Workflow Anomaly Detection with Graph Neural Networks

Hongwei Jin*, Krishnan Raghavan*, George Papadimitriou†, Cong Wang‡, Anirban Mandal‡,
Patrycja Krawczuk†, Loïc Pottier† Mariam Kiran§, Ewa Deelman†, Prasanna Balaprakash*
*Mathematics and Computer Science Division, Argonne National Laboratory, USA
{jinh, kraghavan, pbalapra}@anl.gov
†Information Sciences Institute, University of Southern California, USA
{georgpap, krawczuk, lpottier, deelman}@isi.edu
‡RENCI, University of North Carolina at Chapel Hill, USA
{cwang, anirban}@renci.org
§Energy Sciences Network (ESnet), Lawrence Berkeley National Laboratory, USA
mkiran@es.net

***Abstract*—Reliable execution of scientific workflows is a fundamental concern in computational campaigns. Therefore, detecting and diagnosing anomalies are both important and challenging for workflow executions that span complex, distributed computing infrastructures. In this paper we model the scientific workflow as a directed acyclic graph and apply graph neural networks (GNNs) to identify the anomalies at both the workflow and individual job levels. In addition, we generalize our GNN model to take into account a set of workflows together for the anomaly detection task rather than a specific workflow. By taking advantage of learning the hidden representation, not only from the job features, but also from the topological information of the workflow, our GNN models demonstrate higher accuracy and better runtime efficiency when compared with conventional machine learning models and other convolutional neural network approaches.**

***Index Terms*—Scientific workflows, Anomaly detection, Graph neural networks**

I. Introduction

Computational science today depends on many complex, data-intensive applications operating on datasets that originate from a variety of scientific instruments and repositories. Automating the execution of computational applications is necessary for improving efficiency, robustness, and scientific productivity. Scientific workflows have served as a useful abstraction for conducting these computational experiments in several domains such as astronomy, physics, climate science, earthquake science, and biology [1]. Large-scale workflows typically comprise thousands of compute and/or data-intensive tasks, with intricate data, and control dependencies and process vast amounts of data (from remote sensors, instruments, etc.) to conduct complex modeling, simulations, and data analysis tasks. Scientific workflow management systems, such as Pegasus [2], are critical automation components that enable efficient and resilient workflow execution across distributed and heterogeneous infrastructures.

Science workflows are increasingly being executed by using distributed, federated, heterogeneous cyberinfrastructure resources that span multiple locations (e.g., scientific data collection from instruments, computing and storage resources including supercomputers, high-performance computing clusters, cloud computing systems, and visualization facilities) connected by high-performance networks [3], [4]. The distributed infrastructures are complex because of their inherent characteristics: resource heterogeneity, deployment of complex and heterogeneous system software stacks, and distributed control, stemming from their management by different organizations, domains, and communities. Hence, the operators of these infrastructures and the scientists who use them have limited visibility and thus limited understanding of the entire set of resources used by the science workflows and their behavior. This limited visibility makes it extremely difficult to detect and diagnose anomalies in the infrastructure (e.g., network congestion, system I/O bottlenecks, file system overload) and to understand how they propagate all the way up to the scientists' workflows, resulting in performance degradation and faults in workflow execution.

Many existing anomaly detection methods [5]–[9] are not sufficient to correlate data coming from disparate sources of infrastructure with each other and with information captured at the application level. Also, many prevalent workflow anomaly detection methods are based on thresholds or fixed rule-based procedures [5], [10], which fail to understand longitudinal patterns, miss opportunities for anomaly detection, and can seldom be used for identifying the root cause of the anomalies [5].

Given the scale, complexity, and limited visibility of distributed infrastructures and the need to synthesize disparate data from multidomain infrastructure resources, machine learning (ML) and deep learning (DL) approaches have recently gained importance for detecting and diagnosing anomalies that occur when executing complex workflows on distributed infrastructures. In [7], DL techniques have been used to develop a mechanism that forecasts anomalous behaviors of dynamic jobs in high energy physics experiments by gathering minimal data as early as possible in the job's life cycle. Improvements of up to 14% in resource utilization are

reported, but only application-specific metrics are considered (e.g., CPU/GPU load and memory and disk usage). In [11], Wang et al. used k-means clustering and decision trees to detect workflow and task-level anomalies, considering only application-level metrics. In [12], Gaikwad et al. used autoregression analysis for time-series data on specific application metrics in scientific workflows to detect I/O bottlenecks. While these prior successes are promising, they do not address, in a holistic way, the challenges associated with anomaly detection for complex science workflows.

A recent related work [13] suggested capturing the workflow job features in Gantt charts and used convolutional neural network (CNN) classifiers for anomaly detection. However, it missed the inherent topological information in the workflows.

In this paper we present a novel graph neural network (GNN)-driven approach that holistically models key issues in anomaly detection within complex science workflows. In particular, we first describe the science workflows as a directed acyclic graph (DAG) to model the interdependencies (including intricate data and control dependencies) between different jobs and capture key workflow performance features from applications and the infrastructure. Modeling and training the problem using GNN approaches allow us to capture these dependencies and predict whether a workflow (graph) or a job (node) is anomalous or not. More interestingly, since there are no restrictions on the input of these graphs, such as number of jobs in workflows, our GNN approach can take different workflows for training at the same time, making our method generalizable to a wide range of workflows, in contrast with existing works in the literature.

