Research article
First published online May 30, 2023

Graph neural networks for detecting anomalies in scientific workflows

Hongwei Jin jinh@anl.gov, Krishnan Raghavan, [...], George Papadimitriou, Cong Wang, Anirban Mandal, Mariam Kiran, Ewa Deelman, and Prasanna Balaprakash+5-5View all authors and affiliations

Volume 37, Issue 3-4
https://doi.org/10.1177/10943420231172140

Abstract

Identifying and addressing anomalies in complex, distributed systems can be challenging for reliable execution of scientific workflows. We model these workflows as directed acyclic graphs (DAGs), where the nodes and edges of the DAGs represent jobs and their dependencies, respectively. We develop graph neural networks (GNNs) to learn patterns in the DAGs and to detect anomalies at the node (job) and graph (workflow) levels. We investigate workflow-specific GNN models that are trained on a particular workflow and workflow-agnostic GNN models that are trained across the workflows. Our GNN models, which incorporate both individual job features and topological information from the workflow, show improved accuracy and efficiency compared to conventional learning methods for detecting anomalies. While joint trained with multiple scientific workflows, our GNN models reached an accuracy more than 80% for workflow level and 75% for job level anomalies. In addition, we illustrate the importance of hyperparameter tuning method in our study that can significantly improve the metric(s) measure of evaluating the GNN models. Finally, we integrate explainable GNN methods to provide insights on job features in the workflow that cause an anomaly.

Introduction

Distributed computing infrastructures are increasingly being leveraged for the execution of complex scientific applications. These execution platforms are heterogeneous, geographically distributed, and combine compute, network and storage resources from (often) disparate scientific facilities, forming a “superfacility,” to execute large-scale scientific workloads—from data collection on scientific instruments to moving the data, processing, storing, curating, disseminating, and visualizing the results. Being inherently distributed and heterogeneous, these superfacilities do not have centralized control, which results in limited visibility and thus limited understanding of how scientific workloads perform in the entirety of the execution platform. Hence, operators of these infrastructures and scientists who use them struggle to understand the performance of their systems and applications. The problem is exacerbated by anomalies and system performance degradations that

might occur in such complex infrastructures, for example, network congestion, system I/O bottlenecks, and file system overload. Detecting and diagnosing these anomalies in the execution platforms, and how they propagate all the way up to the scientists' workflows, remains a significant challenge.

Not only are the execution platforms complex but also the scientific applications and workloads running on these platforms are getting more sophisticated, with scientists designing and describing their computational campaigns using large-scale scientific workflows. Scientific workflows abound in domain sciences—weather modeling, climate science, seismology, geodesy, biology, astronomy, ecology, high-energy physics, and ocean sciences, just to name a few (Taylor et al., 2014). Workflows provide an appropriate abstraction to describe these campaigns, and workflow management systems like Pegasus (Deelman et al., 2015) automate and orchestrate the execution of these workflows on distributed, heterogeneous infrastructures, carefully managing the computations and data movements on behalf of the scientists, thereby improving the efficiency, robustness, and scientific productivity. Often, large-scale workflows are constituted of thousands of compute and/or data-intensive tasks, with complex data and control dependencies, which perform modeling, simulation, and data analysis by processing vast amounts of data. Given the scale and complexity, the scientific applications or the data movements in these workflows can exhibit anomalous behavior, and this convolution of complex workflows with distributed superfacilities used to execute them presents an even more significant challenge when it comes to anomaly detection and diagnosis.

Recent works in the literature have tackled the anomaly detection problem (Deelman et al., 2019; Rodriguez et al., 2018; Li and Song 2019; Singh et al., 2018; Papadimitriou et al., 2019). However, they have limited capability in correlating data from infrastructure sources, with each other, and with application-level data. Some methods are threshold based or rule based (Deelman et al., 2019; Stevens et al., 2020), which fail to uncover longitudinal patterns. Several machine learning (ML) and deep learning (DL) approaches have been applied more recently to address the anomaly detection problem. Although Li and Song (2019) use DL to forecast anomalies in high-energy physics jobs by leveraging minimal data during early parts of the job's execution and achieved improvements of up to 14% in resource utilization, this work only considers application-level metrics. Similarly, Wang et al. (2020) also used only application-level metrics to apply clustering and decision trees to detect anomalies. Gaikwad et al. (2016) used autoregression techniques to detect I/O bottlenecks. These works, although promising, fail to take a comprehensive approach that holistically addresses correlations and/or leverages workflow-, task-, and infrastructure-level performance data. Krawczuk et al. (2021) utilized convolutional neural network (CNN) classifiers for anomaly detection in Gantt charts that represented the features of the workflow job. But, this work suffered the weakness that it did not use the topological information present in workflows. For specific IO anomaly, the Taufer (2021) discussed machine learning approaches to be used to prevent and mitigate IO contention in High Performance Computing (HPC) systems while dealing with IO bandwidth constraints.

We model the workflows as directed acyclic graphs (DAGs), wherein the nodes and edges represent jobs and their dependencies. We present a new approach that uses

graph neural networks (GNNs) to holistically model the problem of anomaly detection in distributed scientific workflow executions. First, by leveraging the Pegasus workflow management system, we capture key performance metrics for different jobs in the workflow, both from the application tasks and the infrastructure. Using GNN-based approaches allows us to intrinsically capture the control and data dependencies between jobs inherent in the directed acyclic graph (DAG) representation of scientific workflows (Jin et al., 2022). We develop and train the GNN-based models of the workflows by using the data captured in distributed infrastructures, and use them to predict whether a workflow (graph) or a job (node) is anomalous or not. One advantage of our approach over related works in the literature is that our method is generalizable to a wide range of workflows because there are no restrictions on the input to the graphs (e.g., number of jobs in the workflow), and hence, our GNN approach can simultaneously use different workflows for training.

Compared to deep neural network (DNN) models for image and text data, GNNs are less generalizable because of the diversity of the data and domains that they come from. Consequently, it is not easy to use a GNN developed and trained for one domain (such as molecular property prediction) to a different domain (workflow anomaly prediction). Moreover, anomaly detection is relatively a harder problem than the regular classification problem. While it is feasible to develop a GNN relatively in a short time, improving the accuracy of GNN anomaly detection is typically a trial and error and expert-driven method. To that end, a promising approach to improve the performance of GNNs is hyperparameter tuning, where the crucial hyperparameters of the GNNs such as number of layers, learning rate, and batch size are automatically tuned. We leverage an automated hyperparameter tuning approach to tune the hyperparameters of the GNNs that we developed for workflow anomaly detection and demonstrate the importance and effectiveness compared to the baseline.

While GNN models provide the flexibility to model workflow graphs and detect anomalies, they cannot provide explanations about the predictions. The ability to explain anomaly predictions is critical for several reasons: (1) it improves the transparency and interpretability of the model, so that the insights derived from the explanation can be used to increase trust in the GNN model; (2) it can allow researchers and infrastructure operators to gain an understanding of the workflow characteristics, and identify and correct inaccurate predictions made by the models before deployment. Given a trained GNN model and a workflow graph, we seek to derive information about which node and edge features have significantly contributed to an anomaly. To that end, we will focus on explainable GNN methods that seek to provide explanations and attribute node and edge features to anomalies.

To summarize, the article makes the following main contributions:

- We adopt 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 adopt hyperparameter tuning to improve the basic graph neural networks.

• Finally, we provide explainable results from the machine learning model to domain experts.

Workflow model

This work utilizes the Pegasus Workflow Management System (WMS) (Deelman et al., 2015), to describe workflows and manage their execution. Pegasus workflows have been deployed on various types of distributed and high-performance computing resources (e.g., NERSC, 2022 and OLCF, 2022), shared computing resources (e.g., XSEDE (Towns et al., 2014) and OSG (Pordes et al., 2007)), local clusters, and clouds. Pegasus enables the development of workflows abstracted from the compute resources and from the input data, resulting in easily portable workflows. Given the specification of the compute resources and the location of the input data, Pegasus converts the abstract workflows into executable ones, tailored to be suitable for the provided infrastructure. For the rest of this article, we refer to "executable workflows" as "workflows."

In Pegasus, workflows are represented as directed acyclic graphs (DAGs), where nodes represent jobs and edges represent sequential dependencies and data dependencies between the jobs. A job can only be submitted when all its dependencies have been met.

For a more formal definition, a workflow is described as a DAG G=(V,E), where V={v1,...,vn} represents the set of *n* jobs and E⊆V2 represents data dependencies between jobs. If eij=(vi,vj)∈E, job v_i must complete its execution before v_j can start. We define succ(vi)={vk | (vi,vk)∈E} (resp. pred(vi)={vk | (vk,vi)∈E}) to be the successors (resp. predecessors) of the job vi∈V. A job (v_i) of a Pegasus workflow can be classified into one of these three categories:

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

Moreover, Pegasus collects provenance data and events during the execution of a workflow and associates them with the jobs that produced them. Pegasus Panorama extensions (Deelman et al., 2017; Papadimitriou and Deelman 2018; Papadimitriou et al., 2021) allow us to enhance the workflow DAG with features containing execution metrics and infrastructure statistics per job (v_i). To characterize the end-to-end execution of the workflows, the DAG is enhanced with the following types of features:

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

In Listing 1 we present the list of features that we add as metadata to the workflow DAG, which are used to train the GNN models (Section 3).

