ML-based prediction model

Deliverable No: D1.3
Deliverable Title: ML-based prediction model
Deliverable Publish Date: 30 September 2022

Project Title: SparCity: An Optimization and Co-design Framework for Sparse Computation
Call ID: H2020-JTI-EuroHPC-2019-1
Project No: 956213
Project Duration: 36 months
Project Start Date: 1 April 2021
Contact: sparcity-project-group@ku.edu.tr

List of partners:

Participant no.	Participant organisation name	Short name	Country
1 (Coordinator)	Koç University	KU	Turkey
2	Sabancı University	SU	Turkey
3	Simula Research Laboratory AS	Simula	Norway
4	Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento em Lisboa	INESC-ID	Portugal
5	Ludwig-Maximilians-Universität München	LMU	Germany
6	Graphcore AS	Graphcore	Norway

CONTENTS

1 INTRODUCTION

The SparCity project is funded by EuroHPC JU (the European High Performance Computing Joint Undertaking) under the 2019 call of Extreme Scale Computing and Data Driven Technologies for research and innovation actions. SparCity aims to create a supercomputing framework that will provide efficient algorithms and coherent tools specifically designed for maximizing the performance and energy efficiency of sparse computations on emerging High Performance Computing (HPC) systems, while also opening up new usage areas for sparse computations in data analytics and deep learning.

Sparse computations are commonly found at the heart of many important applications, but at the same time it is challenging to achieve high performance when performing the sparse computations. SparCity delivers a coherent collection of innovative algorithms and tools for enabling both high efficiency of sparse computations on emerging hardware platforms. More specifically, the objectives of the project are:

- to develop a comprehensive application and data characterization mechanism for sparse computation based on the state-of-the-art analytical and machine-learning-based performance and energy models,
- to develop advanced node-level static and dynamic code optimizations designed for massive and heterogeneous parallel architectures with complex memory hierarchy for sparse computation,
- to devise topology-aware partitioning algorithms and communication optimizations to boost the efficiency of system-level parallelism,
- to create digital SuperTwins of supercomputers to evaluate and simulate what-if hardware scenarios,
- to demonstrate the effectiveness and usability of the SparCity framework by enhancing the computing scale and energy efficiency of challenging real-life applications.
- to deliver a robust, well-supported and documented SparCity framework into the hands of computational scientists, data analysts, and deep learning end-users from industry and academia.

1.1 OBJECTIVES OF THIS DELIVERABLE

With this deliverable, we present our progress regarding Task 1.2 of the SparCity project. The main objective of this task is to provide the foundation for a recommendation system for training ML classifiers for storage format, execution platform, and algorithm selection for a wide range of sparse matrix problems that use sparse matrices, sparse tensors and graphs. It also aims at training ML systems for any tuning parameter or characteristic, including power and energy in heterogeneous systems. The trained model is used in Task 2.2 for parameter selection and performance evaluation for node-level optimizations.

1.2 WORK PERFORMED

We have carried out studies on ML-based prediction models for three different research areas: (i) sparse matrices, (ii) sparse tensors, and (iii) graphs.

In the context of sparse matrix operations, we explore semi-supervised learning models to solve the portability problem of format selection for SpMV. The idea is to cluster matrices so that

the matrices in each group have the same optimal format irrespective of the architecture. The work regarding this category is mostly carried out by the Simula partner and the outcomes are reported in Section 2.

For sparse tensor operations, we have extracted a detailed and exclusive set of sparse tensor features to train the ML-based prediction models. Our aim is to predict the optimal reordering model for tensor decomposition (CPD). The work is mainly carried by the KU partner and the progress is reported in Section 3.

In terms of graph operations, we propose effective graph algorithms optimized for GPUs. We use a highly configurable kernel instead of a one-size-fits-all kernel for every graph and architecture. We configure its resource allocation via employing a machine learning model which accounts for each input graph's structure, as well as the architectural details of the GPU. The work is carried by the SU partner and presented in Section 4. Other consortium members were frequently informed about the work in progress and they have provided feedback to the relevant partner.

1.3 DEVIATIONS AND COUNTER MEASURES

There was no deviation from the work plan.

1.4 RESOURCES

The duration of this task in the work plan has not yet ended and some of the research results included in this deliverable are still work-in-progress, thus not yet available as publicly accessible software repositories. Efforts are planned in the near future to publish these as either journal or conference proceedings and open-access repositories.

The preliminary repositories of software and data (currently only open for the project participants) will be made open and organized more coherently in the future under the SparCity project public repository (https://github.com/sparcityeu/).

2 ML BASED PREDICTION MODELS FOR SPARSE MATRICES

Sparse matrices for many applications can be very large which makes it impossible to store them in memory using a dense format. Sparse matrix storage formats save space by not storing zero elements. A number of sparse storage formats have been proposed over the years,[1] and several high-performance libraries like Intel MKL,[2] and CUSP[3] and cuSPARSE[4] from NVIDIA provide support for the more popular storage formats. In the following, we briefly review these formats.

2.1 ML BASED PREDICTION MODELS FOR SPARSE MATRIX FORMAT SELECTION

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 $\mathcal{O}(n)$ 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. ELL uses an index matrix to store the corresponding column index for every nonzero element. ELL thus eschews the need for a compressed row array. Furthermore, since the number of operations per row is known beforehand, the computation for each row can be unrolled completely and optimized for SIMD processing. The ELL format is designed for vector architectures. 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 ELL. Other formats, like *diagonal* (DIA), take advantage of specific sparsity patterns but can also take $\mathcal{O}(n^2)$ space in the worst case. More recent proposals aim to exploit microarchitectural features like vectorization and the compute capabilities of GPUs.[5]

[1] Y. Saad. *SPARSKIT: a basic tool kit for sparse matrix computations*. Tech. rep. RIACS-TR-90-20. Research Institute for Advanced Computer Science, 1990; Nathan Bell and Michael Garland. *Efficient Sparse Matrix-Vector Multiplication on CUDA*. tech. rep. NVR-2008-004. NVIDIA, 2008; Bian Bian et al. "CSR2: A New Format for SIMD-accelerated SpMV". 2020, pp. 350–359; Weifeng Liu and Brian Vinter. "CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication". 2015, pp. 339–350; Yishui Li et al. "VBSF: a new storage format for SIMD sparse matrix–vector multiplication on modern processors". *The Journal of Supercomputing* 76 (2019), pp. 2063–2081; B. Xie et al. "CVR: Efficient Vectorization of SpMV on x86 Processors". 2018, pp. 149–162; Nathan Bell and Michael Garland. "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors". 2009, 18:1–18:11; Jee W. Choi, Amik Singh, and Richard W. Vuduc. "Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs". 2010, pp. 115–126.

[2] Intel Software. *Intel oneAPI Math Kernel Library*. Online. 2021.

[3] NVIDIA Developer. *CUSP*. Online. 2021.