The main contributions of the paper are as follows.

- We adapt a simple and efficient GNN-based approach to learn the node embedding from both the workflow job features and local dependencies.
- We explore and evaluate the workflow anomalies from both the graph level (workflows) and the node level (jobs).
- We build an anomaly detection model from different types of workflows simultaneously.
- We determine the effectiveness and efficiency of the GNN model by comparing it with conventional machine learning models and previous CNN approaches.

II. Workflow Model

Traditional workflow management systems represent workflows as DAGs, in which jobs are represented as nodes and dependencies between jobs are represented as edges. A job can start its execution if and only if all its predecessors (parent jobs) have successfully finished their jobs. More formally, a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \cdots, v_n\}$ represents the set of n jobs and $\mathcal{E} \subseteq \mathcal{V}^2$ represents data dependencies between jobs. If $e_{ij} = (v_i, v_j) \in \mathcal{E}$, job v_i must complete its execution before v_j can start. We define $succ(v_i) = \{v_k \,|\, (v_i, v_k) \in \mathcal{E}\}$ (resp. $pred(v_i) = \{v_k \,|\, (v_k, v_i) \in \mathcal{E}\}$) to be the successors (resp. predecessors) of job $v_i \in \mathcal{V}$.

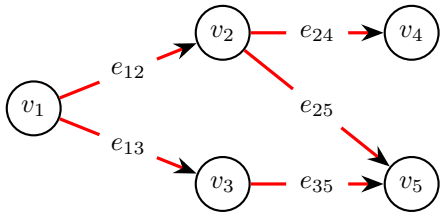


Fig. 1. Example of a DAG.

In this work we use Pegasus [2], an exemplar workflow management system, which enables users to design workflows at a high level of abstraction that is independent of the resources available to execute them and the location of data and executables. Pegasus transforms these abstract workflows into executable workflows, by concertizing the job specifications and by introducing data transfer and auxiliary jobs to establish a successful execution environment on the compute resources. Pegasus workflows have been deployed on distributed and high-performance computing resources (e.g., NERSC [14] and OLCF [15]), shared computing resources (e.g., XSEDE [16] and OSG [17]), local clusters, and clouds.

In Pegasus, edges (e_{ij}) represent sequential dependencies and data dependencies, while jobs (v_i) can be classified into three main categories:

- **Compute:** a job that describes a computational task
- **Transfer:** a job that moves data to/from an execution site
- **Auxiliary:** a job that creates working directories or cleans up unused data.

Additionally, Pegasus collects provenance data and events during the execution of a workflow, and it supports end-to-end workflow execution monitoring. Pegasus via its Panorama [18], [19] branch can collect execution traces of the computational tasks, statistics for individual transfers and infrastructure-related metrics, which get stored into an Elasticsearch instance [20]. These metrics can be associated with the jobs (nodes) of a workflow (DAG) after execution and describe the performance of the job. Currently, edges are not overloaded with any additional features.

In this paper we describe the jobs using the node features of Listing 1, which is a subset of what Pegasus monitoring can offer. For each node $v_j \in \mathcal{V}$, we have the following:

- **Setup features** that describe the executable, the arguments used, the execution site, and the user that triggered the workflow
- **Timing features** that describe the start and the end of the different phases of a job
- **Execution features** that describe the hostname the job assigned, the resources used, the amount of data generated, and the exit code
- **Composite features** that combine timing features to generate relative times.

The meaning of each composite field and how they are calculated is described below:

- **Ready time:** timestamp since the beginning of the workflow, where all dependencies have been met and the

```
"type": <enum(compute, auxiliary, transfer)>,
"is_clustered": <bool>,
"submit_ts": <epoch>,
"pre_script_start_ts": <epoch>,
"pre_script_end_ts": <epoch>,
"stage_in_start_ts": <epoch>,
"stage_in_end_ts": <epoch>,
"execute_start_ts": <epoch>,
"execute_end_ts": <epoch>,
"stage_out_start_ts": <epoch>,
"stage_out_end_ts": <epoch>,
"post_script_start_ts": <epoch>,
"post_script_end_ts": <epoch>,
"transformation": <string>,
"executable": <string>,
"arguments": <string>,
"user": <string>,
"hostname": <string>,
"execution_site": <string>,
"num_of_bytes_staged_in": <int>,
"num_of_bytes_staged_out": <int>,
"cpu_time": <float>,
"exitcode": <int>,
#### composite fields ####
"ready_ts": <epoch>,
"wms_delay": <int>,
"queue_delay": <int>,
"pre_script_delay": <int>,
"runtime": <int>,
"post_script_delay": <int>,
"stage_in_delay": <int>,
"stage_out_delay": <int>
```

Listing 1: Features describing a job.

job can be dispatched. This timestamp is equal to the timestamp the last parent dependency was met.