Graph neural network model for anomaly detection

Our objective is to identify whether a workflow has anomalies, which can be caused by abnormalities in CPU or hard disk performance and can occur in any individual job (as discussed in Section 4). To accomplish this, we designed a graph neural network (GNN) model that uses workflows modeled as directed acyclic graphs (DAGs) with features as input and detects the presence of anomalies. We follow a supervised learning formulation, where we generate training data with normal and anomalous workflow runs and use the trained model to detect anomalies in workflows.
We developed the graph neural network using graph convolutional layers (GCNs, Welling and Kipf (2016)). Our GNN, shown in Figure 1, is composed of two modules: the GCN module, which generates the hidden layer embedding, and the multilayer perceptron (MLP) module, which detects the anomaly. We use the GNN for anomaly detection at both the job level (by framing it as a node classification problem) and the workflow level (by framing it as a graph classification problem). The GCN module is made up of two layers, each containing a GCN operation followed by a rectified linear unit (ReLU) activation function. The MLP module is then applied to the hidden embedding generated by the GCN module to calculate the probability of a workflow/job (graph/node) being anomalous or not. For workflow anomaly detection, a global average pooling (mean of the hidden embedding across all jobs) is applied prior to the MLP module. The aim of the mean pooling is to integrate information across jobs. In the job anomaly detection scenario, the MLP module is applied to each node and we evaluate whether each job is anomalous or not.