[4] NVIDIA Developer. *cuSPARSE*. Online. 2021.

[5] Liu and Vinter, "CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication"; Xie et al., "CVR: Efficient Vectorization of SpMV on x86 Processors"; Li et al., "VBSF: a new storage format for SIMD sparse matrix–vector multiplication on modern processors"; Bian et al., "CSR2: A New Format for SIMD-accelerated SpMV".

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*.[6] 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[7] 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.[8]

SpMV format prediction consists of two stages: training an offline model and inference of the predicted optimal format. Training a format predictor involves profiling an input set of sparse matrices across different sparse formats for a target platform.

Automated sparse storage format selection requires identifying important statistical features that help differentiate classes of sparse matrices.[9] 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. The basic idea is simple. 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,[10] Support Vector Machines,[11] and XGBoost models for automated format selection. Convolutional Neural Networks (CNNs) are popular deep learning models used for image classification.[12] Recent work employs CNN-based deep learning (DL) models for format selection by encoding the sparse matrix as an image and

[6] N. Sedaghati et al. "Automatic Selection of Sparse Matrix Representation on GPUs". 2015, pp. 99–108.

[7] Timothy Davis and Yifan Hu. "The University of Florida Sparse Matrix Collection". *ACM Transactions on Mathematical Software (TOMS)* 38.1 (2011), pp. 1–25. ISSN: 1557-7295. DOI: 10.1145/2049662.2049663.

[8] Sedaghati et al., "Automatic Selection of Sparse Matrix Representation on GPUs"; Yue Zhao et al. "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection". 2018, pp. 94–108; Yue Zhao et al. "Overhead-Conscious Format Selection for SpMV-Based Applications". 2018, pp. 950–959; Jiajia Li et al. "SMAT: An Input Adaptive Auto-tuner for Sparse Matrix-vector Multiplication". 2013, pp. 117–126; Guangming Tan, Junhong Liu, and Jiajia Li. "Design and Implementation of Adaptive SpMV Library for Multicore and Many-Core Architecture". 44.4 (2018), 46:1–46:25. ISSN: 0098-3500; J. C. Pichel and B. Pateiro-López. "A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques". 2018, pp. 46–54; K. Li, W. Yang, and K. Li. "Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling". 26.1 (2014), pp. 196–205; W. Zhou et al. "Enabling Runtime SpMV Format Selection through an Overhead Conscious Method". 31.1 (2020), pp. 80–93; I. Nisa et al. "Effective Machine Learning Based Format Selection and Performance Modeling for SpMV on GPUs". *International Workshop on Automatic Performance Tuning*. 2018, pp. 1056–1065; A. Benatia et al. "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU". 2016, pp. 496–505.

[9] Benatia et al., "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU".

[10] Sedaghati et al., "Automatic Selection of Sparse Matrix Representation on GPUs".

[11] Benatia et al., "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU".

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

showcase good predictive accuracy.[13]

This work employs statistical and dynamic features described in previous reports (**SparCity Project, Deliverable 1.1, Core Set of Sparse Computation Features**). Depicted statistical features are commonly used by existing non-DL supervised-learning-based approaches, but the supervised models may not use all the common statistical features to avoid overfitting and reduce training noise. Note that feature identification in SpMV mostly ignores domain-specific knowledge, since that would restrict the approach's applicability.

2.3 APPROACH

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 modern-day 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 different 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, on every single architecture, 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.[14] 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 compared to xgboost, Random forest or MLP. 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.

As described, 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. However, 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 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.

[13] Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection"; Pichel and Pateiro-López, "A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques"; Zhen Xie et al. "IA-SpGEMM: An Input-Aware Auto-Tuning Framework for Parallel Sparse Matrix-Matrix Multiplication". Association for Computing Machinery, 2019, pp. 94–105.

[14] Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection".

	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 1 *Performance of ML models on different GPUs.*

2.4 ENABLING PORTING OF SPMV MODEL RESULTS

Transfer learning of CNN models gains much attention in the area of SpMV optimization. Training a model by using pre-trained weights from previously learned models can improve training time drastically. Furthermore, in SpMV when transferring a model towards different architectures, a fraction of data can achieve a similar performance when we train a newer model with a full dataset from scratch. This will decrease training/retraining efforts drastically. Transfer models further can be used on different architectures for incremental learning of incorrect examples or cases where formats gets changed for particular matrices.

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 1). However, when the same classifier is used on the NVIDIA Volta V100 platform, accuracy drops to 71.03% (Table 2). 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, 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.

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

Table 2 *Comparison of the effectiveness in sparse matrix format selection of different automated techniques using transfer learning across different GPUs.*

In contrast to this, Quadro has a large number of SMs, and can support a larger number of active threads. The results of the clustering can also be used to improve supervised methods when they are being trained on a small number of samples. Normally, training data is selected randomly from a larger training set. Thus, a small training set might not be very representative of the entire training set, especially if the test set contains many similar matrices. By selecting matrices from each cluster, we ensure that the entire training set is well represented. Furthermore, using the set of features described in Table 1, the accuracy of our supervised learning models has been improved compared with earlier work using more basic features.

For a pre-trained model to be accurate on a new architecture, it needs to *learn* the features of the new architecture and predict formats accordingly. For example, the NVIDIA K80 GPU has relatively few symmetric multiprocessors (SMs) compared to Quadro. The implementation of the ELL format in NVIDIA CUSP uses a single thread for each row in the sparse matrix, and due to

the limited number of active threads available on a small GPU such as K80, is ill-suited for sparse matrices on the K80 platform and thus ELLACK is almost never suitable for sparse matrices on K80.

Parallel to this, Quadro, which has a large number of SMs is well-suited for CSR even on moderately large matrices which would work better as ELL on a Turing GPU with a lesser number of SMs, because it has enough active threads that a single vector in the vector CSR vector kernel has to process a very small number of rows.

2.5 SEMI-SUPERVISED LEARNING

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. In practice, it is preferable to benchmark a small group of matrices per cluster, not only one. The clusters are formed using the matrices from the training set. A matrix will be assigned to some cluster based on its statistical features. 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. Because the clustering algorithm only requires statistical features, and not labels, it requires less work to create the clusters than to train a supervised model. A purely unsupervised model, of course, cannot be used for format prediction since we have to provide labels for the clusters. 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

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

where $\text{count}(c, f)$ 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 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[15] with the features does not work well.

[15] S. P. Lloyd. "Least Squares Quantization in PCM". 28.2 (1982), pp. 129–136.

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 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. It should also be possible to extend clustering to *guide* the selection of a training set for supervised models, instead of random selection, for even better accuracy.

While the dataset which we used for training only contains matrices which are sufficiently large, using 80 clusters gives a high accuracy predictor. The same approach can also be used to provide a significant increase in the accuracy of other feature-based classifiers.