- **Prescript delay:** time spent on a script that is executed before job submission, if it exists (pre_script_stop_ts - pre_script_start_ts).
- **WMS delay:** time spent by the workflow management system to prepare and submit the job (submit_ts - ready_ts).
- **Queue delay:** time spent in the queue waiting for resources (execute_start_ts - submit_ts).
- **Stage-in delay:** time spent transferring input data (start_in_end_ts - start_in_start_ts).
- **Runtime:** time spent during computation (execute_end_ts - execute_start_ts).
- **Stage-out delay:** time spent transferring data to the intermediate scratch directory or final output directory (start_out_end_ts - start_out_start_ts).
- **Post-script delay:** time spent on a script executed after a job indent exits (e.g., WMS parses stdout and exits code) (post_script_stop_ts - post_script_start_ts).

These features capture the end-to-end performance of the workflows, including application and infrastructure metrics, and are used to build the GNN models (Section III).

III. GNN Model for Anomaly Detection

Given a workflow, our goal is to determine whether an anomaly is associated with this workflow or not. These anomalies could occur due to CPU or hard disk abnormalities and could be present in any specific job (more details regarding these anomalies are described in Section IV). To detect these anomalies, we developed a graph neural network model that takes the workflows (modeled as DAGs with features) as input and detects the existence of anomalies in a supervised fashion.

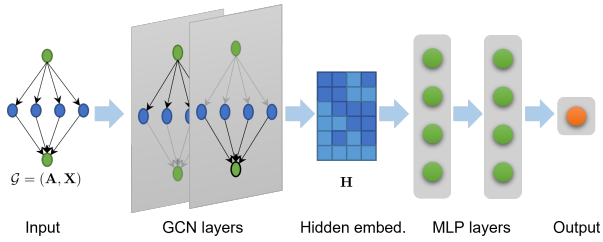


Fig. 2. Graph neural network architecture.

Specifically, we designed a simple graph neural network based on graph convolutional layers (GCNs, [21]). Our GNN, visualized in Figure 2, consists of two modules: the GCN module that provides the hidden layer embedding and the multilayer perceptron (MLP) module that detects the anomaly. We use the GNN for detecting anomalies job-wise (by formulating a node classification problem) and workflow-wise (by formulating a graph classification problem). The GCN module comprises two layers where each layer consists of a GCN operation followed by a rectified linear unit (ReLU) activation function. The MLP module is then applied to the hidden embedding provided by the GCN module to calculate the probability of a workflow/job (graph/node) being anomalous or not. For the workflow anomaly detection scenario, a global average pooling (mean of the hidden embedding across all jobs) is applied prior to the MLP module. The goal of the mean pooling is to integrate out the variation of information across jobs. On the other hand, in the job anomaly detection scenario, the MLP module is applied to each node, and we evaluate whether each job is anomalous or not.

Specifically, given a graph representation of a workflow $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, where $\mathbf{X} \in \mathbb{R}^{n \times m}$ is a matrix representing the node features, and $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a matrix representing the directed adjacency matrix, i.e., $\mathbf{A}_{ij} = 1$ node v_i to node v_j, 0 otherwise. We define the anomaly detection module as

$$\mathbf{z} = \mathsf{MLP}(f(\mathsf{GCN}(\mathbf{A}, \mathbf{X}))). \tag{1}$$

The total number of jobs in each workflow is represented by a variable n and the number of anomalies is represented by p. Therefore, $\mathbf{z} \in \mathbb{R}^{n \times p}$ for job anomaly detection and $\mathbf{z} \in \mathbb{R}^{1 \times p}$ for workflow anomaly detection. For workflow anomaly detection, the function f is the global average pooling (GAP defined as

$$\mathsf{GAP}(\mathbf{H}) = \frac{1}{N} \sum_{k=1}^{N} \mathbf{H}_k, \tag{2}$$

where $\mathbf{H}$ is the output of the GCN module. In the case of job anomaly detection, where no average pooling is done, the output of the GCN module provides the hidden representation corresponding to each job.

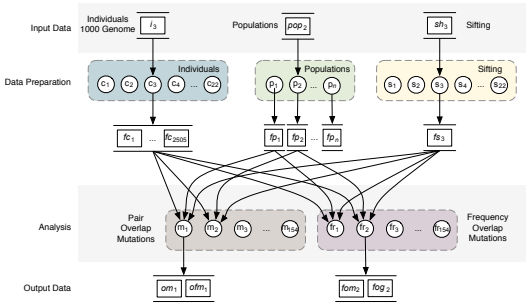


Fig. 3. The 1000Genome Pegasus workflow.

The GCN module learns the hidden node embedding $\mathbf{H}$ by aggregating the information from its neighbors. Intuitively, the GCN layer captures the dependencies between a job and its neighbor jobs (successors). This operation is performed by concatenated layer-wise propagations mathematically represented by the $(l+1)$th hidden embedding as

$$\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{H}^{(l)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}), \qquad (3)$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix with inserted self-loops and $\hat{\mathbf{D}}$ is the diagonal degree matrix of $\hat{\mathbf{A}}$. $\mathbf{W}^{(l)}, \mathbf{b}^{(l)}$ are the learnable parameters and $\sigma(\cdot)$ representing the ReLu activation functions. And for the first layer, where $l = 0$, $\mathbf{H}^0 = \mathbf{X}$ as the input of node embedding.

IV. Experimental Setup

A. Representative Workflows

To evaluate our GNN approach, we use the following representative science workflows.