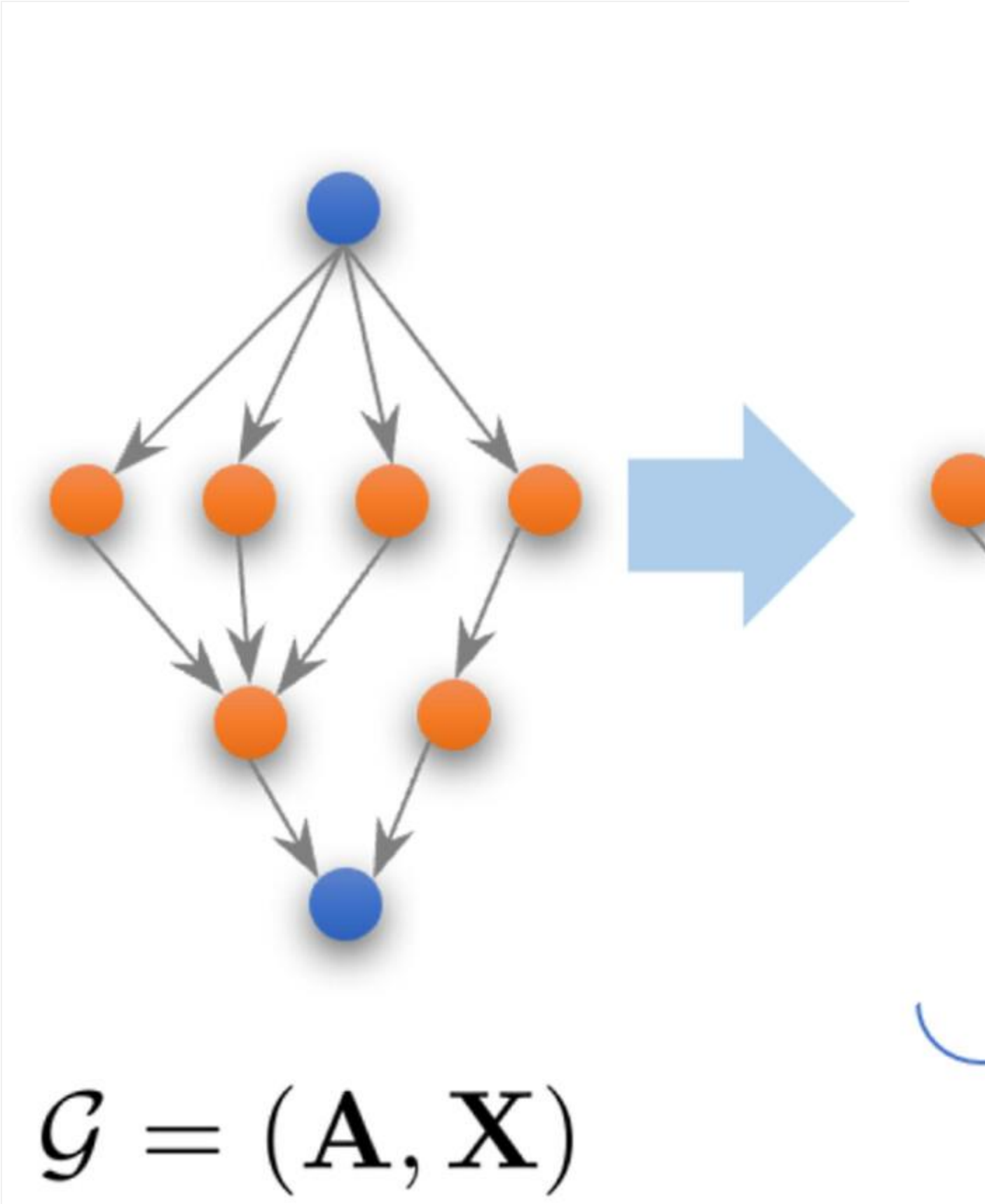

Figure 1. Graph neural network architecture.Open in viewer

```
"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.

In this case, given a graph representation of a workflow G=(A,X), where X∈Rn×m represents the node features and A∈Rn×n represents the directed adjacency matrix, that is, $\mathbf{A}_{ij}$ = 1 if there is an edge from node v_i to node v_j, and 0 otherwise. We define the anomaly detection module as

$$z=\mathrm{MLP}(f(\mathrm{GCN}(A,X))). \quad (1)$$

where the total number of jobs in each workflow is represented by n and the number of anomalies is represented by p. Therefore, z∈Rn×p for job anomaly detection and z∈R1×p for workflow anomaly detection. For the detection of workflow anomalies, the function f is the global average pooling (GAP), defined as

$$\mathrm{GAP}(H)=1N\sum k=1NHk, \quad (2)$$

where **H** is the output of the GCN module. When identifying anomalies at the job level, our GCN module produces a hidden representation for each individual job, without the need for average pooling.

The GCN module extracts the embedding of the hidden node **H** by gathering information from its neighboring nodes. Essentially, the GCN layer detects the relationship between a job and its neighboring jobs (successors). This operation is performed by concatenated layer-wise propagations mathematically represented by the (l + 1)th hidden embedding as

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

where A^=A+I denotes the adjacency matrix with inserted self-loops and D^ is the diagonal degree matrix of A^. The parameters $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$ are learned during the training

process, and $\sigma(\cdot)$ represents the ReLU activation function. For the first layer, denoted as $l = 0$, the embedding of the input node is set to $\mathbf{H}^{(0)} = \mathbf{X}$.

Hyperparameter tuning

GNNs have a number of hyperparameters that need to be set prior to training. Hyperparameter search (HPS) is the process of systematically searching for the best combination of hyperparameters for a GNN model. Typically, this can be done through techniques such as grid search, random search, or Bayesian optimization (Jamieson and Talwalkar 2016), with the aim of achieving the best performance on the validation set. By carefully tuning the hyperparameters, it is possible to significantly improve the performance of GNNs on various tasks including anomaly detection via supervised classification.

In this article, we adopt DeepHyper (Balaprakash et al., 2018), an open-source scalable hyperparameter tuning software. It supports both single- and multi-objective optimization and also supports distributed computing, making it easy to scale up the computation. This is particularly useful in our setting when we have multiple metrics to consider for the imbalanced data, which will be discussed in Section 5.1. The hyperparameter tuning in DeepHyper seeks to find the best hyperparameter values such that the metric measure of the model can be improved. Given a candidate set of hyperparameters Hm, DeepHyper considers the following bilevel optimization problem:

$$hm^*=\operatorname{argmax}_{hm\in Hm} M_{\theta^*_{val}}(m) \quad s.t. \quad \theta^*=\operatorname{argmin}_{\theta} L_{hm_{train}}(X,y;\theta), \tag{4}$$

where M is the metric, for example, the accuracy score, evaluated on the validation set, and L is the loss function used in the GNN model. Such optimization finds the best parameters of the GNN model (θ^*) from the training set by minimizing the loss function, and finds the best hyperparameters for the model (hm^*) from the validation set by maximizing the metric measurement.

DeepHyper adopts an asynchronous model-based search (AMBS) method that consists of sampling a number of hyperparameter configurations and progressively fitting a surrogate model over the input–output space until exhausting the user-defined maximum number of evaluations. The asynchronous aspect allows the search to avoid waiting for all the evaluation results before proceeding to the next iteration. As soon as an evaluation is finished, the data are used to retrain the surrogate model, which is then used to bias the search toward the promising configurations. The framework is designed to operate in the master–worker computational paradigm, where one master node fits the surrogate model and generates promising input configurations and worker nodes perform the computationally expensive evaluations and return the outputs to the manager node.

Explainable GNN models

Explanation of anomaly detection predictions from GNNs is more challenging than traditional machine learning models due to their high complexity and black-box nature. One of the key features of GNNs is that they take into account the relationships between nodes in the graph, unlike traditional neural networks that only