For example, using a K Nearest Neighbors based classifier without this pipeline gives an accuracy of 61%, one which only uses min-max scaling and PCA reduction gives an accuracy of 85%, while one which uses log and square root transforms for features which depend on the matrix size gives an accuracy of 90% and a slowdown of merely 1.02X compared to an *ideal* (i.e., ground truth) classifier.

When we set the number of clusters to be 80, we found the average purity of the clusters to be 86% for a set of 1000 datapoints. Thus, in practice, it is preferable to benchmark a small group of matrices per cluster, not only one. 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%

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 3 *Different NVIDIA GPUs used in our experiments.*

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.

Thus even if the subset chosen in a particular cluster results in a format which is not actually the majority format, the detrimental effect is confined to that cluster. Furthermore, with an 87% chance to hit the majority format for each matrix that is benchmarked, increasing number

Note that only the a *subset* of matrices in each cluster need to be profiled on SpMV kernels. In case of the transfer learning methods proposed till date the number of matrices that must be profiled depends upon the transfer learning percentage.

In order to compare the proposed method to the existing models, we trained the supervised models from scratch on a new architecture using a number of matrices in the training set equal to the number of matrices benchmarked for the semi-supervised method.

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**[17] and **Birch** clustering.[18] 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.

2.6 EVALUATION

2.6.1 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 3 shows technical details of the three GPUs. We will refer to the GPUs by their architecture names.

[16] K Krishna and M Narasimha Murty. "Genetic K-means algorithm". *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29.3 (1999), pp. 433–439.

[17] Dorin Comaniciu and Peter Meer. "Mean shift: A robust approach toward feature space analysis". 24.5 (2002), pp. 603–619.

[18] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". *ACM Sigmod Record* 25.2 (1996), pp. 103–114.

	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 4 *Distribution of the best sparse formats across GPUs.*

	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 5 *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.*

Implementation. We have reimplemented several existing automated sparse matrix format selection approaches that use supervised learning.[19] We use the scikit-learn library[20] 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. 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 SVM, we use log transforms on the features. For XGBoost, we set a learning rate of 0.1 and the number of rounds to 100. For KMeans, we set the number of clusters to the optimal values selected in a series of initial experiments. We have reimplemented the publicly available convolutional neural network model (CNN) from prior work[21] using the TensorFlow v2 framework.[22] We use the NVIDIA CUDA toolkit v9.2 and the NVIDIA CUSP[23] libraries for obtaining the SpMV benchmark results.

[19] Sedaghati et al., "Automatic Selection of Sparse Matrix Representation on GPUs"; Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection"; Benatia et al., "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU".

[20] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[21] Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection".

[22] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[23] NVIDIA Developer, *CUSP*.

Benchmarks. We test all models using sparse matrices from the SuiteSparse Matrix Collection.[24] 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[25]). 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.[26] 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.[27] Prior work[28] 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 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 4 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 4). 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 4, the classes obtained are highly unbalanced, with the majority of matrices having CSR as the best format.

2.6.2 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 5 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.[29] MCC is widely regarded as a superior metric, especially for multiclass problems.[30] 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.

[24] Davis and Hu, "The University of Florida Sparse Matrix Collection".

[25] Benatia et al., "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU".

[26] Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection"; Pichel and Pateiro-López, "A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques".

[27] NVIDIA Developer, *CUSP*.

[28] **sedaghati-ics15** ; Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection".

[29] Brian W Matthews. "Comparison of the predicted and observed secondary structure of T4 phage lysozyme". *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405.2 (1975), pp. 442–451.

[30] Davide Chicco and Giuseppe Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". *BMC genomics* 21.1 (2020), pp. 1–13.

	Algorithm	NC	0% Training Data MCC	ACC	F1	25% Training Data MCC	ACC	F1	50% Training Data MCC	ACC	F1
P to T	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
P to V	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
T to P	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
T to V	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
V to P	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
V to T	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

Table 6 *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. Abbreviations are used to refer the architectures: Pascal (P), Volta (V) and Turing (T).*

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 6 shows the results of this portability experiment. The worse-performing classifiers are omitted from the table to improve presentation and can be found in the detailed results paper.[31] 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 4, 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.

2.6.3 COMPARISON TO SUPERVISED CLASSIFIERS

In the next experiment, we study the performance of all supervised classifiers in the *local* setting (i.e., training and inference on the same platform). Table 1 summarizes the results. Columns 2–4 show the *accuracy* (ACC), *F1 score* (F1), and MCC metrics. We also use *Matthew's correlation coefficient* (MCC) since the classes are highly unbalanced. 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 $\geqslant$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[32] has good results for the Turing platform, but its performance drops significantly for the Pascal and Volta architectures , which we believe is due to the composition of the training dataset. Our result of the CNN model is poorer compared to prior work,[33] which we believe is due to the composition and size of the training dataset. The CNN model used fewer matrices due to a bug in downsampling, which may be a reason behind the weak results. Given that, we do not expect the results to improve drastically with a larger dataset too. 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

[31] Sunidhi Dhandhania et al. "Explaining the Performance of Supervised and Semi-Supervised Methods for Automated Sparse Matrix Format Selection". *50th International Conference on Parallel Processing Workshop*. 2021, pp. 1–10.

[32] Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection".

[33] Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection"; Pichel and Pateiro-López, "A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques".

GPU, which is close to prior work,[34] 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. As a consequence, we excluded the CNN from the following experiments. The semi-supervised method lies between the weakest and the best supervised methods. Thus, the cost of going to the semi-supervised method is not very high and the semi-supervised approach is quite competitive with supervised models.

Table 2 shows the results for the *transfer* case for the supervised classifiers (see paper[35] for the details). 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 4). 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 ~ 15 hours to complete. Following the structure of Table 6, 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 1. 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. In the Pascal to Volta transfer, KNN, XGBoost, an accuracy of about 75% and MCC scores of 0.43 even though the CSR value is below one, which means that the quantitative performance (i.e., running time) is worse than what we would obtain if we always used CSR. 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.

2.6.4 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 7 shows the relative cost in format conversion, normalized to the cost of performing SpMV with the default CSR format (adapted from prior work[37]). The table shows the time taken

[34] Zhao et al., "Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection".

[35] Dhandhania et al., "Explaining the Performance of Supervised and Semi-Supervised Methods for Automated Sparse Matrix Format Selection".

[37] Zhao et al., "Overhead-Conscious Format Selection for SpMV-Based Applications".

Format	Conversion Cost
COO	9
ELL	102
HYB	147

Platform	Time (Hours)
Pascal	27
Quadro	24
Volta	19

Table 7 *Relative cost of format conversion (adapted from[36]) and the time (rounded to the nearest hour) for benchmarking.*

to benchmark the matrices on each platform (Table 4), 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.

	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	$\geqslant$ 30,000	$\geqslant$ 30,000
K-Means-VOTE	5	6	7
K-Means-LR	11	15	20
K-Means-RF	7	9	10

Table 8 *Average training times (rounded to seconds) of the models in the local and the transfer setting.*

Table 8 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.

