Reimagine Application Performance as a Graph: Novel Graph-Based Method for Performance Anomaly Classification in High-Performance Computing

Chase Phelps*, Ankur Lahiry*, Tanzima Z. Islam*, Line C. Pouchard†
*Texas State University, †Brookhaven National Laboratory,
{chaseleif, vty8, tanzima}@txstate.edu, pouchard@bnl.gov

***Abstract*—Performance anomaly in High Performance Computing (HPC) can be defined as run-to-run variation of an application in repeated runs with the same set of configuration parameters. Such variations can occur for myriad reasons, including contention for shared resources such as the network and dynamic data distribution across application processes. Traditionally HPC researchers focus on real-time anomaly detection using different Machine Learning (ML) methods. These popular methods, such as auto-encoder, limit finding anomalous event patterns during training time. On top of that, in HPC, performance data are stored in tabular format. Though gradient-based methods have already proved their significant improvement over classification tasks, they explicitly use feature-feature relationships, ignoring the potential sample-sample relationship. To fill this gap, we build a performance anomaly classification technique leveraging the potential graph-based representation learning. We hypothesize that a meaningful and robust representation considering the sample-sample relationship for the given tabular datasets will improve the downstream anomaly classification technique. We conduct our experiment on 5 HPC datasets and 6 ML datasets. Our empirical study proves that graph-based anomaly classification outperforms the gradient-based approaches in 6 out of 11 experiments. We also explain how anomaly decisions are made inside the performance graph.**

***Index Terms*—High-Performance Computing, Graph Neural Network, Performance Analytics, Anomaly Classification**

I. Introduction

HPC systems consist of billions of components from compute nodes, memories, networks, and storage, where these components interact with each other in a complex manner. To ensure the applications are effectively making progress, HPC facilities deploy real-time performance monitoring tools, such as `Chimbuko` [1], [2] to monitor system performance. Despite significant improvements in HPC, different hardware heterogeneity, resource contention, system load, and external issues such as human errors are responsible for generating performance variations in applications. Performance anomalies manifest as a variation in the execution time of the same application run with the same configurations multiple times.

Although real-time performance monitoring systems such as `Chimbuko` detect anomalies in real-time, they don't answer if there is a pattern of anomalies in the system. If system administrators know patterns in anomalous events, they can anticipate and take preventive actions against those anomalies, helping to build a more reliable and robust system. In this work, we propose a new method for classifying performance anomalies and explain the factors affecting an ML model's determination of sample anomaly.

Tree-based gradient descent algorithms, such as XGBoost [3] and LightGBM [4], have shown significant effectiveness in improving the fidelity of a classifier in different domains [5]–[8]. Additionally, the Deep Neural Network (DNN)-based approach is very popular for different domains such as [9]–[12]. However, gradient-based and DNN based algorithms explicitly work on feature-feature relationships, without exploring the potential of sample-sample relationships in a given tabular dataset. Our work aims to bridge the gap by transforming tabular data into a graph that models both sample and feature relationships. We leverage a state-of-the-art Graph Neural Network (GNN)-based representation learning technique to learn embeddings from the performance samples. These embeddings are then used to train a binary classifier to predict whether a previously unseen performance sample is anomalous.

However, unlike other research domains, graphs aren't given in the performance analysis domain, they need to be built. A related work, [13], proposes building graphs from tabular data for regression tasks, our work specifically addresses classification. While our graph-building method is similar to [14], their work uses an inductive graph learning method for ML classification datasets. In contrast, our work studies 2 graph learning approaches, defines a new distance measure for considering domain-specific features, and applies this approach to HPC performance data.

To summarize, the contributions made in this paper are:

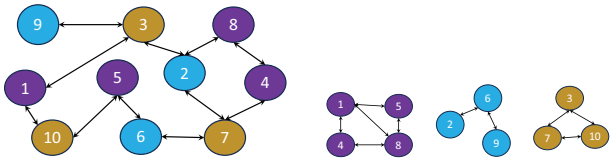
- Study the impact of different graph learning methods on the effectiveness of embeddings;
- Leverage both numerical and domain-specific features during embedding learning;
- Compare how graph-based representation learning impacts the efficacy of downstream classification tasks.