1000Genome workflow: This Pegasus workflow (Figure 3) is based on the 1000 Genomes project, which provides a reference for human variation, having reconstructed the genomes of 2,504 individuals across 26 different populations [22]. The workflow identifies mutational overlaps for statistical evaluation of potential disease-related mutations [23]. The workflow is composed of five tasks: (1) *individuals* – fetches and parses the Phase 3 data from the 1000 Genomes project per chromosome; (2) *populations* – fetches and parses five super populations and a set of all individuals; (3) *sifting* – computes the SIFT scores of all of the single nucleotide polymorphisms variants, as computed by the Variant Effect Predictor; (4) *pair overlap mutations* – measures the overlap in mutations among pairs of individuals; and (5) *frequency overlap mutations* – calculates the frequency of overlapping mutations across subsamples of certain individuals.

CASA Nowcast workflow: Nowcasts [24] are short-term advection forecasts generated by mosaicing asynchronous individual CASA [25] radar reflectivity data, accumulating the composite grids over a short duration, and projecting into the future by estimating the derivatives of motion and intensity with respect to time. Every minute the CASA Nowcasting system generates 31 grids of predicted reflectivity, one for each minute into the future 0–30 minutes. The Nowcast workflow creates raster images for all grids every minute and then contours for multiple reflectivity levels on each of these grids. The Pegasus Nowcast workflow (Figure 4) contains 63 compute tasks, with one task to split the input data into individual grids and then 62 independent tasks to compute the reflectivity and respective contour images.

CASA Wind workflow: This workflow [26] identifies areas of maximum observed wind magnitudes using data from a network of seven overlapping Doppler weather radars from the CASA system. This workflow periodically takes all available scans and creates a new file in a World Geodetic System 1984 latitude/longitude projection representing the highest winds that have been observed in the time period. The remainder of the workflow computes geographic overlays of the wind contours and communicates the risks to the relevant users of the system. The Pegasus Wind workflow (Figure 5) has a data preparation stage that unzips the input data, which is followed by four compute tasks that output the wind products and notify points of interest for severe weather.

B. Task Clustering

Pegasus enables clustering of workflow tasks into larger jobs via horizontal or label task clustering, which changes the number of jobs in the final executable workflow. With horizontal clustering, tasks on the same level are grouped together; with

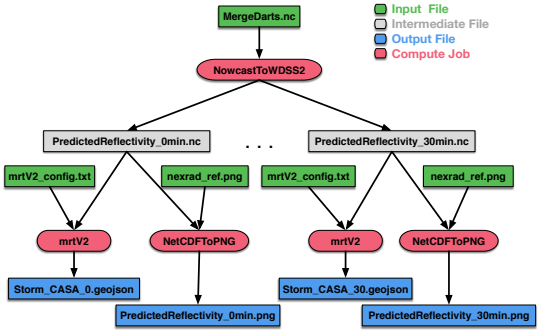


Fig. 4. CASA Nowcast Pegasus workflow.

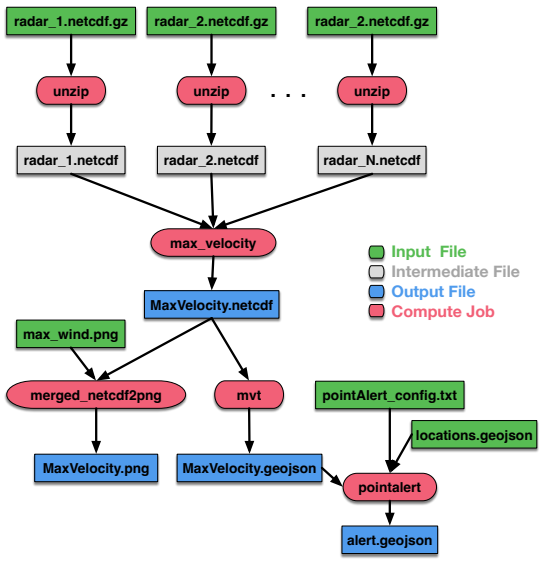


Fig. 5. CASA Wind Pegasus workflow.

TABLE I
STATISTICS OF WORKFLOWS

Workflow	DAG		Normal	CPU		HDD						Packet Loss				
	Nodes	Edges		2	3	50	60	70	80	90	100	0.1%	0.5%	1.0%	3.0%	5.0%
1000 Genome	57	129	200	125	125	150	75	75	75	75	125	50	50	50	50	50
CASA Nowcast w/ Clustering 8	13	20	270	150	150	60	60	60	60	60	60	60	60	60	60	60
CASA Nowcast w/ Clustering 16	9	12	270	150	150	60	60	60	60	60	60	60	60	60	60	60
CASA Wind w/ Clustering	7	8	270	150	150	60	60	60	60	60	60	60	60	60	60	60
CASA Wind w/o Clustering	26	44	270	150	150	60	60	60	60	60	60	60	60	60	60	60

label clustering, Pegasus groups together tasks that carry the same label in their metadata. In our experiments we enable horizontal clustering for the CASA Nowcast workflow and label clustering for the CASA Wind workflow.

- **CASA Nowcast with Clustering 8**: Horizontal clustering with max cluster size 8.
- **CASA Nowcast with Clustering 16**: Horizontal clustering with max cluster size 16.
- **CASA Wind with Clustering**: Label clustering, where all the compute tasks except the unzip tasks are grouped together.