3 ML BASED PREDICTION MODELS FOR SPARSE TENSORS

We propose to utilize the sparse tensor features extracted in Task 1.1 of the project for predicting the optimal tool and reordering model using machine learning.

3.1 RELATED WORK

To the best of our knowledge, there is only one study (and its extension) that uses ML for sparse tensor computations in the literature: Sun et al.[38] proposed a framework, namely SpTFS, that automatically predicts the optimal storage format for cananonical polyadic decomposition (CPD). SpTFS lowers the sparse tensor to fixed-sized matrices and gives them to convolutional neural network (CNN) as inputs along with tensor features. The authors improve SpTFS by adopting

[38]Qingxiao Sun et al. "Sptfs: Sparse tensor format selection for mttkrp via deep learning". *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2020, pp. 1–14.

both supervised and unsupervised learning based machine learning models in a recent work.[39]

On the other hand, the utilization of ML models for sparse matrix computations is well-studied in the literature. Since tensors are generalizations of matrices to higher dimensions, and due to the similarity of the sparse matrix and tensor computations, there is a huge potential to explore and extend these findings for sparse tensors.

Chen et. al.[40] proposed a machine learning based model that predicts the best storage format and parameters using input matrix features for the ARMv8-based many-core architectures. They have considered random forests based model (RF), Gaussian naïve bayes (GNB), multilayer perception (MLP), soft voting/majority rule Classification (VC), k-Nearest Neighbor (KNC, k = 1), logistic regression (LR), decision tree classification (DCT). At the end, RF model is chosen because its accuracy is comparable to alternative techniques. Their motivation for using a decision-tree-based model (i.e., random forests) is that it is interpretable and gives insights of why a certain storage format is chosen.

Xie et. al.[41] proposed a framework for automatically determining the best format and algorithm for sparse matrix-matrix multiplication (SpGEMM). They have applied a scaling method to map the sparse matrix to the density representation with size of 128×128. The ML model is generated by using the collected performance data, extracted features, and density representations.

Deep learning models, most specifically convolutional neural networks (CNN) are popularly used for sparse matrix operations. CNN is first used by Zhao et al.[42] for implementing sparse matrix format selection through histogram representation. Berrada et. al.[43] leverage CNNs in order to provide an effective estimation of the performance of the SpMV. They propose a blockwise realization to make the CNN model architecture independent of the sizes of the matrix in order to increase the amount of training/validation data. Guo el.al.[44] select the proper sparse matrices for CSR-based SpMV on GPUs by converting it to a matrix classification problem, then addressing this classification problem by using the CNNs. Mohammed et. al. developed DIESEL,[45] a deep learning-based tool that predicts and executes the best performing SpMV kernel using a feature set consisting of 14 carefully selected features.

CNNs are efficient at identifying patterns in data classification problems and are mainly used in pattern recognition tasks with the purpose of mimicking the functionality of the receptive fields in the human visual cortex. Despite the these attractive properties, utilizing CNN for algorithm prediction of sparse computations brings some challenges. One is that sparse tensors have very high sparsity and different sizes while the CNN generally requires fixed size input data. To alleviate this challenge, previous efforts consider converting the data to fixed-sized small matrices. One drawback of this approach is that most of the important information is lost.

[39] Qingxiao Sun et al. "Input-aware Sparse Tensor Storage Format Selection for Optimizing MTTKRP". *IEEE Transactions on Computers* (2021).

[40] Donglin Chen et al. "Optimizing sparse matrix–vector multiplications on an armv8-based many-core architecture". *International Journal of Parallel Programming* 47.3 (2019), pp. 418–432.

[41] Zhen Xie et al. "IA-SpGEMM: An input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication". *Proceedings of the ACM International Conference on Supercomputing*. 2019, pp. 94–105.

[42] Yue Zhao et al. "Bridging the gap between deep learning and sparse matrix format selection". *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*. 2018, pp. 94–108.

[43] Maria Barreda et al. "Performance modeling of the sparse matrix–vector product via convolutional neural networks". *The Journal of Supercomputing* 76.11 (2020), pp. 8883–8900.

[44] Ping Guo and Changjiang Zhang. "Sparse Matrix Selection for CSR-Based SpMV Using Deep Learning". *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. IEEE. 2019, pp. 2097–2101.

[45] Thaha Mohammed et al. "DIESEL: A novel deep learning-based tool for SpMV computations and solving sparse linear equation systems". *The Journal of Supercomputing* 77.6 (2021), pp. 6313–6355.

3.2 REORDERING ALGORITHMS FOR CPD

Cananonical polyadic decomposition (CPD) is one of the most common tensor decomposition methods that is used to extract the latent features of multi-dimensional data. There are various implementations for CPD and each has different reordering algorithms proposed specifically for that scheme. SPLATT and ParTI are two most commonly used libraries for CPD.

SPLATT provides a shared-memory implementation for CPD. It adopts a medium-grain partitioning for sparse tensors for parallel execution of CPD. There are two reordering models proposed for SPLATT in the original paper,[46] namely bipartite graph model as mode independent ordering and hypergraph model for mode dependent ordering. Additionally, Acer et. al.[47] proposed a hypergraph partitioning-based reordering model utilizing fixed vertices and multiple weights proposed for the medium-grain partitioning.

ParTI[48] is a parallel tensor infrastructure which supports essential sparse tensor operations and tensor decompositions on multicore CPU and GPU architectures. Two reordering models were proposed for and integrated in ParTI, namely BFS-MCS and Lexi-Order reorderings.[49] BFS-MCS is a Breadth First Search (BFS)-like heuristic approach based on the maximum cardinality search family. It construct a hypergraph for a sparse tensor, where vertices represent tensor indices in all modes and hyperedges represent the nonzero entries. Lexi-Order is an extension of doubly lexical ordering of matrices to tensors. A lexicographic ordering of a vector is the standard dictionary ordering of the elements.

3.3 THE PROPOSED MODEL

Due to the vast diversity of the structure of the real-world sparse tensors, different reordering algorithms deliver their best performance on different tensors, and no single algorithm dominates on all tensors in terms of the CPD performance. Therefore, we propose to utilize sparse tensor features to predict the optimum reordering and execution algorithm via machine learning models.

We provide the features across all modes to the machine learning model as inputs. The previous work[50] has considered the features for only one mode, which may cause lack of some critical intuition from other dimensions. Therefore we propose to utilize significantly more features with critical roles to give information across different modes of the tensor. Our intuition is that carefully selected tensor features are more meaningful than turning a tensor into a fixed-size small matrix. It is because the real-world tensor sizes diverge significantly (sizes in some modes reach to millions while some are less than 50). On the other hand, an excessive number of features would have the risk of increasing the computational time for predictions. Therefore we opt to use an increased but wisely-selected feature set to train the ML-based models.