II. Background

A. Graph Learning Methods

Transductive and inductive learning are approaches used in graph-based ML, particularly in the context of GNNs and graph-based semi-supervised learning. This section provides

2836-3795/24/$31.00 ©2024 IEEE
DOI 10.1109/COMPSAC61105.2024.00041



(a) Transductive graph learning. All nodes are within a single graph structure, nodes are sampled to create training, validation, and test sets.

(b) Inductive graph learning. Nodes are sampled to create training, validation, and test sets, from which 3 disjoint graphs are created.

Fig. 1: Comparison of graph-based learning methods. The colors indicate the training set–purple, the validation set–blue, and the test set–gold.

a high-level overview of these methods. Later, we study the performance of building graphs from tabular datasets using these methods.

1) Transductive Learning

In transductive learning [15], a model learns to predict labels for nodes or edges in a graph based on the information available within the entire graph structure and features of the observed data points. From Figure 1(a) we see that though nodes are split into different sets, they are still connected, such that the *graph structure* remains unchanged during training and testing. Though there exist connections from the test set to other sets, only labels from samples within the train set are available during training to ensure the model does not train using labels of nodes from another set.

2) Inductive Learning

In inductive learning [16], a model is trained using disjoint graphs to predict the label of an unseen node or edge. The data for inductive learning is split as shown in Figure 1(b). Models trained using graphs created for inductive learning generalize better than those created using the transductive approach. A key benefit of the inductive learning approach is that it is more scalable, as the representations learned for node classification are more localized and rely less on a node's position within the larger graph.

III. Our Approach

A. Problem Formulation

We propose reimagining a tabular performance dataset as a graph to build an effective representation of performance samples. The rationale for creating graphs from tabular performance datasets is that a graph's structure can describe inherent relationships among performance samples and features within a dataset. However, unlike other relational graphs, these relationships are not explicitly provided in a tabular performance dataset.

With the construction of a graph, embeddings for each node can be learned, and the task of anomaly classification can be reduced to the prediction of node labels. We hypothesize that capturing the sample-sample relationships will make the sample embedding more meaningful, improving the downstream anomaly classifier's efficacy.

B. Building Initial Graph from Performance Data

Figure 2 shows the overall pipeline of our work. In the first step, we transform the tabular data to a graph, $G = (V, E)$, where V is the set of all vertices and E is the set of all edges. Specifically, we map each sample of the tabular dataset into a node $V_i \in \mathrm{V}$. To create edges, we begin by computing the pairwise distance between each pair of samples using a distance measure, as described in Section III-C. For a given sample, we choose the top N nearest neighbors as neighbors with the N lowest distances, and create edges with weights $1 - distance$, where $distance$ is between 0 and 1. Each edge, $e \in \mathrm{E}$, connects 2 samples based on their feature values.

C. Distance Measures

HPC datasets often contain a mixture of both numerical and domain-specific features, e.g., algorithm names, hardware performance counters, and callstacks. Hence, while building a graph, we need to develop a data-type-aware distance measure. Considering the use case of a real-time performance monitoring and anomaly detection framework, `Chimbuko`, columns in the tabular format are the features while the rows are the samples. Most features are numeric, except the domain-specific *callstack* column. Numeric features are normalized by column to be between 0 and 1.