process individual examples. This is achieved by applying a propagation step to the graph, which is used to update the embeddings of each node based on its neighbors. GNNExplainer (Ying et al., 2019) is an open-source library for explaining GNN predictions. It aims to provide a unified interface for interpreting the decision-making process of GNN models, and also it provides various techniques for explaining the predictions of GNN models. Specific to the anomaly detection task, we focus on (1) node-level explanations, which allow users to understand how a specific node's features and graph topology contributed to the final prediction; and (2) graph-level explanations, which allow users to understand how the graph structure as a whole contributed to the final prediction. This is indeed useful for identifying and quantifying the importance of features and nodes in the scientific workflow where the anomaly occurs.

For the node-level explanation, given a node v, the explanation tries to identify a subgraph $G_S \subseteq G$ and the associated node features $X_S = \{x_j \mid x_i \in G_S\}$ that are important for the prediction of the trained GNN model. The GNNExplainer takes the mutual information (MI) as the measurement of the importance, formulated as

$$\max_{G_S} MI(Y,(G_S,X_S)) = h(Y) - h(Y \mid G = G_S, X = X_S) \quad (5)$$

where $h(Y)$ is the marginal entropy and $h(Y \mid X) := -\sum_{x \in X, y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)}$ is the conditional entropy of Y given X. Searching among all the subgraphs is impractical. Therefore, by Jensen's inequality, the model eventually optimizes an upper bound

$$\min_{G} h(Y \mid G = E[G_S \sim G], X = X_S), \quad (6)$$

where $G_S \sim G$ is the random graph variable.

For the graph-level explanation, the explainer chooses a reference node v, and takes explanation $G_S(v)$ for reference node and aligns it to explanation of other nodes associated on the graph. More specifically, we extract the 1-hop subgraph from v and explain the nodes extracted from the topological structure, extending the formulation of equation (6) to

$$\min_{G} h(Y \mid G = E[G_S(u) \sim G], X = X_S(u)), \forall u \in N(v) \quad (7)$$

where N(v) is the subset of nodes in the workflow associated v. And $G_S(u) \subseteq G$ and $X_S(u)$ are the associated subgraph and node feature with node u, respectively. Therefore, it allows to aggregate aligned adjacency matrices into a prototype, such that it gives insights into graph patterns shared between nodes connected together.

Experimental setup

Representative workflows

To evaluate our GNN approach, we use the following science workflows orchestrated by Pegasus.

1000 Genome workflow: This workflow (Figure 2) 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 (1000 Genomes Project Consortium 2012). The workflow identifies mutational overlaps for the statistical

evaluation of potential disease-related mutations (Ferreira da Silva et al., 2019). 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.

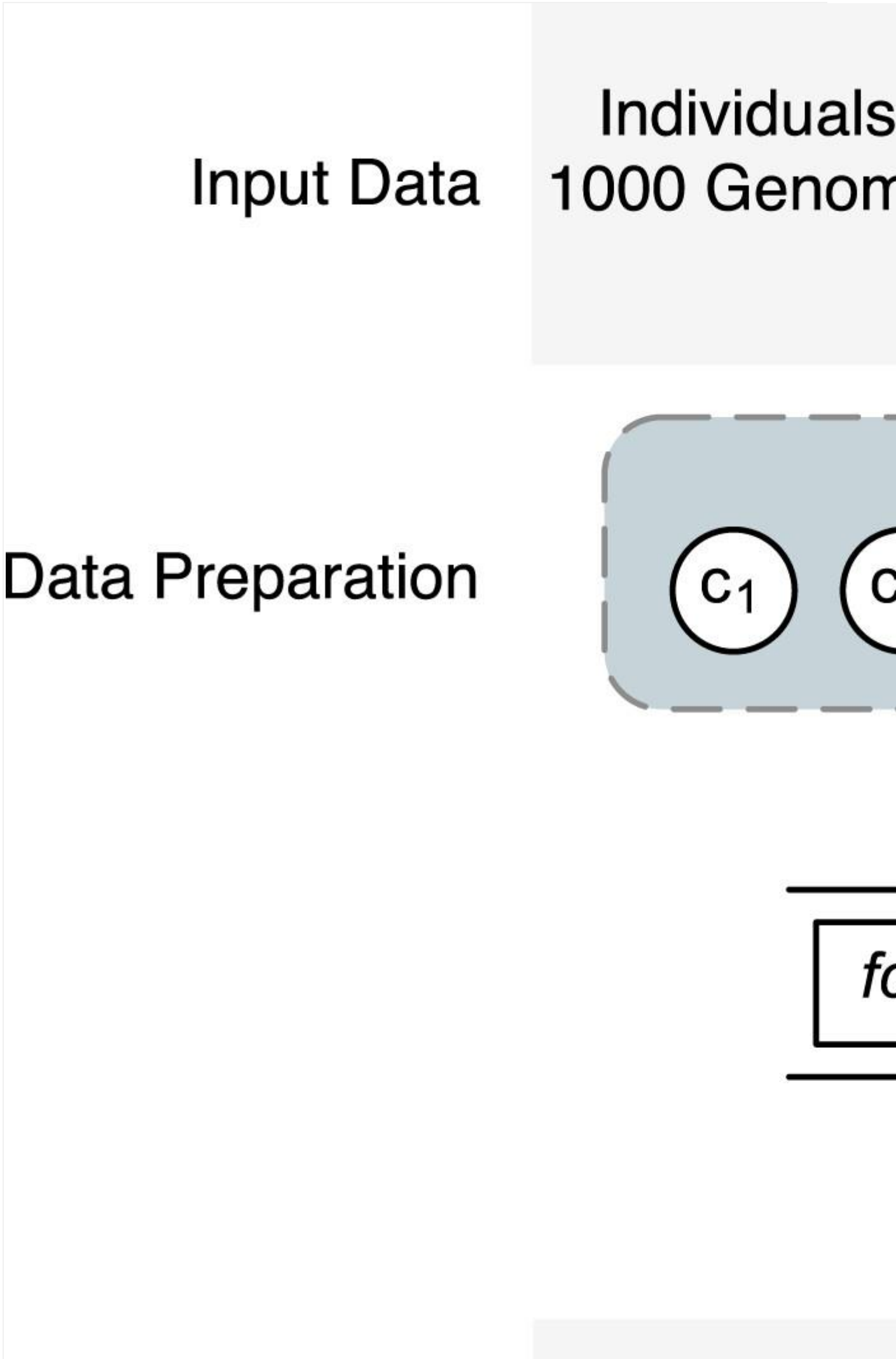
Input Data
Individuals
1000 Genom
Data Preparation
c1
c

Figure 2. 1000 Genome Pegasus workflow.Open in viewer

CASA Nowcast workflow: Nowcasts (Ruzanski and Chandrasekar, 2015) are short-term advection forecasts generated by mosaicing asynchronous individual CASA (2020) 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 3) contains 63 computing tasks, with one task to split the input data into individual grids and then 62 independent tasks to compute the reflectivity and the respective contour images.

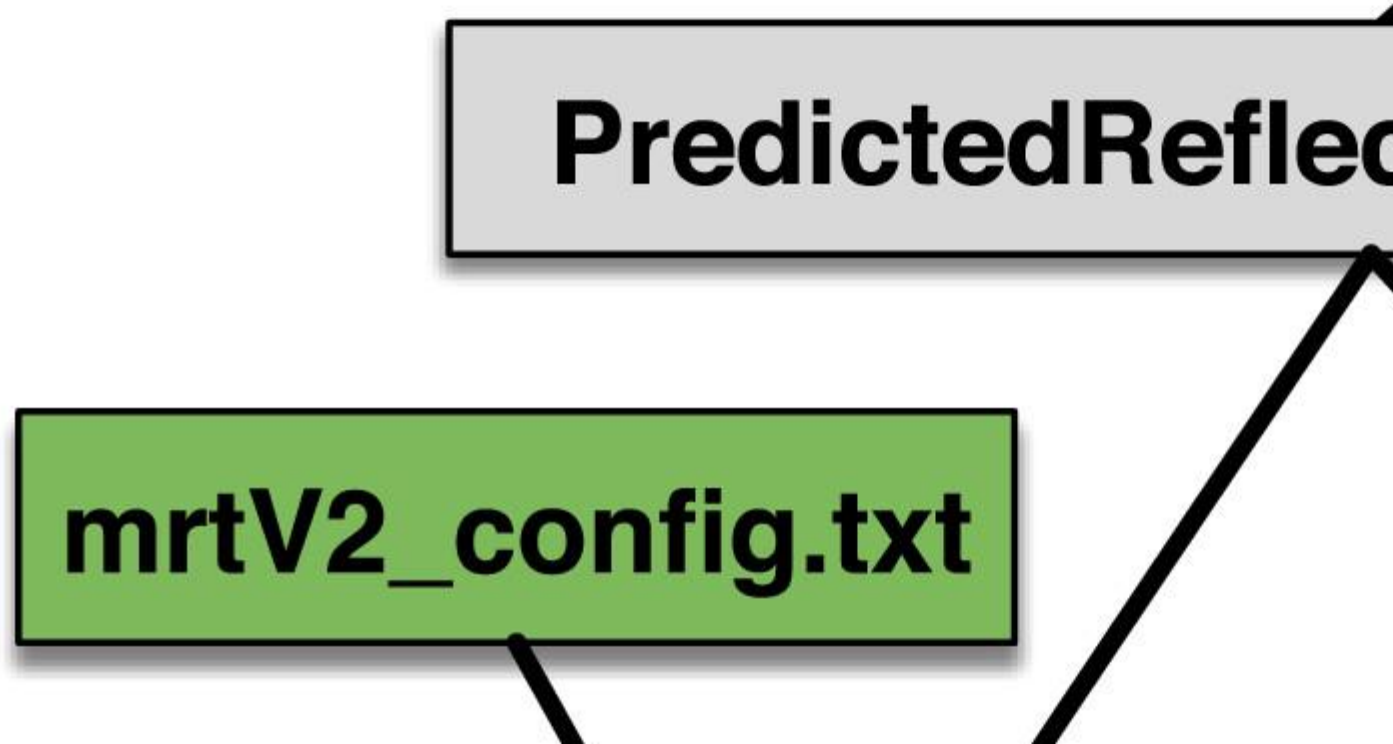
PredictedReflec
mrtV2_config.txt

Figure 3. CASA Nowcast Pegasus workflow.Open in viewer

CASA Wind workflow: This workflow (Lyons et al., 2019) 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 4) 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.

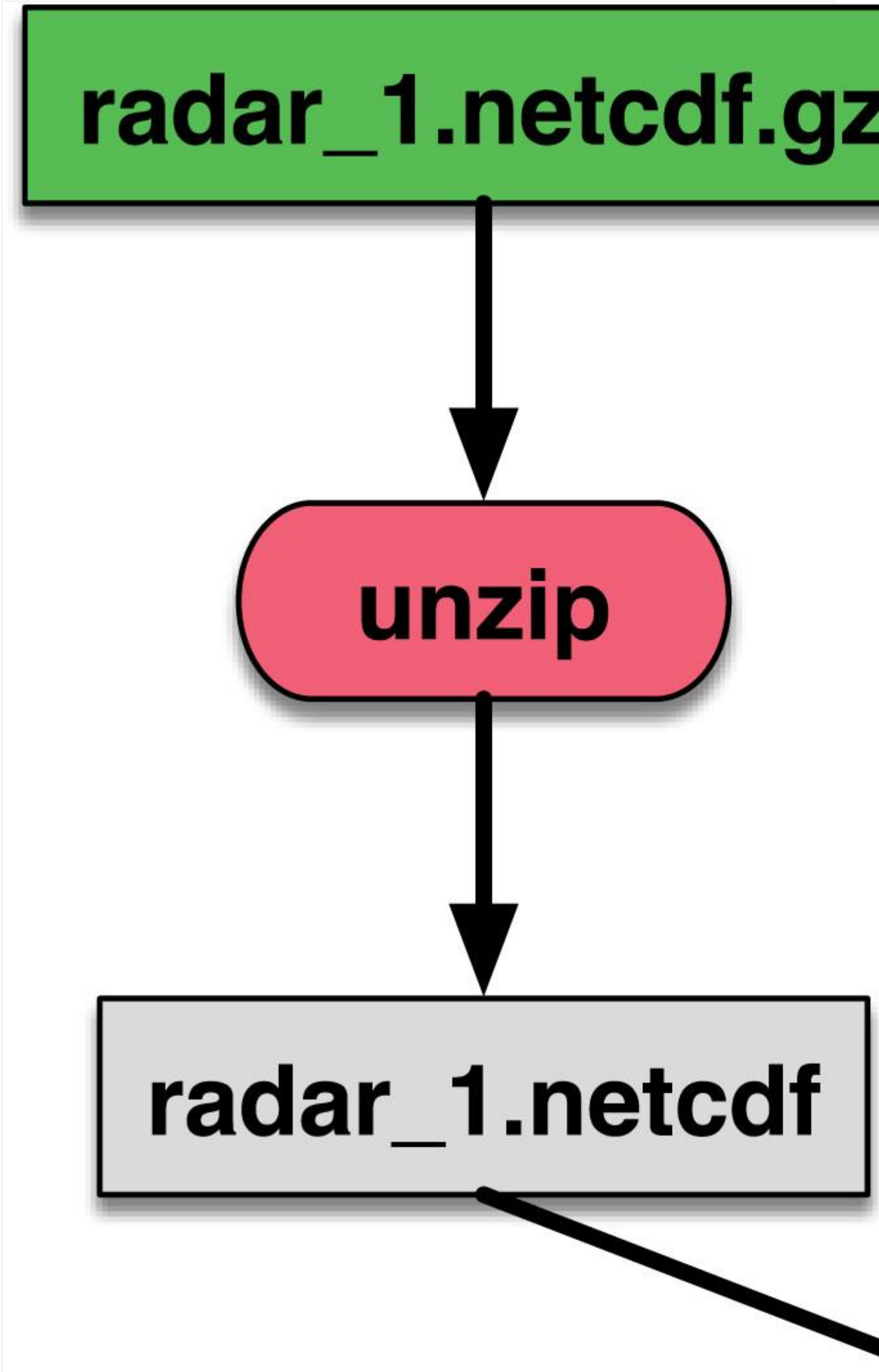
radar_1.netcdf.gz
unzip
radar_1.netcdf

Figure 4. CASA Wind Pegasus workflow.Open in viewer

Task clustering

Pegasus enables the clustering of workflow tasks into larger jobs via horizontal or label task clustering. This technique helps to optimize for the time spent waiting in the queue instead of doing computations, which is essential when a workflow has small short running tasks. With horizontal clustering, tasks on the same level are grouped together; with label clustering, Pegasus groups together tasks that carry the same label in their metadata. By employing the task clustering technique, the shape of the final executable workflow DAG changes, without changing the semantics of the workflow. And we can analyze the new DAG independently, as a new instance.

Executable workflows

In our experiments, we used the three workflows described in Section 4.1, but we invoked the workflows with different input sizes and different configuration settings to create six different executable workflow DAGs.
More specifically, we invoked the 1000 Genome workflow with two different input sizes (2 GBs vs 5 GBs). This resulted in a different number of nodes and edges in the two executable workflow instances, as seen in Table 1 and Table 2.
• **1000 Genome A**: This instance of the 1000 Genome workflow operated on the 2 GBs input.
• **1000 Genome B**: This instance of the 1000 Genome workflow operated on the 5 GBs input.

Table 1. Statistics of executable workflows—anomalies affecting all jobs.

Workflow	DAG		Normal	CPU				HDD		
	Nodes	Edges		2	3	50	60	70	80	90
1000 Genome A	57	129	200	125	125	150	75	75	75	75
CASA Nowcast w/ clustering 8	13	20	270	150	150	60	60	60	60	60
CASA Nowcast w/	9	12	270	150	150	60	60	60	60	60