In addition to the tensor features utilized in the related work, we also include the features of standard deviation and coefficient of variation of nonzeros per slice or fiber, fibers per slice. In a relevant work,[51] it is shown that high standard deviation of fiber length causes inter-warp load imbalance and low GPU occupancy; whereas high standard deviation of the slice volume

[46] Shaden Smith et al. "SPLATT: Efficient and parallel sparse tensor-matrix multiplication". *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2015, pp. 61–70.

[47] Seher Acer, Tugba Torun, and Cevdet Aykanat. "Improving medium-grain partitioning for scalable sparse tensor decomposition". *IEEE Transactions on Parallel and Distributed Systems* 29.12 (2018), pp. 2814–2825.

[48] Jiajia Li, Yuchen Ma, and Richard Vuduc. *ParTI!: A parallel tensor infrastructure for multicore CPUs and GPUs*. 2018.

[49] Jiajia Li et al. "Efficient and effective sparse tensor reordering". *Proceedings of the ACM International Conference on Supercomputing*. 2019, pp. 227–237.

[50] Sun et al., "Sptfs: Sparse tensor format selection for mttkrp via deep learning".

[51] Israt Nisa et al. "Load-balanced sparse MTTKRP on GPUs". *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2019, pp. 123–133.

is related to significant inter-thread-block load imbalance. Coefficient of variation is another important metric for analysis, which is defined as the ratio of standard deviation to the mean. It yields an intuition regarding the distribution of nonzeros among different slices or fibers and it is useful as it allows to compare between data sets with widely different means. It is also used to evaluate the dispersion of the number of non-zero elements per row for sparse matrix computations in the literature.[52]

Our model also takes the running time results of the tensor-reordering-tool triples for each case to train the model. We will use 5-fold cross validation over the tensor data. That is, the dataset is divided into five equal-sized partitions such that one partition is retained for testing and the remaining four are used for training.

The output of the model is a matching between sparse tensors and the reordering methods. We consider 5 different reordering and algorithm cases as an output. These are 2 different reordering models for SPLATT[53] along with one reordering model for medium-grain partitioning,[54] and 2 different reordering models proposed for ParTI.[55] The proposed model is extensible in the sense that it is open to the addition of new algorithms and training data.

3.4 DATASET

Choosing the best reordering algorithm for CPD is a complex task which requires a large amount of data for training. Although there are thousands of publicly-available data sources for sparse matrices, only 20-30 of sparse tensor data are available to study on. This might be due to that the tensor data generally stem from some commercial company and so are not distributed.

We have gathered 16 large real-world sparse tensors from the FROSTT[56] and HaTen2[57] datasets. Moreover, we add 12 smaller tensors from ITensor,[58] NWChem,[59] and GenTen.[60] Yet this dataset consisting of 28 sparse tensors is not enough for training and testing ML models.

Therefore we consider extending the dataset by randomly generating sparse tensor data. However, simply generating random multi-dimensional data is not adequate since this way we mostly obtain tensors with fibers containing only 1 nonzero. On the other hand, we have observed that real-world sparse tensors often have fibers with diverse amount of nonzeros such that the most loaded fibers have thousands of nonzeros. Thus we propose to generate random sparse tensors having the number of nonzeros per fiber is also random, ranging from 1 to a number relative to the size of the respective mode. We also consider generating more sparse tensors via utilizing sparse matrices in Suite Sparse[61] collection, as done in the related work.[62]

---

[52] Walid Abu-Sufah and Asma Abdel Karim. "Auto-tuning of sparse matrix-vector multiplication on graphics processors". *International Supercomputing Conference*. Springer. 2013, pp. 151–164.

[53] Smith et al., "SPLATT: Efficient and parallel sparse tensor-matrix multiplication".

[54] Acer, Torun, and Aykanat, "Improving medium-grain partitioning for scalable sparse tensor decomposition".

[55] Li et al., "Efficient and effective sparse tensor reordering".

[56] Shaden Smith et al. *FROSTT: The formidable repository of open sparse tensors and tools*. 2017.

[57] Inah Jeon et al. "Haten2: Billion-scale tensor decompositions". *2015 IEEE 31st international conference on data engineering*. IEEE. 2015, pp. 1047–1058.

[58] Matthew Fishman, Steven White, and Edwin Stoudenmire. "The ITensor software library for tensor network calculations". *SciPost Physics Codebases* (2022), p. 004.

[59] Marat Valiev et al. "NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations". *Computer Physics Communications* 181.9 (2010), pp. 1477–1489.

[60] Eric T Phipps and Tamara G Kolda. "Software for sparse tensor decomposition on emerging computing architectures". *SIAM Journal on Scientific Computing* 41.3 (2019), pp. C269–C290.

[61] Davis and Hu, "The University of Florida Sparse Matrix Collection".

[62] Sun et al., "Sptfs: Sparse tensor format selection for mttkrp via deep learning".

4 ML BASED PREDICTION MODELS FOR GRAPHS

Graphs are a commonly used modeling tool for many real world data sets spanning a variety of domains, among which are bioinformatics, computational chemistry, and information systems. Entities and their relationship in a data set correspond to nodes and edges in a graph, respectively. A data set represented as a graph can be analyzed through graph-based mathematical formulations. This analysis can take place at different granularities, from entity-wise and relationship-wise properties to sub-graph and whole-graph properties.

Generally, a graph is expressed as an adjacency matrix and is stored in one of the many sparse matrix storage formats. It is important to note that the relations in the vast majority of real-world data sets are highly sparse, and consequently, so are the graphs that represent them. For this reason, calculations over graphs are characterized by highly irregular memory access patterns and uneven workload distributions per unit of output. Consequently, developing efficient parallel graph algorithms is quite challenging. Not only do graph kernels have to deal with varying sparsity patterns across different graphs and even within the same graph, but they also need to account for architectural differences among HPC environments.

In this work, we propose a simple yet effective approach to optimize graph algorithms on GPUs; rather than build a one-size-fits-all kernel for every graph and every architecture, we use a highly configurable kernel, and employ a machine learning model to configure said kernel's resource allocation in a manner that accounts for each input graph's structure, as well as the architectural details of the GPU. We show the effectiveness of this approach on calculating the node-node similarity measure known as the Jaccard Weight.

4.1 JACCARD WEIGHTS

Given an unweighted graph $G = (V, E)$ where V and E are the vertices and edges of the graph, respectively, the Jaccard Weight of an edge $(u, v) \in E$ is defined as the ratio between the intersection and union cardinalities of u and v's (outgoing) neighborhoods, defined as $\Gamma(u) = \{x : (u, x) \in E\}$ and $\Gamma(v) = \{x : (v, x) \in E\}$, respectively. More formally, the Jaccard Weight of the edge (u, v) is

$$w^{\mathcal{J}}((u, v)) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}$$

which can be reduced to

$$w^{\mathcal{J}}_{(u,v)} = \frac{|\Gamma(u)| + |\Gamma(v)| - I(u, v)}{I(u, v)}$$

where $I(u, v) = |\Gamma(u) \cap \Gamma(v)|$.