Numeric features: The Euclidean distance between 2 samples is calculated as the sum of the absolute value of the difference of their feature vectors divided by the length of a feature vector. Given **X** and **Y**, 2 feature vectors, the Euclidean distance is calculated using Equation 1. Here, $\ell(X)$ is the length of the vector **X**.

$$d_{euclid}(X, Y) = \frac{\sum |X - Y|}{\ell(X)} \tag{1}$$

Domain-specific features: We define custom distance functions to calculate the similarity between domain-specific features that are non-numeric. Specifically, the HPC `Chimbuko` performance monitoring framework captures the callstack of logged events to describe the provenance of performance anomalies. Each callstack is comprised of many parameters, e.g., function id (FID), entry timestamp ($entry$), and exit timestamp ($exit$). We find the symmetric difference between 2 sets using FIDs. A greater difference indicates less similarity between 2 callstacks. If $C1$ and $C2$ represent the callstacks of samples **X** and **Y**, the callstack distance is found using Equation 2 where $\ell(C1)$ and $\ell(C2)$ are the lengths of callstacks $C1$ and $C2$.

$$d_{callstack}(X, Y) = \frac{2 \times \ell(C1 \cap C2)}{\ell(C1) + \ell(C2)} \tag{2}$$

Equation 2 yields a distance between 0, when 2 callstacks include the same FIDs, and 1 when there are no common FIDs, and could be applied to any such categorical feature.

In graph creation, we combine the feature distance with ratio **R** of the callstack distance using Equation 3, where **R** is $1/2$, $1/3$, or 0 for `Chimbuko` datasets and 0 for all others.

$$distance = d_{feature} \times (1 - R) + d_{callstack} \times R \tag{3}$$

D. Learning Embeddings from Graphs

After creating a graph, we leverage a GNN-based method to learn node embeddings. In this study, we evaluate both transductive and inductive approaches. **To our knowledge, this is the first work to evaluate both methods in the context of performance graph learning.**

Fig. 2: Pipeline of our graph-based anomaly classification method: we map each sample as a node of a graph, calculate pairwise distances between nodes, connect each node to their nearest N neighbors with an edge-weighted by their distance, then learn and create node embeddings using GNN.

E. Explainability of our methods

GNNExplainer: GNNExplainer [17] is an innovative method to improve the interpretation of Graph Neural Networks. It provides researchers and practitioners with a means to visualize semantically relevant subgraph structures, providing insight into the underlying patterns that the GNN has learned during training.

SHAP: SHapley Additive exPlanations (SHAP) [18] is a game-theoretic approach to explain the output of machine learning models. Each element is assigned a value of importance, which reveals its influence on model forecasts. By showing the effect of individual features on results, this method increases model transparency and interpretation.

F. Parallelization of Graph Creation

To reduce graph creation time for large datasets, we use `ProcessPoolExecutor` from `concurrent.futures` to parallelize the method. Each subprocess receives a copy of the graph without edges, the number of processes, and its process ID, then creates an empty graph, adding edges for nodes starting from its process ID and stepping by the number of processes. Completed graphs are merged into the original, resulting in a final graph with all nodes, edges, and data.

IV. Implementation

A. Software and Libraries

Our pipeline is implemented using Python packages and libraries including Scikit-learn [19], NetworkX [20], and DGL [21].

B. GNN Architecture

We leverage the following in the GNN-based model:

Activation: The Leaky ReLU [22] activation introduces non-linearity, enabling complex functional relationships.

Dropout: Dropout layers are employed to improve robustness and limit overfitting.

Optimizer: The Adam Optimizer [23] is used to adjust model parameters effectively to minimize the loss function.

Scheduler: The ReduceLROnPlateau scheduler controls the learning rate to improve convergence during training.

V. Experimental Setup

A. System

Texas Advanced Computing Center (TACC) provides supercomputing facilities to researchers. We leverage the Lonestar6 computing cluster for our experiments. Each compute node is comprised of 2 AMD EPYC 7763 64-core (Milan) CPUs and 256 GB of DDR4 memory.

B. Datasets

We evaluate our graph-based performance anomaly classification approach with 5 HPC datasets. To evaluate the generalizability of our approach, we also assess performance using 6 ML benchmark datasets. Table I describes each dataset.

TABLE I: Description of anomaly classification datasets.

