chunhua Liao, and Xipeng Shen. 2018. Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection. In *PPoPP*. 94–108.

[39] Yue Zhao, Weijie Zhou, Xipeng Shen, and Graham Yiu. 2018. Overhead-Conscious Format Selection for SpMV-Based Applications. In *IPDPS*. 950–959.

[40] W. Zhou, Y. Zhao, X. Shen, and W. Chen. 2020. Enabling Runtime SpMV Format Selection through an Overhead Conscious Method. *IEEE TPDS* 31, 1 (Jan. 2020), 80–93.