Finding the cardinality of the neighborhood of each vertex (i.e. vertex degrees) is an easy task, hence, the bottleneck of the Jaccard Weight computation problem is counting the intersections between vertices' neighborhoods. To find the the intersection count at an edge $(u, v) \in E$, we linearly iterate through every neighbor of u, and check whether that neighbor is also a neighbor of v through another linear iteration through the neighborhood of v. Given that the number of edges in a graph is m and the maximum degree in the graph is Δ, the worst-case complexity of finding the Jaccard Weights of a graph is $\mathcal{O}(m\Delta^2)$. However, this complexity can be improved by replacing the linear search over the neighborhood of v with a binary search operation.[63] This reduces the complexity to $\mathcal{O}(m\Delta \log(\Delta))$.

[63] Alexandre Fender et al. "Parallel Jaccard and related graph clustering techniques". *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. 2017, pp. 1–8.

4.2 RESOURCE ALLOCATION SCHEMA

The algorithm described above has the advantage of being parallellizable on three different levels: one can parallelize computation across source vertices, across the edges of each source vertex, and across counting intersections of the vertices on each edge. By varying the number of computations running in parallel at each one of these levels, we can adapt the kernel to the structure of the graph. We describe the work distribution schema over these three levels using the parameters k, a, and g. We define k as the number of source vertices being processed in parallel, a as the number of edges of a source vertex whose Jaccard Weights are being calculated in parallel, and g as the number of threads counting neighborhood intersections of the vertices on a single edge. For example, setting $k = 5$, $a = 4$, and $g = 32$ for a kernel means that $5 \times 4 \times 32 = 640$ threads are running concurrently. These threads are split into five "assemblies" of $4 \times 32 = 128$ threads, each processing a single source vertex. Each assembly is split into four "groups" of 32 threads, with every group counting neighborhood intersections at a single edge.

Given this level of resource allocation configurability, one might proceed by assigning a single resource allocation schema for an input graph. However, we find that a single configuration for the entire graph is not enough to maximize performance; configuration at a finer granularity is needed. Real world graphs, especially those that follow a small-world distribution, exhibit a very wide range of degrees across their vertices. In such graphs, even though the degree of the majority of vertices is small, a small minority of vertices has exponentially higher degrees. Take for instance the HYPERLINK2012[64] graph. Its average degree is 29.5, but its maximum degree is 4,309,221. Clearly, assigning the same number of threads to calculate the Jaccard weights of every edge will lead to some threads taking on exponentially more work than other threads, and hence, to unwanted thread idleness. To tackle this problem, we introduce finer configuration granularity by separating vertices according to their degrees into mutually exclusive bins and assigning each bin its own configuration.

4.3 MACHINE LEARNING FOR RESOURCE ALLOCATION

Assigning a distribution schema for a vertex bin in a graph is not a trivial task. The right values for a certain bin in a graph depend on many aspects. For instance, the number of vertices in the bin, degree distributions, and the sparsity pattern of the graph can all affect how well a certain work distribution works with some input bin. Additionally, the architectural properties of the GPU being used affect the distribution schema. Computation speed, memory size and communication bandwidth all play a role in determining the best schema. To capture the high complexity of this problem, we train a multi-label, multi-class, supervised machine learning model to use structural features of vertex bins to assign them the most suitable work distribution schemas.

The process of building the model proceeds as follows: given a certain GPU architecture and a set of real-world training graphs that vary in scale and degree distribution, each input graph is binned and kernels are executed on each bin using all legal distribution schemes. Then, for each bin, we extract the distribution schema with the best performance, and use that as a training sample. We use structural features of the bin as the sample's features, and its schema values as the labels of the sample. Specifically, for each sample, we use the average and the maximum degrees of the graph, the maximum and minimum degrees of the bin, and the size of the bin as training features. We find that even such a small set of features is sufficient to build an accurate model. After extracting a single training sample per bin for every graph, we use these samples

[64] R. Meusel. "The Graph Structure in the Web – Analyzed on Different Aggregation Levels". *J. of Web Science* 1 (2015), pp. 33–47.

to train a Random Forest classifier to predict work distribution schemas. Note that we capture the architectural properties of different GPU architectures by training a separate model for every architecture.

A model need only be trained once per GPU architecture. At algorithm deployment time, the graph is binned and the resource allocation schemas are inferred from the trained model in negligible time.

4.4 EVALUATION

We evaluate the model using a suite of 12 graphs from different sources. These graphs and their key properties are shown in Table 9. When evaluating our approach on a certain target graph, we randomly select 10 other graphs as training graphs and train a model using their runtime data as described in Section 4.3. We use this model to predict the distribution schema of the target graph. We execute our experiments on a system equipped with an NVIDIA A100 GPU with 80 GB of memory, a 64-core AMD7742 CPU running on 2.24 GHz, and 512GB memory. Our GPU kernels are implemented in `CUDA` and compiled with `CUDA 11.3` using `-O3` as the optimization flag. We use the Random Forest module in the scikit-learn library[65] as our machine learning classifier. When reporting the runtime of our approach, we include the time to bin the vertices of the graph, carry out the inference using the trained machine learning model, and execute the computation kernels. We do not include the time to copy the data to and back from the GPU memory since it's common between all the evaluated approaches.

Table 9 *Graphs used for evaluating the Jaccard Weight algorithm.*

Graph	|V|	|E|	Max Degree	Avg. Degree
youtube[66]	1,138,499	5,980,886	28,754	5.25
wiki-topcats[67]	1,791,489	50,888,414	238,342	28.41
com-orkut[68]	3,072,627	234,370,166	33,313	76.28
com-lj[69]	3,997,962	69,362,378	14,815	17.35
soc-livejournal[70]	4,847,571	85,702,474	20,333	17.68
indochina-2004[71]	7,414,866	301,969,638	256,425	40.72
wb-edu[72]	9,845,726	92,472,210	25,781	9.39
uk-2002[73]	18,520,344	523,574,516	194,955	28.27
hyperlink2012[74]	39,497,204	1,165,134,582	4,309,221	29.50
twitter[75]	41,652,230	2,405,026,092	2,997,487	57.74
soc-sinaweibo[76]	58,655,849	522,642,066	278,489	8.91
com-friendster[77]	65,608,366	3,612,134,270	5,214	55.06

4.4.1 PERFORMANCE AGAINST PREVIOUS STATE OF THE ART

Table 10 compares the Jaccard computation runtime using our approach with two state-of-the-art implementations:

- **cuGraph** a graph library from NVIDIA's RAPIDS.[78] It includes a Jaccard Weights kernel which is an implementation of the algorithm proposed by Fender et al.[79]

[65] Pedregosa et al., "Scikit-learn: Machine Learning in Python".