Dataset	Samples	Features	Anomaly instances	%-of-Anomaly
singlereuse [1]	3349	19	1787	53.36
allreuse [1]	23286	20	13654	58.63
singlerepeat [1]	53034	18	40034	75.48
allrepeat [1]	144771	20	106109	73.29
seismic [24]	2584	15	170	6.57
phoneme [25]	5404	5	1586	29.34
satimage [26]	5803	36	71	1.22
musk [27]	6598	166	1017	15.41
bank [28]	45211	10	5289	11.70
smtp [29]	95156	3	30	0.03
1000-genome [30]	20687	30	5173	25

C. Anomaly Classification Baselines

In this work, we evaluate 3 different graph-based representation learning methods—Graph Convolutional Network (GCN) [31], Graph Attention Network (GAT) [32], and GraphSAGE [16]. We compare our proposed graph-based anomaly classification techniques with those of several baseline methods commonly used as classification for tabular data. Specifically, we compare with XGBoost [3], LightGBM [4], and DNN [33].

D. Evaluation Metrics

We use three different metrics for evaluating our model: accuracy, F1 score, and Mathews Correlation Coefficient (MCC) score. Accuracy is the ratio of true predictions to the total number of predictions, and is given by the formula $\frac{TP+TN}{TP+FP+TN+FN}$. The F1 score is the harmonic mean of precision, where precision is the ratio of instances predicted to be positive that were correct, and is given by the formula $\frac{2\times TP}{2\times TP+FN+FP}$. The MCC [34] score captures the quality of a binary classification model's predictions, works well with imbalanced data, and is given by the formula $\frac{TN\times TP-FP\times FN}{\sqrt{(TN+FN)(FP+TP)(TN+FP)(FN+TP)}}$.

VI. Results

In this section, we evaluate the effectiveness of our proposed graph-based anomaly classification method using various GNN architectures. Specifically, we report the accuracy, F1 score, and MCC score for 11 datasets.

- RQ1: Does the learning method (transductive vs inductive) during graph building impact the effectiveness of anomaly classification methods?
- RQ2: Does the minimum number of neighbors impact the effectiveness of graph-based anomaly classification methods?
- RQ3: What is the overall performance of different anomaly classification methods?
- RQ4: Can we explain "why" performance samples are classified as anomalous?

TABLE II: F1 score, MCC score, and accuracy for each method by dataset. The graph creation parameters used were those which achieved the highest F1 and MCC scores with the SAGE classifier. The highest MCC score of each row is highlighted.

Dataset	XGboost			LightGBM			DNN			GAT			GCN			SAGE			Parameters		
	F1	MCC	ACC	F1	MCC	ACC	F1	MCC	ACC	F1	MCC	ACC	F1	MCC	ACC	F1	MCC	ACC	T/I	N	C
singlerepeat	0.92	0.63	86.96	0.92	0.63	87.22	0.58	-0.05	47.69	0.89	0.63	83.98	0.90	0.68	86.09	0.94	0.79	91.39	T	7	1/3
singlereuse	0.80	0.53	76.90	0.83	0.58	79.73	0.29	0.00	45.88	0.81	0.59	79.64	0.81	0.61	79.93	0.93	0.85	92.42	T	2	0
allrepeat	0.91	0.65	86.79	0.91	0.62	85.98	0.31	0.00	36.21	0.89	0.64	84.08	0.89	0.68	85.15	0.86	0.64	81.39	T	7	1/2
allreuse	0.86	0.65	83.17	0.88	0.69	84.85	0.47	0.00	48.27	0.74	0.43	71.33	0.76	0.48	73.61	0.83	0.64	81.51	T	2	0
1000-genome	0.85	0.80	92.60	0.86	0.82	92.93	0.22	0.00	65.00	0.80	0.73	88.97	0.82	0.76	90.65	0.84	0.79	91.13	T	3	-
seismic	0.17	0.15	92.26	0.13	0.15	93.11	0.14	0.08	51.03	0.30	0.25	84.98	0.29	0.23	86.80	0.37	0.33	93.42	I	6	-
phoneme	0.81	0.73	88.75	0.82	0.74	89.31	0.39	0.03	51.55	0.74	0.62	82.40	0.74	0.62	81.48	0.76	0.65	83.77	I	2	-
satimage	0.93	0.93	99.83	0.92	0.92	99.81	0.53	0.58	97.99	0.86	0.86	99.62	0.85	0.85	99.67	0.89	0.89	99.69	T	6	-
musk	0.92	0.90	97.50	0.95	0.94	98.50	0.35	0.21	66.98	0.91	0.90	97.07	0.77	0.73	93.01	0.93	0.92	97.79	T	7	-
bank	0.45	0.39	88.69	0.47	0.44	90.14	0.20	0.03	52.20	0.53	0.47	85.77	0.51	0.44	86.01	0.53	0.47	85.23	I	4	-
smtp	0.33	0.34	99.97	0.00	0.00	98.73	0.06	0.00	80.00	0.49	0.50	99.96	0.75	0.77	99.99	0.75	0.77	99.99	T	2	-