C. Data Collection

To execute the three workflows, we provisioned resources from the ExoGENI [27] testbed, including 7 virtual machines (*1 submit node*, *5 worker nodes*, and *1 data node*). The worker nodes are located within the same ExoGENI region, while the submit node and the data node are located in a different region. Each virtual machine has 4 2.2 GHz vCPUs, 10 GB RAM, and 75 GB storage. The connectivity between the two ExoGENI regions was established over a high-speed layer 2 VLAN. To facilitate the workflow execution, we configured our nodes with Pegasus and HTCondor, and on the data node we install a web server to serve files over HTTP.

1) Injecting Anomalies: To introduce synthetic network and I/O anomalies, we used the Linux Traffic Control (TC) toolset [28]. TC is able to replicate network anomalies such as delay, packet loss, and jitter, by configuring the Linux kernel packet scheduler. Additionally, to reduced the performance of the worker nodes, we use the stress tool [29], a simple workload generator that can impose a configurable amount of CPU, memory, I/O, and disk stress on the system.

2) Workflow Data Collection: To collect the data, we used the Pegasus Panorama branch [18], which offers advanced monitoring capabilities [19]. It enables end-to-end online workflow execution monitoring and provides execution traces of the computational tasks, statistics for individual transfers, and infrastructure-related metrics, which get stored in an Elasticsearch instance [20].

During our data collection, we generated 6,000 traces of the above workflows for 4 main classes, as seen in Table I. Examples of how these anomalies affect the workflow execution can be found in our prior work [19].

- **Normal.** No anomaly is introduced – normal conditions.
- **CPU.** 2–3 cores are occupied by the stress tool on each worker node.
- **HDD.** 50–100 MB of data are continuously written by the stress tool on each worker node.
- **Packet loss.** The network connection between two ExoGENI regions is experiencing 0.1%–5.0% of packet loss.

V. EVALUATION OF THE GNN MODEL FOR ANOMALY DETECTION

A. Model Setup and Metrics

For an exhaustive evaluation of our present approach, we developed 24 GNN models. For each of the five workflows, first we developed workflow-specific GNN models for binary classification where ($p = 2$) and the multilabel classification case ($p = 4$). In addition to these 20 models, we consider a case "ALL" where all the workflows are utilized together to train the GNN. The "ALL" experiment is unique in our work since it allows a single GNN model to predict anomalies across workflows even when these workflows have an unequal number of jobs. We realize the "ALL" model for both the binary, multilabel job anomaly and the workflow anomaly cases.

We split the data into training, validation, and testing at 60%, 20%, and 20%, respectively. All the layers within the GNN architecture take the hidden dimension of 64 with bias terms and apply the cross-entropy loss to quantify the wellness of the trained model. For the optimization, we employ the Adam optimizer with a learning rate $1e^{-3}$. As the binary classification of normal/anomaly are imbalanced, we report not only the accuracy on the test dataset but also the F1-score, precision score, and recall, which are widely used metrics for imbalanced data.

B. Graph-Level Anomaly Detection

To diagnose anomalies of the workflows, we fed the graph through the GCN module, then through the MLP module, and

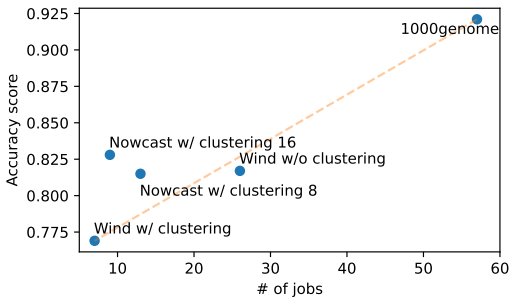


Fig. 6. Rationale behind accuracy vs. number of jobs in the workflow.

TABLE II
GRAPH-LEVEL CLASSIFICATION

Workflow	Binary				Multilabel
	Accuracy	F1	Recall	Precision	Accuracy
1000 Genome	0.921 ± 0.019	0.955 ± 0.011	0.955 ± 0.017	0.954 ± 0.006	0.852 ± 0.010
Nowcast w/ clustering 8	0.815 ± 0.017	0.873 ± 0.013	0.817 ± 0.024	0.937 ± 0.003	0.683 ± 0.008
Nowcast w/ clustering 16	0.828 ± 0.010	0.901 ± 0.005	0.982 ± 0.004	0.832 ± 0.004	0.593 ± 0.011
Wind w/ clustering casa	0.769 ± 0.003	0.860 ± 0.002	0.911 ± 0.003	0.815 ± 0.002	0.434 ± 0.010
Wind w/o clustering casa	0.817 ± 0.003	0.893 ± 0.002	0.971 ± 0.005	0.826 ± 0.002	0.586 ± 0.015
ALL	0.841 ± 0.006	0.907 ± 0.004	0.951 ± 0.010	0.867 ± 0.005	0.674 ± 0.008