[78] RAPIDS Development Team. *RAPIDS: Collection of Libraries for End to End GPU Data Science*. 2018. URL: https://rapids.ai.

[79] Fender et al., "Parallel Jaccard and related graph clustering techniques".

- **Anzt et al.**[80] a Jaccard Weight computation kernel that leverages the thread divergence capabilities, i.e., *independent thread scheduling*, of recent GPU architectures.

The proposed algorithm consistently outperforms **cuGraph**, with an average speed-up of 12×. The same variant outperforms **Anzt et al.** on all but one graph and is, on average, 27× faster. As expected, the proposed technique is exceptionally beneficial for graphs with large-degree hubs such as `hyperlink2012`. On this graph, our approach is 35× and 211× faster than **cuGraph** and **Anzt et al.**, respectively.

Table 10 *Comparing Jaccard computation runtimes of our proposed ML-based approach and two state-of-the-art algorithms.*

Graph	Anzt et al.	cu Graph	Proposed approach	Graph	Anzt et al.	cu Graph	Proposed approach
`youtube`	0.05	0.19	**0.01**	`wiki-topcats`	3.35	2.93	**0.08**
`com-orkut`	1.53	2.49	**0.50**	`com-lj`	0.11	0.20	**0.08**
`soc-LiveJournal`	0.21	0.46	**0.12**	`indochina-2004`	11.97	8.00	**3.61**
`wb-edu`	**0.14**	0.18	0.16	`uk-2002`	3.76	2.14	**0.49**
`hyperlink2012`	1786.83	302.48	**8.44**	`soc-sinaweibo`	3.69	41.41	**1.29**
`twitter`	4967.34	327.88	**28.44**	`com-friendster`	27.66	26.24	**10.97**

4.4.2 Evaluating effectiveness of the machine learning model

Figure 1 presents the improvement acquired by using the machine learning model to predict work distribution schemas. The figure shows speed-up values of three different variants of our approach. All three variants use the vertex binning method in which vertices are placed in bins based on their degrees. However, they differ in how the work distribution schemas are assigned for each bin. **Oracle** is a hypothetical algorithm obtained by handpicking the best work distribution schema for each bin and each graph. **ML-based** is the approach in which a machine learning model is used to select the work distribution schema for each bin (our proposed algorithm). **Rule-based** uses ad-hoc work distribution rules that we empirically derive through experimentation. Each point on the figure shows the speed-up achvieved using the respective model over a version of algorithm that does not use binning and simply executes a single kernel for the entire graph. We find that **Oracle** can produce up to 40% speed-up over the base approach, clearly demonstrating the importance of smart resource allocation for graph algorithms. In addition, we find that the **ML-based** approach is only 2.76% slower than **Oracle**, showing the effectiveness of our machine learning model in selecting the right work distribution schema for each graph.

[80] Hartwig Anzt and Jack Dongarra. "A Jaccard Weights Kernel Leveraging Independent Thread Scheduling on GPUs". *30th International SBAC-PAD*. 2018, pp. 229–232. DOI: 10.1109/CAHPC.2018.8645946.

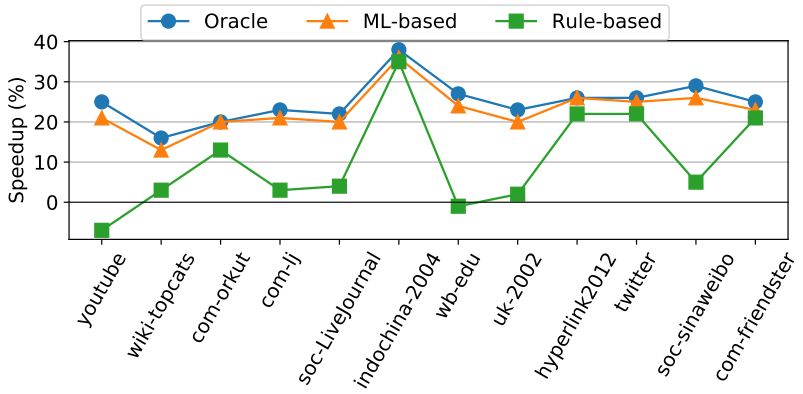

Figure 1 *Speed-up percentages achieved by using vertex binning approaches over running a single kernel for the entire graph.*

5 CONCLUSIONS

Exclusive research on ML-based prediction models in the areas of sparse matrices, tensors and graphs is performed. The target platforms are selected as one of the most widely-used applications in the literature for each category, and the models are extensible for other applications as well, which sheds the light on future research. We have presented the evaluation results of the proposed models for sparse matrices and graphs, which demonstrate the effectiveness of the models. The work regrading sparse tensors is on the stage of development and is planned to be finalized within next 6 months, as the work plan of this task permits. We have investigated several ML-based models to determine the most suitable one for predicting the reordering method for tensor decomposition. Since the main challenge in this area is the vast diversity in the mode sizes and their structure, we will train suitable ML-based models via utilizing an extensive set of sparse tensor features.

REFERENCES

Abadi, Martin et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

Abu-Sufah, Walid and Asma Abdel Karim. "Auto-tuning of sparse matrix-vector multiplication on graphics processors". *International Supercomputing Conference*. Springer. 2013, pp. 151–164.

Acer, Seher, Tugba Torun, and Cevdet Aykanat. "Improving medium-grain partitioning for scalable sparse tensor decomposition". *IEEE Transactions on Parallel and Distributed Systems* 29.12 (2018), pp. 2814–2825.

Anzt, Hartwig and Jack Dongarra. "A Jaccard Weights Kernel Leveraging Independent Thread Scheduling on GPUs". *30th International SBAC-PAD*. 2018, pp. 229–232. DOI: 10.1109/CAHPC.2018.8645946.

Barreda, Maria et al. "Performance modeling of the sparse matrix–vector product via convolutional neural networks". *The Journal of Supercomputing* 76.11 (2020), pp. 8883–8900.

Bell, Nathan and Michael Garland. *Efficient Sparse Matrix-Vector Multiplication on CUDA*. Tech. rep. NVR-2008-004. NVIDIA, 2008.

— "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors". 2009, 18:1–18:11.

Benatia, A. et al. "Sparse Matrix Format Selection with Multiclass SVM for SpMV on GPU". 2016, pp. 496–505.

Bian, Bian et al. "CSR2: A New Format for SIMD-accelerated SpMV". 2020, pp. 350–359.

Boldi, Paolo and Sebastiano Vigna. "The WebGraph Framework I: Compression Techniques". *WWW*. ACM Press, 2004, pp. 595–601.

Chen, Donglin et al. "Optimizing sparse matrix–vector multiplications on an armv8-based many-core architecture". *International Journal of Parallel Programming* 47.3 (2019), pp. 418–432.

Chicco, Davide and Giuseppe Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". *BMC genomics* 21.1 (2020), pp. 1–13.