Workflow	DAG		Normal	CPU				HDD		
	Nodes	Edges		2	3	50	60	70	80	90
clustering 16										
CASA Wind w/ clustering	7	8	270	150	150	60	60	60	60	60
CASA Wind w/o clustering	26	44	270	150	150	60	60	60	60	60

Open in viewer

Table 2. Statistics of executable workflows—anomalies affecting some jobs.

Workflow	DAG		Normal	CPU		HDD	
	Nodes	Edges		2	3	5	10
1000 Genome B (partial anomaly)	137	289	50	100	25	75	100

Open in viewer

Additionally, we enabled horizontal clustering for the CASA Nowcast workflow and label clustering for the CASA Wind workflow. The clustering details are as follows.

- **CASA Nowcast with clustering 8**: Horizontal clustering with maximum cluster size 8.
- **CASA Nowcast with clustering 16**: Horizontal clustering with max cluster size 16.
- **CASA Wind without clustering**: No clustering is used.
- **CASA Wind with clustering**: Label clustering, where all the compute tasks except the unzip tasks are grouped together.

The different sizes of the executable workflow DAGs for CASA Nowcast and CASA Wind workflows can be found in Table 1.

Data collection

To collect the data, we used the Pegasus Panorama extension (Deelman et al., 2017; Papadimitriou and Deelman, 2018), which offers advanced monitoring capabilities (Papadimitriou et al., 2021). 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, and stores the measurement data in an Elasticsearch instance (ELK, 2018). In this article, we collected two datasets, where in the first one anomalies are introduced to the entire workflow, while in the second one anomalies are introduced in some of the workflow jobs.

Dataset 1

For the first dataset, we executed the five workflows of Table 1 on the ExoGENI testbed (Baldin et al., 2016). We used seven virtual machines, with one *submit node*, five *worker nodes*, and one *data node*. The worker nodes were located within the same ExoGENI region, while the submit node and the data node were located in a different regions. Each virtual machine had four 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, with 1 Gbps links for each node. To facilitate workflow execution, we configured our nodes with Pegasus and HTCondor. On the data node, we used a standard web server to enable file transfers over HTTP.

To introduce synthetic network and I/O anomalies, we used the Linux Traffic Control (TC) toolset (Hubert et al., 2002). TC is able to replicate network anomalies such as delay, packet loss, and jitter, by configuring the Linux kernel packet scheduler. Additionally, to reduce the performance of the worker nodes, we used the stress tool (Waterland, 2013), a simple workload generator that can impose a configurable amount of CPU, memory, I/O, and disk stress on the system.

In this set of experiments, the injected anomalies were affecting all worker nodes, and as a result all jobs of the workflow. During our data collection, we generated 6,000 traces of the above workflows for four main classes, as seen in Table 1. Examples of how these anomalies affect workflow execution can be found in our prior work (Papadimitriou et al., 2021).

- **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.

Dataset 2

Dataset 2 differs from Dataset 1 in the following ways: (1) as summarized in Table 2, our second dataset is limited to the 1000 Genome workflow, which has been scaled up to more jobs (1000 Genome B); (2) the anomalies are now injected to some of the worker nodes, and, as a result, this affects only a subset of workflow jobs, but not all of them; and (3) as we will explain in the next paragraph, the experiments are conducted on Chameleon testbed (Keahey et al., 2020).

To generate the second dataset, we provisioned three bare-metal nodes on Chameleon testbed, with one *submit node* and two *container executor nodes*. By using Docker containers (Docker Inc. 2022), we virtualized the 2 container executor nodes to 20 workers (10 workers per node). The worker nodes were located within the same Chameleon region (Texas Advanced Computing Center—TACC), while the submit node

was located at the University of Chicago region. Each worker node had 4 cores and 16 GB RAM, that were assigned to them via Docker's runtime options (cpuset, memory). Connectivity between the two Chameleon regions was established over a high-speed layer 2 VLAN, with capped speed of 1 Gbps for each worker node. To facilitate the workflow execution, we configured our nodes with Pegasus and HTCondor, and the data was served by the submit node over HTTP.
In this dataset, to introduce synthetic anomalies, we used Docker's runtime options to limit and shape the performance of the spawned worker containers. This way, we can configure the amount of CPU time a worker node (container) gets, and limit the average I/O each worker node can perform. These capabilities are supported via Linux's control groups version 2 (cgroups v2) (Linux Kernel Organization 2023). This approach offers sufficient isolation among the workers and allows us to obtain reproducible results from each type of experiment. For this dataset, we generated 350 traces of the scaled up version of the 1000 Genome workflow for three main classes, as seen in Table 2.

- **Normal.** No anomaly is introduced—normal conditions.