TABLE III
NODE-LEVEL CLASSIFICATION

Workflow	Binary				Multilabel
	Accuracy	F1	Recall	Precision	Accuracy
1000 Genome	0.870 ± 0.001	0.927 ± 0.001	0.959 ± 0.006	0.896 ± 0.004	0.743 ± 0.006
Nowcast w/ clustering 8	0.798 ± 0.002	0.883 ± 0.001	0.961 ± 0.003	0.816 ± 0.002	0.590 ± 0.003
Nowcast w/ clustering 16	0.801 ± 0.003	0.876 ± 0.001	0.947 ± 0.007	0.815 ± 0.004	0.487 ± 0.009
Wind w/ clustering casa	0.775 ± 0.001	0.868 ± 0.001	0.991 ± 0.002	0.772 ± 0.001	0.389 ± 0.003
Wind w/o clustering casa	0.789 ± 0.001	0.879 ± 0.000	0.977 ± 0.002	0.799 ± 0.001	0.472 ± 0.002
ALL	0.829 ± 0.001	0.903 ± 0.000	0.980 ± 0.002	0.837 ± 0.002	0.594 ± 0.002

TABLE IV
GRAPH-LEVEL BINARY CLASSIFICATION BY ANOMALY CATEGORIES.

Workflow	CPU	HDD	Loss	Joint Anomalies
1000 Genome	1.000	0.981	0.722	0.921
Nowcast w/ clustering 8	0.763	0.968	0.719	0.815
Nowcast w/ clustering 16	0.745	0.825	0.790	0.828
Wind w/ clustering casa	0.544	0.779	0.693	0.769
Wind w/o clustering casa	0.784	0.967	0.784	0.817

TABLE V
GRAPH-LEVEL BINARY CLASSIFICATION BY ANOMALY LEVELS

Workflow	CPU 2	CPU 3	HDD 60	HDD 100	Loss 0.5%	Loss 5%
1000 Genome	0.985	1.000	0.927	0.985	0.760	1.000
Nowcast w/ clust. 8	0.857	0.902	0.985	0.985	0.788	0.970
Nowcast w/ clust. 16	0.750	0.786	0.818	0.894	0.833	0.879
Wind w/ clustering	0.679	0.714	0.773	0.849	0.773	1.000
Wind w/o clustering	0.683	0.951	0.891	0.938	0.797	1.000

evaluated probabilistic predictions for each graph. Table II reports the graph classification performance on the testing set in both binary and multilabel settings. In addition, to generalize our approach to a wide range of different workflows, we evaluated the performance of a single model to predict the anomaly in a set of workflows simultaneously, denoted "ALL" in the last row. Figures 7 (a, binary) and (b, multilabel) present the accuracies on training and validation of a single model for "ALL" workflows. We can see that the binary classification performs better than the multilabel anomaly detection, demonstrating better detection for normal versus anomaly than detecting a specific anomaly category, and reaching around 10% to 30% higher accuracies across different workflows.

Moreover, we investigated the rationale of the performance from the perspective of workflow structures. Figure 6 shows the relationship between the accuracy of the binary setting and the number of jobs in the workflows. Clearly, the accuracy is proportional to the workflow size; that is, with more complex structures, the anomaly detection reaches higher accuracy. This is largely due to the intrinsic property of graph neural networks, where more complex structures help aggregate from local neighbors through the layer-wise propagation, overcoming the oversmoothing problems [30].

C. Node-Level Anomaly Detection

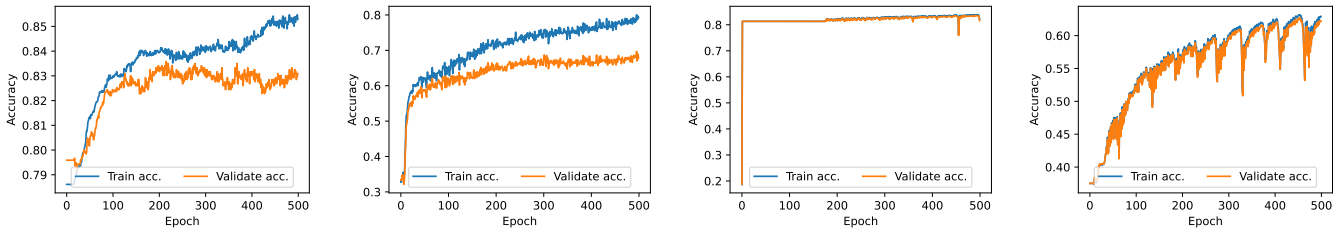
Detecting the anomaly jobs (nodes) within a workflow (graph) is another fundamental problem in science workflows. Unlike the graph-level approach with a single label for each run, we assign the node labels by adapting the label from the entire run. That is, the labels of the jobs are the same as the label of a single run. The training, validation, and testing sets are randomly sampled jobs across all the runs. Therefore, it is sufficient to train the model with information from different labels.

Similar to the graph-level setting, we report the performance on the testing set in Table III. "ALL" in the last row represents the performance of a single model training different workflows at the same time. We observe that the binary (normal versus anomaly) setting reaches better accuracy than the multilabel setting does, the same observation we had in graph-level anomaly detection. More specifically, the recall score, which is a widely used metric for imbalanced data for binary problems, reaches 0.98 even in the case of a single model utilizing all workflows together. Figures 7 (c, binary) and (d, multilabel) show the accuracies on training and validation of a single model for "ALL" workflows in terms of node-level anomalies.