The following sections describe the results in detail.

A. RQ1: Impact of graph learning method

This experiment aims to study the impact of 2 different graph learning methods—transductive and inductive—and the impact of the minimum number of neighbors N on the effectiveness of the models, as explained in Section II-A. The X-axis of Figures 3a and 3b show results from varying the minimum number of neighbors when building graphs for the HPC datasets listed in Table I. The Y-axis shows the corresponding F1-score metric using the GraphSAGE method for learning embeddings with default hyperparameters: 5 layers, 256 hidden features, each internal layer's aggregator type is `gcn`, and the output layer's aggregator type is `pool`. Based on the results shown in Figure 3, we can conclude that while the HPC datasets perform better using transductive graph learning, half of the ML datasets benefit more from inductive graph learning.

B. RQ2: Impact of the minimum number of neighbors

Figure 3 shows the effect of varying the minimum number of neighbors during graph creation. As expected, increasing N increases the number of edges created and the memory footprint of the graph, through graph creation runtime only increases marginally as increasing N does not add additional computation. We observe that the best minimum number of neighbors, N, is at least 2 and is better when higher in the ML datasets. This may be indicative that inductive learning allows for better generalization.

C. RQ3: What is the overall performance of different anomaly classification methods

The objective of this experiment is to study the overall performance of different anomaly classification methods using both HPC and ML datasets. In this experiment, we compare the graph-based methods, GraphSAGE, GAT, and GCN, with several baselines, XGBoost, LightGBM, and DNN. Details of different methods and their parameters are described in Section V-C.

Important attributes of each dataset are described in Table I. As seen in the smtp dataset, which has the lowest ratio of anomalous samples with only 30 of 95159 samples, accuracy is not always a meaningful metric. The F1 score describes