Choi, Jee W., Amik Singh, and Richard W. Vuduc. "Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs". 2010, pp. 115–126.

Comaniciu, Dorin and Peter Meer. "Mean shift: A robust approach toward feature space analysis". 24.5 (2002), pp. 603–619.

Davis, Timothy and Yifan Hu. "The University of Florida Sparse Matrix Collection". *ACM Transactions on Mathematical Software (TOMS)* 38.1 (2011), pp. 1–25. ISSN: 1557-7295. DOI: 10.1145/2049662.2049663.

Dhandhania, Sunidhi et al. "Explaining the Performance of Supervised and Semi-Supervised Methods for Automated Sparse Matrix Format Selection". *50th International Conference on Parallel Processing Workshop*. 2021, pp. 1–10.

Fender, Alexandre et al. "Parallel Jaccard and related graph clustering techniques". *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. 2017, pp. 1–8.

Fishman, Matthew, Steven White, and Edwin Stoudenmire. "The ITensor software library for tensor network calculations". *SciPost Physics Codebases* (2022), p. 004.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Guo, Ping and Changjiang Zhang. "Sparse Matrix Selection for CSR-Based SpMV Using Deep Learning". *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. IEEE. 2019, pp. 2097–2101.

Intel Software. *Intel oneAPI Math Kernel Library*. Online. 2021.

Jeon, Inah et al. "Haten2: Billion-scale tensor decompositions". *2015 IEEE 31st international conference on data engineering*. IEEE. 2015, pp. 1047–1058.
Krishna, K and M Narasimha Murty. "Genetic K-means algorithm". *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29.3 (1999), pp. 433–439.
Leskovec, Jure and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. 2014.
Li, Jiajia, Yuchen Ma, and Richard Vuduc. *ParTI!: A parallel tensor infrastructure for multicore CPUs and GPUs*. 2018.
Li, Jiajia et al. "Efficient and effective sparse tensor reordering". *Proceedings of the ACM International Conference on Supercomputing*. 2019, pp. 227–237.
Li, Jiajia et al. "SMAT: An Input Adaptive Auto-tuner for Sparse Matrix-vector Multiplication". 2013, pp. 117–126.
Li, K., W. Yang, and K. Li. "Performance Analysis and Optimization for SpMV on GPU Using Probabilistic Modeling". 26.1 (2014), pp. 196–205.
Li, Yishui et al. "VBSF: a new storage format for SIMD sparse matrix–vector multiplication on modern processors". *The Journal of Supercomputing* 76 (2019), pp. 2063–2081.
Liu, Weifeng and Brian Vinter. "CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication". 2015, pp. 339–350.
Lloyd, S. P. "Least Squares Quantization in PCM". 28.2 (1982), pp. 129–136.
Matthews, Brian W. "Comparison of the predicted and observed secondary structure of T4 phage lysozyme". *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405.2 (1975), pp. 442–451.
Meusel, R. "The Graph Structure in the Web – Analyzed on Different Aggregation Levels". *J. of Web Science* 1 (2015), pp. 33–47.
Mislove, A. et al. "Measurement and Analysis of Online Social Networks". *Proc. of the 7th ACM SIGCOMM*. 2007, pp. 29–42.
Mohammed, Thaha et al. "DIESEL: A novel deep learning-based tool for SpMV computations and solving sparse linear equation systems". *The Journal of Supercomputing* 77.6 (2021), pp. 6313–6355.
Nisa, I. et al. "Effective Machine Learning Based Format Selection and Performance Modeling for SpMV on GPUs". *International Workshop on Automatic Performance Tuning*. 2018, pp. 1056–1065.
Nisa, Israt et al. "Load-balanced sparse MTTKRP on GPUs". *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2019, pp. 123–133.
NVIDIA Developer. *CUSP*. Online. 2021.
— *cuSPARSE*. Online. 2021.
Pedregosa, F. et al. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
Phipps, Eric T and Tamara G Kolda. "Software for sparse tensor decomposition on emerging computing architectures". *SIAM Journal on Scientific Computing* 41.3 (2019), pp. C269–C290.
Pichel, J. C. and B. Pateiro-López. "A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques". 2018, pp. 46–54.
Rossi, R. A. and N. K. Ahmed. *The Network Data Repository with Interactive Graph Analytics and Visualization*. 2015. URL: http://networkrepository.com.
Saad, Y. *SPARSKIT: a basic tool kit for sparse matrix computations*. Tech. rep. RIACS-TR-90-20. Research Institute for Advanced Computer Science, 1990.
Sedaghati, N. et al. "Automatic Selection of Sparse Matrix Representation on GPUs". 2015, pp. 99–108.
Smith, Shaden et al. *FROSTT: The formidable repository of open sparse tensors and tools*. 2017.

Smith, Shaden et al. “SPLATT: Efficient and parallel sparse tensor-matrix multiplication”. *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2015, pp. 61–70.

Sun, Qingxiao et al. “Input-aware Sparse Tensor Storage Format Selection for Optimizing MTTKRP”. *IEEE Transactions on Computers* (2021).

Sun, Qingxiao et al. “Sptfs: Sparse tensor format selection for mttkrp via deep learning”. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2020, pp. 1–14.

Tan, Guangming, Junhong Liu, and Jiajia Li. “Design and Implementation of Adaptive SpMV Library for Multicore and Many-Core Architecture”. 44.4 (2018), 46:1–46:25. ISSN: 0098-3500.

Team, RAPIDS Development. *RAPIDS: Collection of Libraries for End to End GPU Data Science*. 2018. URL: https://rapids.ai.

Valiev, Marat et al. “NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations”. *Computer Physics Communications* 181.9 (2010), pp. 1477–1489.

Xie, B. et al. “CVR: Efficient Vectorization of SpMV on x86 Processors”. 2018, pp. 149–162.

Xie, Zhen et al. “IA-SpGEMM: An Input-Aware Auto-Tuning Framework for Parallel Sparse Matrix-Matrix Multiplication”. Association for Computing Machinery, 2019, pp. 94–105.

— “IA-SpGEMM: An input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication”. *Proceedings of the ACM International Conference on Supercomputing*. 2019, pp. 94–105.

Zhang, T., R. Ramakrishnan, and M. Livny. “BIRCH: An Efficient Data Clustering Method for Very Large Databases”. *ACM Sigmod Record* 25.2 (1996), pp. 103–114.

Zhao, Yue et al. “Bridging the Gap Between Deep Learning and Sparse Matrix Format Selection”. 2018, pp. 94–108.

— “Bridging the gap between deep learning and sparse matrix format selection”. *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*. 2018, pp. 94–108.

Zhao, Yue et al. “Overhead-Conscious Format Selection for SpMV-Based Applications”. 2018, pp. 950–959.

Zhou, W. et al. “Enabling Runtime SpMV Format Selection through an Overhead Conscious Method”. 31.1 (2020), pp. 80–93.