D. Anomaly Detection by Category

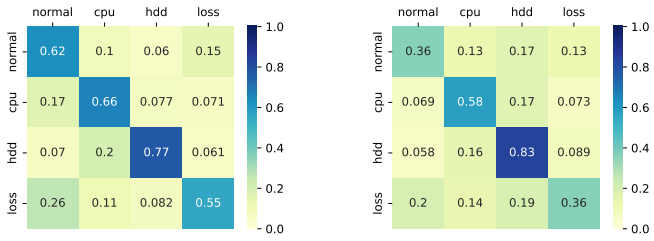
For both the graph-level and node-level anomaly detection problems, the binary label setting outperforms the multilabel settings in terms of the test accuracy. We investigated the rationale behind this performance and explored anomaly detection by their categories. As described in Section IV, there are three categories of anomalies: CPU-related anomaly by stressing on each worker node, HDD-related anomaly by continuously writing on each worker node, and network packet-loss anomaly by enforcing packet loss between regions.

Table IV shows the average accuracy comparison based on different anomaly categories for the graph-level problem,



(a) Graph level, binary (b) Graph level, multilabel (c) Node level, binary (d) Node level, multilabel

Fig. 7. Training of a single model for all workflows.



(a) Graph-level (b) Node-level

Fig. 8. Confusion matrix of multilabel classification.

where the last column represents the mixed anomalies together as a binary classification. Among the three categories, the HDD-related anomalies are easiest to detect in general, namely, with higher accuracy than the other two.

Figures 8a and 8b show the confusion matrix, also known as error matrix, to evaluate the multilabel classification for the single-model graph-level and node-level classifications, respectively. The value in each cell indicates the number of observations known to be in group i and predicted to be in group j, the higher the better. These results also match our findings, given in Table IV: the HDD-related anomalies reach a higher score than the others do (on the diagonal).

E. Anomaly Detection for Different Anomaly Levels

We synthetically generated the anomalies by injecting different levels of anomalies for CPU, HDD, and packet loss. We show the performance of anomaly detection by levels in Table V for the graph-level anomaly detection problem. CPU 2 and 3 indicate the number of cores occupied by stress tool, HDD 60 and 100 indicate the number of megabytes continuously written by the stress tool on each worker node, and Loss 0.5% and 5% indicate the percentage of packet loss from the network connection. The results meet our expectation that with higher anomaly levels, the model reaches better accuracy in classification, as data corresponding to anomalies are pushed far away from the distribution of normal runs in the high-dimensional space.

F. Model Comparison

To demonstrate the advantages of taking graph neural networks for DAG modeling, we compared with standard machine learning models including the following: (1) support vector machines (SVMs) with radial basis function kernels

TABLE VI
PERFORMANCE COMPARISON ON 1000GENOME

Model	Acc.	Recall	Prec.	F1
SVM	0.622	0.622	0.667	0.550
MLP	0.874	0.874	0.875	0.874
RF	0.898	0.898	0.908	0.887
AlexNet	0.910	0.914	0.910	0.910
VGG-16	0.900	0.900	0.900	0.900
ResNet-18	0.910	0.916	0.910	0.910
Our GNN	**0.923**	**0.929**	**0.921**	**0.922**

TABLE VII
RUNTIME OF TRAINING DEEP LEARNING MODELS ON 1000GENOMES

Model	AlexNet	VGG-16	ResNet-18	Our GNN
Runtime (sec.)	251	435	991	**142**

and $C = 1$, (2) multilayer perceptron with hidden layers $(128, 128, 128)$, and (3) random forest (RF) with maximum depth set to 3. In previous work mentioned in Section I, Krawczuk et al. [13] took a computer-vision-inspired deep learning approach by generating Gantt charts from node features. To keep the consistency of the data split, we took the data split as 60%, 20%, and 20% for training, validation, and testing, respectively, and compared the performance of our proposed GNN with the computer vision approach in Table VI on the 1000Genome workflow, including the pretrained models AlexNet, VGG-16, and ResNet-18 with data augmentation (rotations, flips, shifts, and augmented noises).

With the additional structural information from the workflow, the GNNs outperforms the standard machine learning models and Gantt charts representation of workflows, because of their capacity to learn the embedding from local structural information and node features. We note that all the other hyperparameters are the same as in Section V-A, without fine-tuning.

In addition, we measured the training time of the deep learning approaches to check the efficiency of our proposed GNN. The results are presented in Table VII. We implemented the models in PyTorch and trained them with a single NVIDIA A100 40 GB GPU. Reaping the advantages of aggregating local neighbors through matrix multiplication, the GNN approach is much faster than the CNN approaches for training.

VI. CONCLUSION AND FUTURE WORK

In this work we considered workflow anomaly detection by modeling the workflows as directed acyclic graphs and using graph neural networks to detect anomalous workflows.

By learning the node features and the local structural information, the GNN approach surpasses the conventional machine learning approaches and computer vision approaches.

Next, we will investigate analysing larger scientific workflows in terms of efficiency, and generalizing the models from one learned workflow to another. Also, since generating the systematic anomalies is time-consuming, we aim to take advantage of generative models to create synthetic noise representing the anomaly scenarios.

VII. Acknowledgments

This work is funded by the Department of Energy under the Integrated Computational and Data Infrastructure (ICDI) for Scientific Discovery, grant #DE-SC0022328. Experimental data was collected on the ExoGENI testbed supported by NSF. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

References

[1] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: Scientific Workflows for Grids*. Springer Publishing Company, Incorporated, 2014.