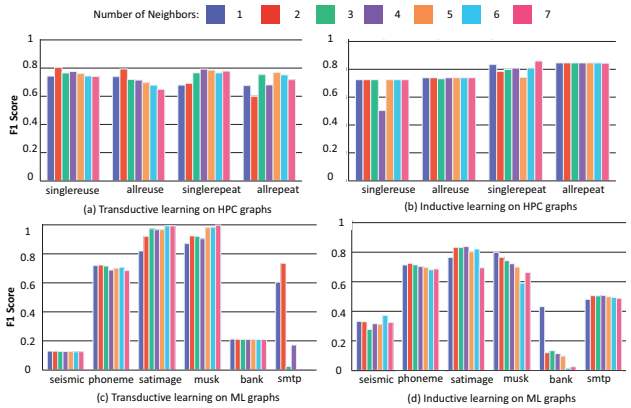


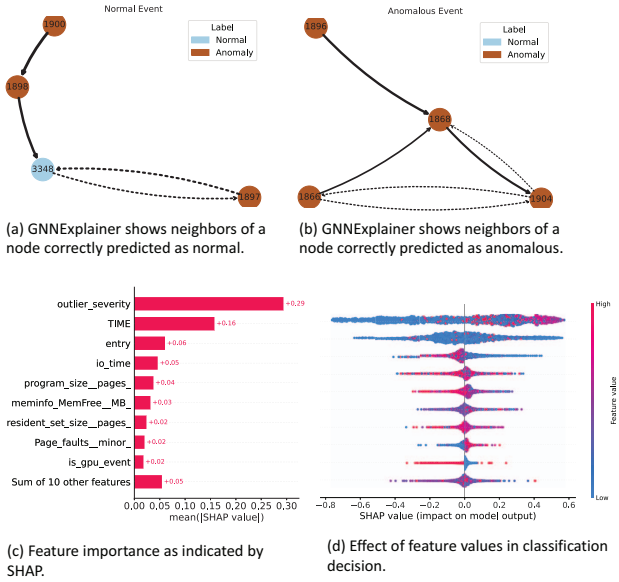
Fig. 3: Comparison of graph building methods using an untuned GraphSAGE model.

a model's ability to correctly classify positive cases, and the MCC score additionally describes a model's ability to correctly classify negative cases. For the GNN methods, both GCN and GraphSAGE achieve near-perfect classification accuracy by all chosen metrics for the extremely unbalanced smtp dataset. For all of the experiments, DNN based approach performs the worst. This may be due to less important features negatively affecting the DNN [35].

D. RQ4: Can we explain "why" performance samples are classified as anomalous?

This experiment aims to explain "why" the performance samples are classified as anomalous. To provide an explanation of the classification model's decisions, we present 2 pieces of information: (1) sample relations, and (2) feature importance to explain what could have caused the anomaly. To compute feature importance, we leverage the SHAP [18] library with XGBoost to analyze what could have caused the performance anomalies. To validate why a GNN model decides a new performance sample as anomalous, we leverage the GNNExplainer [17] library to identify "similar" samples.

Explanation: Figure 4a and Figure 4b show the neighborhoods of a correctly predicted normal and anomalous node, respectively. While the anomalous node is strongly connected



(a) GNNExplainer shows neighbors of a node correctly predicted as normal.

(b) GNNExplainer shows neighbors of a node correctly predicted as anomalous.

(c) Feature importance as indicated by SHAP.

(d) Effect of feature values in classification decision.

Fig. 4: Explainability of classifications made by the GraphSAGE classifier, (a) and (b), and the XGBoost classifier, (c) and (d), with the singlereuse dataset.

to other anomalous nodes, the normal node has weak connections to anomalous nodes. Further investigation shows that the feature values of the normal and anomalous events are the same, only the function runtime of the anomalous event is significantly larger than that of a normal sample. This observation validates that the model is able to correctly classify a node as anomalous because it does not conform to an expected behavior.

Feature importance: From Figure 4c, we observe that (1) features related to function runtime, outlier_severity and TIME, are the most important, followed by entry, the timestamp of the function entry. As described in Section V-B, performance events that occur toward the beginning of an application are more likely to be labeled as anomalous due to a lack of running statistics. Figure 4d indicates that higher values for function runtime, as well as lower values for entry timestamp, are correlated with anomalous events. Though providing a smaller contribution to the classification decision, we observe that minor page faults are correlated with anomalous events.

E. Discussions

Table II shows that our graph-based approach outperforms the baseline approaches in 6 of 11 experiments with respect to MCC scores. The benefit of parallelization is shown in Table III, with observed speedups of up to 10.7x. The greater the number of nodes, edges, and features, the greater benefit there is in parallelization of graph creation. Though we do not see a significant speedup for small graphs, we do not observe a slowdown.

VII. Related Work

A. Anomaly detection in HPC

Despite extensive research on anomaly detection in HPC, identifying anomalous instances is still an open research question. Borghesi et al. proposed an unsupervised [36], and later a semi-supervised [37] autoencoder-based anomaly detection method. The unsupervised autoencoder is trained solely using normal instances. Thus, it is unable to learn anomalous patterns during training, while the semi-supervised autoencoder contains a limited number of anomalous instances. The limitation of these approaches is that an unsupervised autoencoder only learns the underlying patterns of normal instances, not the characteristics of an anomalous representation. On the other hand, the semi-supervised approach is prone to overfit the model if there is not enough anomalous data.