- **CPU *K*.** Even though four cores are advertised on each worker node, on some nodes K cores are not allowed to be used ($K = 2, 3$).
- **HDD *K*.** On some worker nodes, the average write speed to the disk is capped at K MB/s and the read speed at $(2 \times K)$ MB/s ($K = 5, 10$).

Evaluation of the GNN model for anomaly detection

Model setup and metrics

For each of the five workflows, first we develop workflow-specific GNN models for the binary classification ($p = 2$ with normal and anomaly labels) and the multi-label classification case ($p = 4$ with CPU, HDD, packet loss, and normal labels). In addition to these 20 models, we consider a case "ALL" where all workflows are utilized together to train the GNN. Our work utilizes a distinctive approach with the "ALL" experiment, which enables a single GNN model to predict anomalies across workflows, even when the number of jobs in these workflows is not equal. We applied the "ALL" model to both binary and multi-label job anomaly cases, as well as workflow anomalies, shown in Tables 3 and 4.

Table 3. Graph-level classification.

Workflow	Binary				Multi-label
	Accuracy	F1	Recall	Precision	Accuracy
1000 Genome A	0.917 ± 0.014	0.915 ± 0.019	0.921 ± 0.009	0.938 ± 0.010	0.882 ± 0.006
Nowcast w/ clustering 8	0.768 ± 0.009	0.715 ± 0.017	0.778 ± 0.023	0.768 ± 0.15	0.792 ± 0.009
Nowcast w/ clustering 16	0.837 ± 0.012	0.675 ± 0.020	0.815 ± 0.012	0.837 ± 0.011	0.830 ± 0.007
Wind w/ clustering CASA	0.776 ± 0.002	0.652 ± 0.032	0.769 ± 0.021	0.776 ± 0.017	0.764 ± 0.19
Wind w/o clustering CASA	0.781 ± 0.02	0.853 ± 0.013	0.800 ± 0.012	0.781 ± 0.008	0.886 ± 0.007
1000 Genome B (partial anomaly)	1.000 ± 0.0	1.000 ± 0.0	1.000 ± 0.0	1.000 ± 0.0	1.000 ± 0.0
ALL	0.836 ± 0.006	0.878 ± 0.013	0.886 ± 0.011	0.856 ± 0.009	0.877 ± 0.008

Table 4. Node-level classification.

Workflow	Binary				Multi-label
	Accuracy	F1	Recall	Precision	Accuracy
1000 Genome A	0.782 ± 0.009	0.851 ± 0.005	0.810 ± 0.008	0.898 ± 0.015	0.745 ± 0.010
Nowcast w/ clustering 8	0.703 ± 0.023	0.751 ± 0.017	0.729 ± 0.013	0.823 ± 0.010	0.599 ± 0.006

Workflow	Binary				Multi-label
	Accuracy	F1	Recall	Precision	Accuracy
Nowcast w/ clustering 16	0.714 ± 0.011	0.695 ± 0.010	0.734 ± 0.020	0.780 ± 0.024	0.690 ± 0.015
Wind w/ clustering CASA	0.641 ± 0.020	0.680 ± 0.011	0.673 ± 0.009	0.779 ± 0.013	0.641 ± 0.011
Wind w/o clustering CASA	0.712 ± 0.007	0.688 ± 0.012	0.731 ± 0.013	0.771 ± 0.019	0.698 ± 0.013
1000 Genome B (partial anomaly)	0.777 ± 0.015	0.815 ± 0.019	0.785 ± 0.017	0.837 ± 0.007	0.723 ± 0.011
ALL	0.758 ± 0.010	0.777 ± 0.009	0.754 ± 0.020	0.839 ± 0.022	0.747 ± 0.012

Open in viewer

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 optimization, we employ Adam optimizer (Kingma and Ba, 2015) with a learning rate of $1e^{-3}$.

It is worth noting that our collected data are quite imbalanced, that is, the number of normal and anomaly samples differs significantly. Figure 5 shows the statistics of 1000 Genome B with partial anomalies for the binary classification problem. Instead of feeding the data into the training model, we explicitly specify the weights of the labels according to the number of samples in the training set. We adopt cross-entropy loss as the final prediction of probabilities for each label from outputs. Here, we introduce the weights as

$$wc=1-nctrain\sum cCnctrain,$$

(8)

where nctrain is the number of samples in training for label c. Therefore, we finalize our weighted cross-entropy loss by introducing the weights of labels W_c in the logrithmic of softmax loss. More specifically, it is defined as

$$\ell(X,y)=1N[-\sum cCwclogexpXn,c\sum iCexpxn,i].$$

(9)

Thus, we report not only the accuracy of the test dataset but also the F1 score, precision score, and recall, which are widely used metrics for imbalanced data.

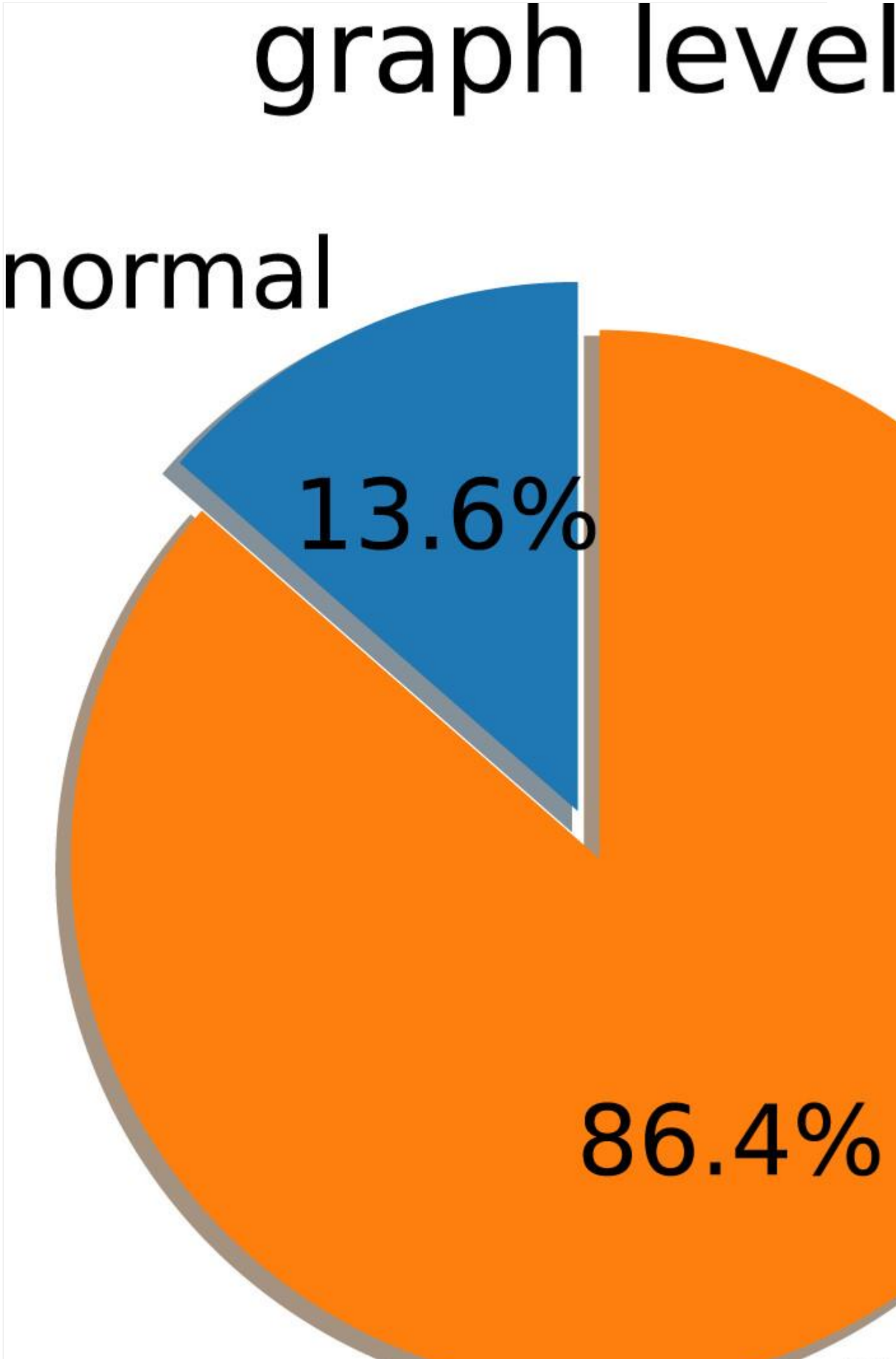
graph level
normal
13.6%
86.4%

Figure 5. Imbalanced data (1000 Genome B with partial anomaly).Open in viewer

Graph-level anomaly detection

To identify abnormalities in the workflow, we used a GCN module to analyze the graph, followed by an MLP module to make probabilistic predictions for each graph. The results of the graph classification on the test set are shown in Table 3 for both binary and multi-label settings. Furthermore, in order to apply our approach to various types of workflow, we also tested the ability of a single model to predict anomalies in multiple workflows at once, which is labeled as “ALL” in the final row of the table.

We demonstrated the accuracies and loss values during training of a single model for the 1000 Genome A workflows in Figures 6(a, binary) and (b, multi-label). We presented the mean accuracy and loss values for each epoch during the 10 rounds of evaluations. To start with, a randomly generated GNN model, which is presented as the first epoch in the figures, can be improved during the backpropagation update on parameters of the GNN model. The results indicate that binary classification outperforms multi-label anomaly detection in detecting normal versus anomaly, achieving 10% to 20% higher accuracies across various workflows. This suggests that binary classification is more effective in detecting anomalies compared to identifying specific anomaly categories.

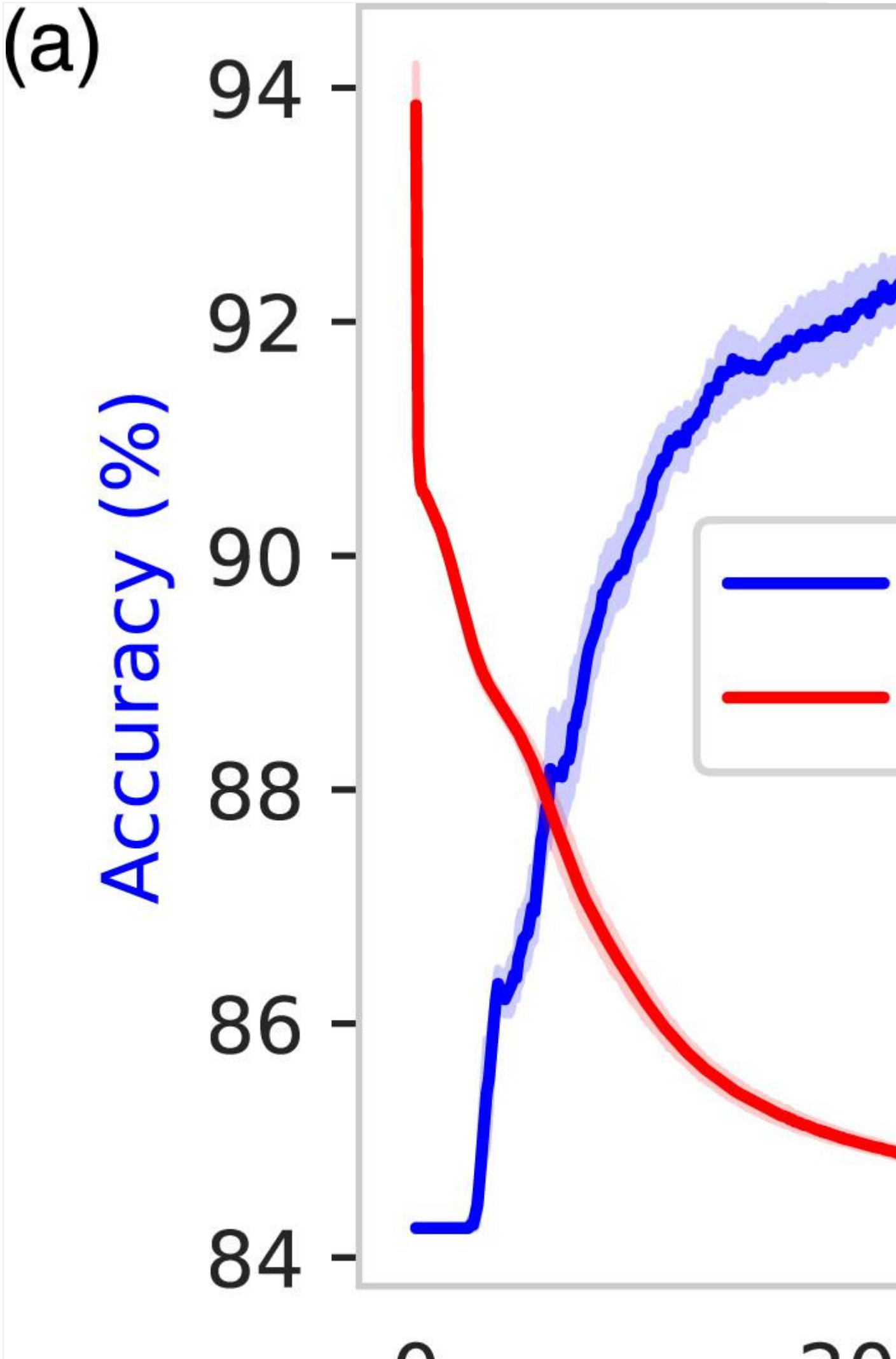