[2] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[3] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter, "The future of scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, pp. 159–175, 2018.

[4] K. D. Moreland, "The future of scientific workflows. report of the doe ngns/cs scientific workflows workshop (sandia contributions)," Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2015.

[5] E. Deelman, A. Mandal, M. Jiang, and R. Sakellariou, "The role of machine learning in scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 33, no. 6, pp. 1128–1139, 2019.

[6] M. A. Rodriguez, R. Kotagiri, and R. Buyya, "Detecting performance anomalies in scientific workflows using hierarchical temporal memory," *Future Generation Computer Systems*, vol. 88, pp. 624–635, 2018.

[7] F. Li and F. Song, "Building a scientific workflow framework to enable real-time machine learning and visualization," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 16, p. e4703, 2019.

[8] A. Singh, I. Altintas, M. Schram, and N. Tallent, "Deep learning for enhancing fault tolerant capabilities of scientific workflows," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3905–3914.

[9] G. Papadimitriou, M. Kiran, C. Wang, A. Mandal, and E. Deelman, "Training classifiers to identify TCP signatures in scientific workflows," in *2019 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*. Los Alamitos, CA, USA: IEEE Computer Society, nov 2019, pp. 61–68. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/INDIS49552.2019.00012

[10] R. Stevens, V. Taylor, J. Nichols, A. B. Maccabe, K. Yelick, and D. Brown, "AI for Science," Argonne National Lab.(ANL), Argonne, IL (United States), Tech. Rep., 2020.

[11] C. Wang, G. Papadimitriou, M. Kiran, A. Mandal, and E. Deelman, "Identifying execution anomalies for data intensive workflows using lightweight ML techniques," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2020, pp. 1–7.

[12] P. Gaikwad, A. Mandal, P. Ruth, G. Juve, D. Król, and E. Deelman, "Anomaly detection for scientific workflow applications on networked clouds," in *2016 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2016, pp. 645–652.

[13] P. Krawczuk, G. Papadimitriou, S. Nagarkar, M. Kiran, A. Mandal, and E. Deelman, "Anomaly detection in scientific workflows using end-to-end execution Gantt charts and convolutional neural networks," in *Practice and Experience in Advanced Research Computing*, 2021, pp. 1–5.

[14] "National Energy Research Scientific Computing Center (NERSC)," https://www.nersc.gov.

[15] "Oak Ridge Leadership Computing Facility (OLCF)," https://www.olcf.ornl.gov.

[16] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. Scott, and N. Wilkins-Diehr, "XSEDE: Accelerating scientific discovery," *Computing in Science & Engineering*, vol. 16, no. 05, pp. 62–74, sep 2014.

[17] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick, "The open science grid," *Journal of Physics: Conference Series*, vol. 78, p. 012057, jul 2007. [Online]. Available: https://doi.org/10.1088%2F1742-6596%2F78%2F1%2F012057

[18] SciTech, "Pegasus panorama," https://github.com/pegasus-isi/pegasus/tree/panorama.

[19] G. Papadimitriou, C. Wang, K. Vahi, R. Ferreira da Silva, A. Mandal, L. Zhengchun, R. Mayani, M. Rynge, M. Kiran, V. E. Lynch, R. Kettimuthu, E. Deelman, J. S. Vetter, and I. Foster, "End-to-end online performance data capture and analysis for scientific workflows," *Future Generation Computer Systems*, vol. 117, pp. 387 – 400, 2021. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X20330570

[20] "ELK stack," https://www.elastic.co/elk-stack, 2018.

[21] M. Welling and T. N. Kipf, "Semi-supervised classification with graph convolutional networks," in *J. International Conference on Learning Representations (ICLR 2017)*, 2016.

[22] 1000 Genomes Project Consortium, "A global reference for human genetic variation," *Nature*, vol. 526, no. 7571, pp. 68–74, 2012.

[23] R. Ferreira da Silva, R. Filgueira, E. Deelman, E. Pairo-Castineira, I. M. Overton, and M. Atkinson, "Using simple PID-inspired controllers for online resilient resource management of distributed scientific workflows," *Future Generation Computer Systems*, vol. 95, pp. 615–628, 2019.

[24] E. Ruzanski and V. Chandrasekar, "Weather radar data interpolation using a kernel-based lagrangian nowcasting technique," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 6, pp. 3073–3083, June 2015.

[25] "Collaborative Adaptive Sensing of the Atmosphere," http://www.casa.umass.edu/.

[26] E. Lyons, G. Papadimitriou, C. Wang, K. Thareja, P. Ruth, J. Villalobos, I. Rodero, E. Deelman, M. Zink, and A. Mandal, "Toward a dynamic network-centric distributed cloud platform for scientific workflows: A case study for adaptive weather sensing," in *15th International Conference on eScience (eScience)*, 2019, pp. 67–76.

[27] I. Baldin, J. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills, "ExoGENI: A multi-domain infrastructure-as-a-service testbed," in *The GENI Book*, R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds. Springer International Publishing, 2016, pp. 279–315.

[28] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, vol. 213, 2002.

[29] A. Waterland, "stress, POSIX workload generator," 2013.

[30] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3438–3445.