Dey et al. [38] propose a signal-based anomaly detection in `Chimbuko` by formulating a novel metric to evaluate anomaly detection algorithms treating anomalies as noise. Prodigy [39], proposed by Aksar et al. uses multivariate time series telemetry data for finding out performance anomalies. However, those approaches are based on a time-series dataset, which we are not considering in this study.

B. Graph-Based Classification

TableGraphNet [40] is a DNN based approach that converts tabular data to graphs for each data sample and suffers from the poor performance of DNN-based methods. TabGNN [41] constructs a direct multiplex graph from the given tabular dataset using a DNN-based approach. These 2 approaches heavily rely on DNN, and our method outperforms DNN, implying significant improvement over those methods.

TABLE III: Comparison of time to build graphs for each dataset, serial times indicated with asterisks are extrapolated from 20 minutes of partial graph creation.

Dataset	Nodes	Features	Serial (s)	Parallel (s)	Speedup
singlerepeat	53034	18	*4800	765.18	*6.27x
singlereuse	3349	19	20.7	16.13	1.28x
allrepeat	144771	20	*58200	4776.89	*12.18x
allreuse	23286	20	975.45	94.63	10.31x
1000genome	20687	30	779.31	108.62	7.17x
seismic	2584	15	29.51	7.99	3.69x
phoneme	5404	5	113.59	19.67	5.77x
satimage	5803	36	66.81	6.52	10.25x
musk	6598	166	99.53	9.29	10.71x
bank	45211	10	8149.44	1286.51	6.33x
smtp	95156	3	15620.79	1490.33	10.48x

VIII. Conclusions

This paper demonstrates a graph-based representation learning for anomaly classification technique in HPC performance analysis domain. We convert our performance tabular data to a graph structure and later fed the graph to generate an effective representation learning for identifying anomalous patterns. We evaluate both our proposed methodology and state-of-the-art ML models on 11 datasets. Our model outperforms gradient decent-based approaches 6 out of 11 experiments in terms of MCC score. We also leverage parallelization techniques to build graphs, achieving a maximum 10.7x speed up from serial graph creation. We also include the interpretability of our graph-based method to explain the inherent pattern of both anomalous and normal events.

IX. Acknowledgement

This material is based upon work supported by the U.S. Department of Energy, Office of Science under Award Number DE-SC0023173.

References

[1] C. Kelly, S. Ha, K. Huck, H. Van Dam, L. Pouchard, G. Matyasfalvi, L. Tang, N. D'Imperio, W. Xu, S. Yoo *et al.*, "Chimbuko: A workflow-level scalable performance trace analysis tool," in *ISAV'20 In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, 2020, pp. 15–19.

[2] Brookhaven National Laboratory, "Chimbuko User Guide," https://chimbuko-performance-analysis.readthedocs.io/en/latest/io_schema/schema.html\#provenance-database-schema.

[3] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[5] A. Ogunleye and Q.-G. Wang, "Xgboost model for chronic kidney disease diagnosis," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 17, no. 6, pp. 2131–2140, 2019.

[6] L. Torlay, M. Perrone-Bertolotti, E. Thomas, and M. Baciu, "Machine learning–xgboost analysis of language networks to classify patients with epilepsy," *Brain informatics*, vol. 4, no. 3, pp. 159–169, 2017.

[7] W. Li, Y. Yin, X. Quan, and H. Zhang, "Gene expression value prediction based on xgboost algorithm," *Frontiers in genetics*, vol. 10, p. 1077, 2019.

[8] Y. Hua, "An efficient traffic classification scheme using embedded feature selection and lightgbm," in *2020 Information Communication Technologies Conference (ICTC)*. IEEE, 2020, pp. 125–130.

[9] T. S. Borkar and L. J. Karam, "Deepcorrect: Correcting dnn models against image distortions," *IEEE Transactions on Image Processing*, vol. 28, no. 12, pp. 6022–6034, 2019.