(a)
94
92
90
88
86
84
Accuracy (%)

Figure 6. Training for graph level anomaly detection (1000 Genome A).Open in viewer

Moreover, we investigated the rationale of the performance from the perspective of workflow structures. Figure 7 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, anomaly detection reaches higher accuracy. Even among the full anomaly workflows, the 1000 Genome A reached best and has more jobs involved. This is largely due to the intrinsic property of graph neural networks, where more complex structures help to aggregate from local neighbors through layer-wise propagation, overcoming the over-smoothing problems (Chen et al., 2020).

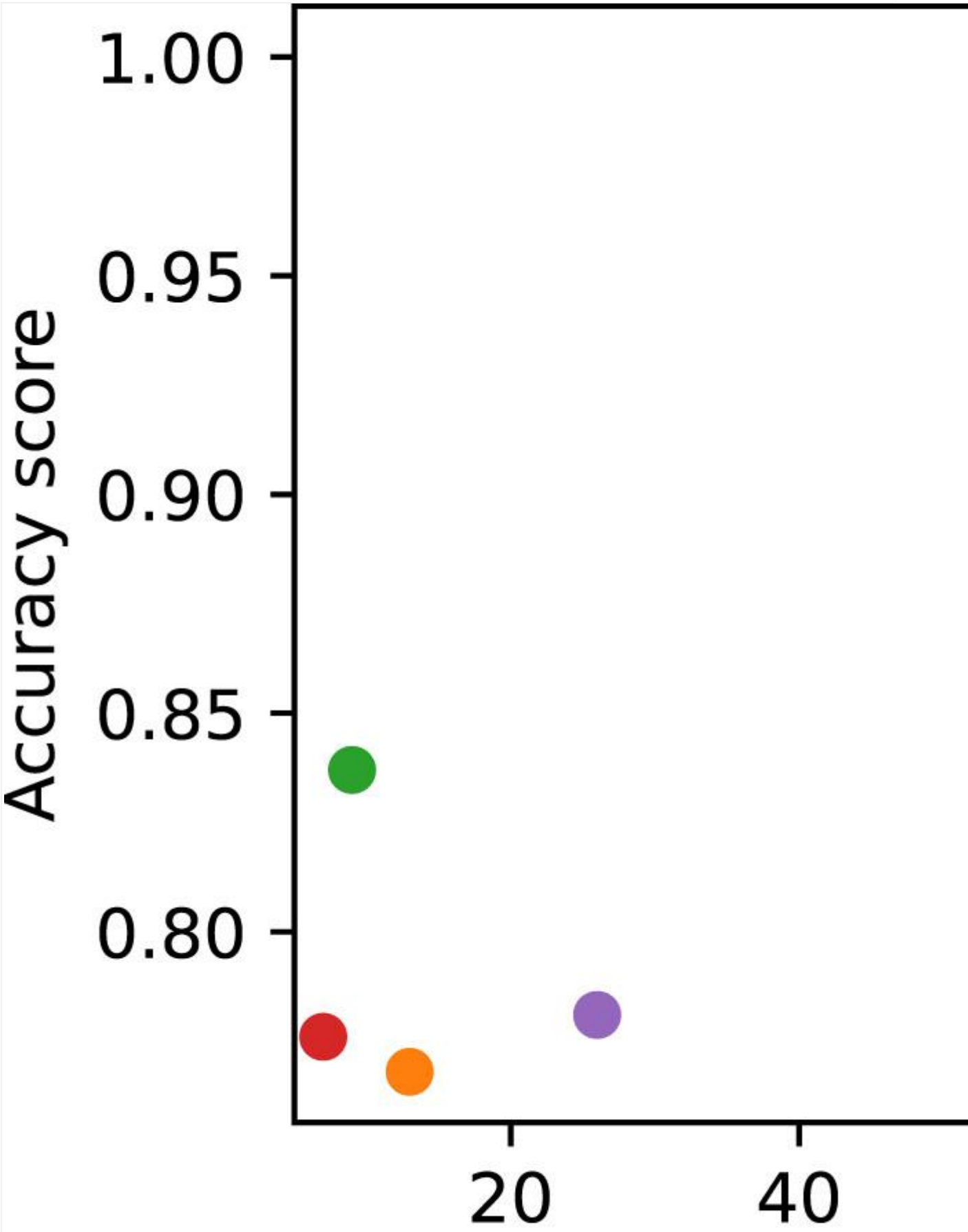

Figure 7. Rationale behind accuracy versus the number of jobs in the workflow.Open in viewer

Node-level anomaly detection

Another important issue in science workflows is identifying abnormal jobs (nodes) within a workflow (graph). Instead of using a single label for each run in a graph-level approach, we assign labels to individual nodes by adapting them from the entire run. This means that the labels of the jobs are the same as the label of the corresponding run. We randomly select jobs from all runs to use as training, validation, and testing sets, allowing the model to be trained with information from a variety of labels. Similar to the graph-level setting, we report the performance on the testing set in Table 4. “ALL” in the last row represents the performance of a single model that trains different workflows at the same time. We observe that the binary (normal vs anomaly) setting reaches better accuracy than the multi-label 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 0.777 even in the case of a single model that utilizes all workflows together. Figures 8(a) (a, binary) and (b, multi-label) show the accuracy and loss values on the training of a single model for the 1000 Genome A workflows in terms of node-level anomalies. Again, the results are reported with 10 repetitions.

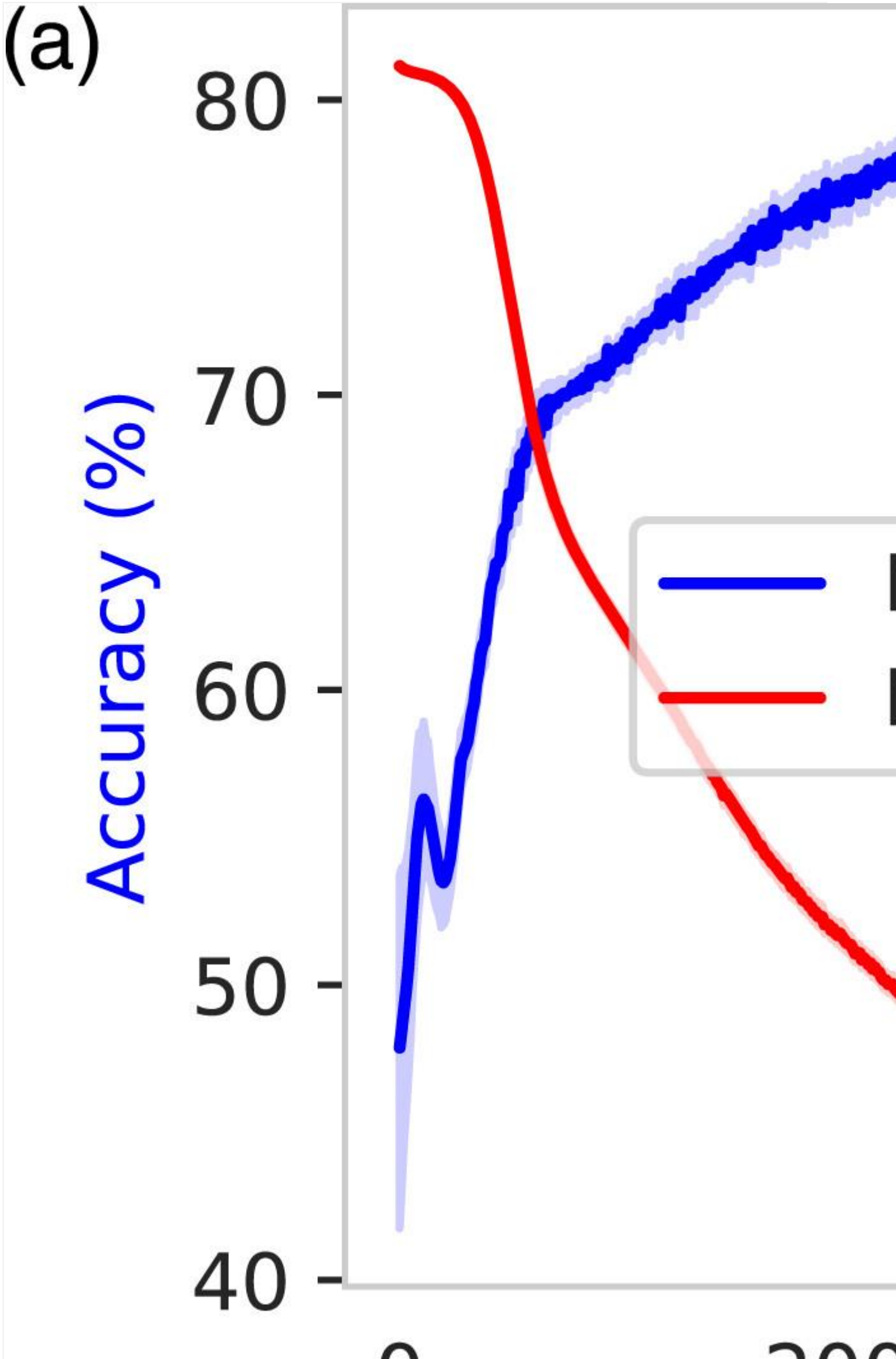
(a)
80
70
60
50
40
Accuracy (%)

Figure 8. Training for node level anomaly detection (1000 Genome A).Open in viewer

Anomaly detection by category

In both graph-level and node-level anomaly detection, using a binary label system results in better test accuracy compared to using a multi-label system. We examine the reason for this improved performance and study anomaly detection by categorizing them. As outlined in Section 4, there are three types of anomalies: CPU-related anomalies caused by overloading worker nodes, HDD-related anomalies caused by constant writing on worker nodes, and network packet-loss anomalies caused by dropped packets between regions. Table 5 compares the average accuracy across different anomaly categories for graph-level detection, with the last column indicating the accuracy for detecting a mixture of all anomalies as a binary classification. Among the three categories, HDD-related anomalies are typically the easiest to detect, with higher accuracy than the other two.

Table 5. Graph-level binary classification by anomaly categories.

Workflow	CPU	HDD	Packet loss	Joint anomalies
1000 Genome A	0.909	0.981	0.722	0.917
Nowcast w/ clustering 8	0.785	0.968	0.719	0.768
Nowcast w/ clustering 16	0.829	0.825	0.790	0.837
Wind w/ clustering CASA	0.762	0.779	0.693	0.776
Wind w/o clustering CASA	0.744	0.967	0.784	0.781
1000 Genome B (partial anomaly)	1.000	1.000	-	1.000

Open in viewer

Figures 9(a) and 9(b) show the confusion matrix, also known as the error matrix, to evaluate the multi-label classification for 1000 Genome A workflow with the single-model graph-level and node-level classifications, respectively. The value in each cell indicates the ratio of observations known to be in group *i* and predicted to be in group *j*, with row-wise summing being 1. The diagonal parts indicate the corrected predictions, and the higher, the better. These results also match our findings, given in Table 5: HDD-related anomalies achieve a higher score than the others do (on the diagonal).

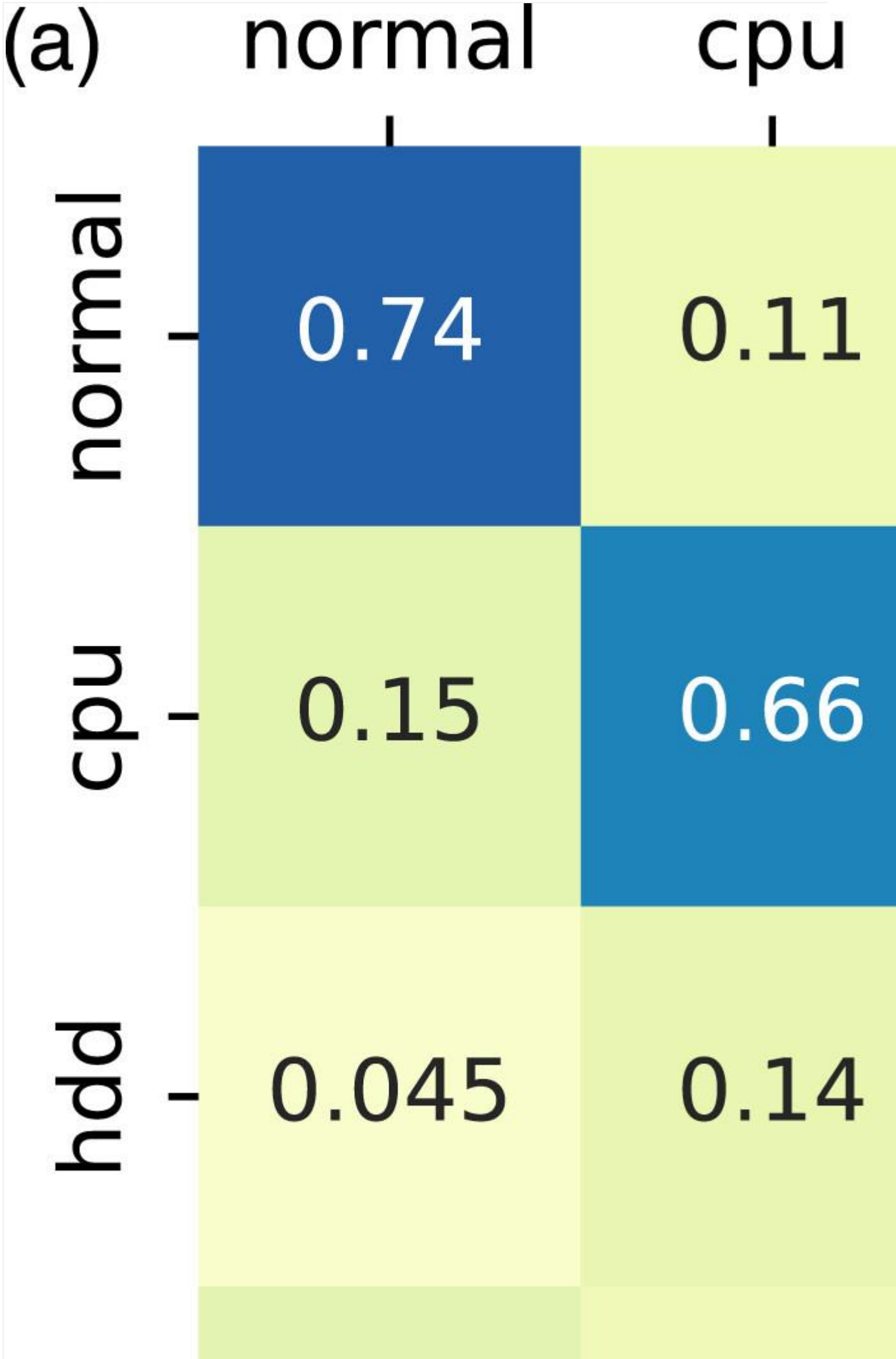
(a)
normal
cpu
normal
cpu
hdd
0.74
0.11
0.15
0.66
0.045
0.14

Figure 9. Confusion matrix of multi-label classification.Open in viewer

Anomaly detection for different anomaly levels

In order to evaluate the performance of anomaly detection, we artificially created anomalies by injecting varying levels of anomalies for CPU, HDD, and packet loss as described in Section 4. Table 6 shows the results of the graph-level anomaly detection based on these levels of anomalies. The levels of CPU 2 and 3 indicate the number of cores occupied by the 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 in the network connection. As expected, we can see that as the level of anomaly increases, the model's accuracy in classification improves. This is because the data corresponding to anomalies are increasingly distant from the distribution of normal runs in high-dimensional space.

Table 6. Graph-level binary classification by anomaly levels.

Workflow	CPU 2	CPU 3	HDD 60	HDD 100	Loss 0.5%	Loss 5%
1000 Genome A	0.985	1.000	0.927	0.985	0.760	1.000
Nowcast w/ clustering 8	0.857	0.902	0.985	0.985	0.788	0.970