[10] A. Sento, "Image compression with auto-encoder algorithm using deep neural network (dnn)," in *2016 Management and Innovation Technology International Conference (MITicon)*. IEEE, 2016, pp. MIT–99.

[11] Y. Quan, H. Teng, Y. Chen, and H. Ji, "Watermarking deep neural networks in image processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 5, pp. 1852–1865, 2020.

[12] M. B. Sahaai *et al.*, "Brain tumor detection using dnn algorithm," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 11, pp. 3338–3345, 2021.

[13] T. Ramadan, A. Lahiry, and T. Z. Islam, "Novel representation learning technique using graphs for performance analytics," in *2023 International Conference on Machine Learning and Applications (ICMLA)*, 2023, pp. 1311–1318.

[14] S. B. Rabbani and M. D. Samad, "Between-sample relationship in learning tabular data using graph and attention networks," *arXiv preprint arXiv:2306.06772*, 2023.

[15] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[16] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[17] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *Advances in neural information processing systems*, vol. 32, p. 9240, 2019.

[18] A. B. Parsa, A. Movahedi, H. Taghipour, S. Derrible, and A. K. Mohammadian, "Toward safer highways, application of xgboost and shap for real-time accident detection and feature analysis," *Accident Analysis & Prevention*, vol. 136, p. 105405, 2020.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[21] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. J. Smola, and Z. Zhang, "Deep graph library: Towards efficient and scalable deep learning on graphs." *CoRR*, vol. abs/1909.01315, 2019. [Online]. Available: http://arxiv.org/abs/1909.01315

[22] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] M. Sikora and L. Wrobel, "seismic-bumps," UCI Machine Learning Repository, 2013, DOI: https://doi.org/10.24432/C5W902.

[25] P. Alinat and J.-M. Pierrel, "Esprit ii project 5516 roars robust analytic speech recognition system," 1994. [Online]. Available: https://api.semanticscholar.org/CorpusID:114700444

[26] A. Zimek, M. Gaudet, R. J. Campello, and J. Sander, "Subsampling for efficient and effective unsupervised outlier detection ensembles," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 428–436.

[27] D. Chapman and A. Jain, "Musk (Version 2)," UCI Machine Learning Repository, 1994, DOI: https://doi.org/10.24432/C51608.

[28] R. P. Moro, S. and P. Cortez, "Bank Marketing," UCI Machine Learning Repository, 2012, DOI: https://doi.org/10.24432/C5K306.

[29] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne, "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 320–324.

[30] G. Papadimitriou, H. Jin, C. Wang, K. Raghavan, A. Mandal, P. Balaprakash, and E. Deelman, "Flow-bench: A dataset for computational workflow anomaly detection," *arXiv preprint arXiv:2306.09930*, 2023.

[31] S. Haykin and N. Network, "A comprehensive foundation," *Neural networks*, vol. 2, no. 2004, p. 41, 2004.

[32] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[33] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[34] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC genomics*, vol. 21, no. 1, pp. 1–13, 2020.

[35] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 507–520.

[36] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in *Proceedings of the AAAI Conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 9428–9433.

[37] ——, "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems," *Engineering Applications of Artificial Intelligence*, vol. 85, pp. 634–644, 2019.

[38] A. Dey, T. Islam, C. Phelps, and C. Kelly, "Signal processing based method for real-time anomaly detection in high-performance computing," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2023, pp. 233–240.

[39] B. Aksar, E. Sencan, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, B. Kulis, M. Egele, and A. K. Coskun, "Prodigy: Towards unsupervised anomaly detection in production hpc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3581784.3607076

[40] G. Terejanu, J. Chowdhury, R. Rashid, and A. Chowdhury, "Explainable deep modeling of tabular data using tablegraphnet," *arXiv preprint arXiv:2002.05205*, 2020.

[41] X. Guo, Y. Quan, H. Zhao, Q. Yao, Y. Li, and W. Tu, "Tabgnn: Multiplex graph neural network for tabular data prediction," *arXiv preprint arXiv:2108.09127*, 2021.