Nowcast w/ clustering 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

Open in viewer

Model comparison

To evaluate the efficacy of our GNN for anomaly detection, we compared our GNN models with several standard machine learning models: (1) support vector machines (SVMs) with radial basis function kernels and $C = 1$, (2) multilayer perceptron with hidden layers (128, 128, 128), and (3) random forest (RF) with maximum depth set to 3. Previous work (Krawczuk et al., 2021) used a computer vision inspired deep learning approach by generating Gantt charts from node features. To maintain consistency, we used the same data split of 60%, 20%, and 20% for training, validation, and testing, respectively, and compared the performance of our GNN approach with the computer vision approach using the 1000 Genome A workflow. The comparison includes the use

of pre-trained models such as AlexNet, VGG-16, and ResNet-18 with data augmentation (rotations, flips, shifts, and augmented noise) and is shown in Table 7.

Table 7. Performance comparison on 1000 Genome A.

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	**0.917**	**0.921**	**0.939**	**0.915**

Open in viewer

As seen in the table, the GNNs outperform both the standard machine learning models and the Gantt chart representation of workflows, due to their ability to learn embeddings from both local structural information and node features. This is made possible by the inclusion of additional structural information from the workflow. It is worth noting that all the other hyperparameters were kept the same as in Section 5.1 and no fine-tuning was done. We will explicitly discuss the hyperparameter tuning to boost the performance of GNNs in next subsection.

In addition, we measured the training time of deep learning approaches for graph-level anomaly, to check the efficiency of our proposed GNN. We implemented the models in PyTorch and trained them with a single NVIDIA A100 40 GB GPU and measured the training time of deep learning approaches to check the efficiency of our proposed GNN. We reported the results for workflows in Table 8, where the computer vision approaches had similar runtimes across different workflow. This is because all the input sizes from generated Gantt chart need to be the same, regardless of the sizes from workflows. However, our graph neural networks have a significant improvement in efficiency as they have fewer parameters to train and their inputs vary based on the sizes of workflows. Reaping the advantages of aggregating local neighbors through matrix multiplication, the GNN approach is much faster than the CNN approach for training. Furthermore, as the workflows increase, we can use mini-batch and adaptive sample-based graph training (Huang et al., 2018).

Table 8. Runtime (sec.) of training deep learning models.

Workflows	AlexNet	VGG-16	ResNet-18	Our GNN
1000 Genome A	251	435	991	**142**
Nowcast w/ clustering 8	235	374	887	**112**
Nowcast w/ clustering 16	220	394	879	**93**
Wind w/ clustering CASA	241	413	931	**94**
Wind w/o clustering CASA	255	423	945	**104**

Open in viewer

Hyperparameter search

To improve the overall performance of GNNs in anomaly detection, we make use of DeepHyper, a framework that uses the Bayesian optimization method to search for the best hyperparameter settings. To demonstrate this approach, we apply it to the 1000 Genome A workflow with the task of binary node-level anomaly detection. As introduced in Section 3.1, our first target is the accuracy score, a single objective of the metric to measure the performance of our GNN model. We tune for the number of hidden dimensions in {16, 32, 64, 128, 256} and dropout rate in [0,0.5] for GNN models, and tune for the learning rate in [10−5,10−2] and decay rate in [0,10−3] in the Adam optimizer. We set a model evaluation budget of 500 epochs as a stopping criterion and report the accuracy attained versus the training time within the hyperparameter optimizer in Figure 10. The green dots represent the results without hyperparameter tuning (baseline) and the black ones are the reported results with hyperparameter tuning. Of course, with a slight change in hyperparameters, the model could perform worse than previously (lower accuracy). However, among the tuned hyperparameters, the red ones indicate the model trials with highest accuracy, which are better than the baseline model. Moreover, the HPS could reach a relatively high accuracy even with just tens of seconds.

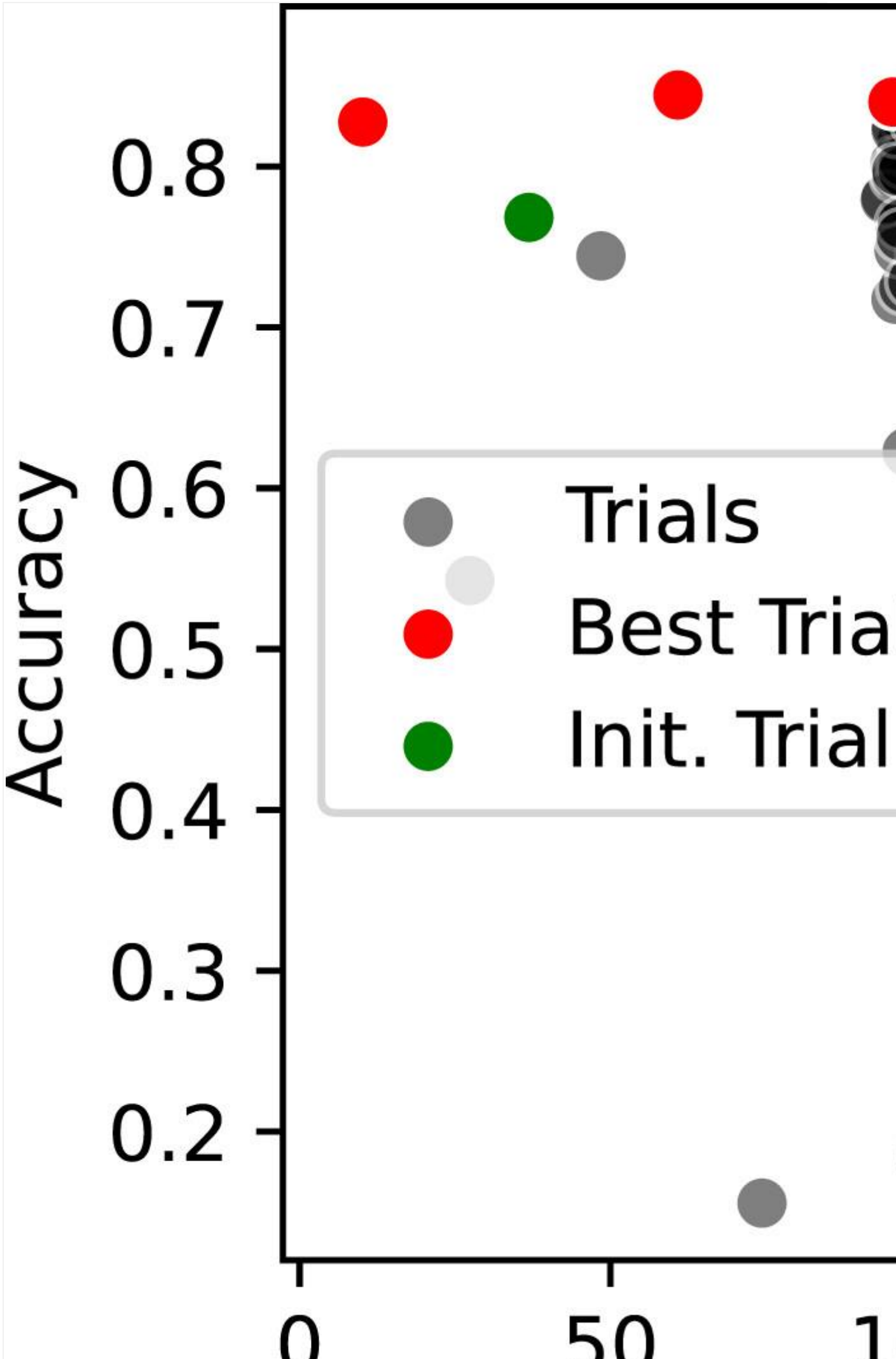
Accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Trials
Best Tria
Init. Trial
0
50

Figure 10. HPS for accuracy score.Open in viewer

Furthermore, considering that the data is quite imbalanced, we also tune the hyperparameter for both accuracy and AUC-ROC score. The latter computes the Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores, and it is typically used in imbalanced binary classification. Figure 11 shows the results after HPS with 500 model evaluations. Among all the results (black dots), the red ones represent the best choices after considering the trade-off between the accuracy and ROC-AUC scores. Both the single-objective and multi-objective HPS demonstrate that the performance of our GNN model can be improved effectively and efficiently.

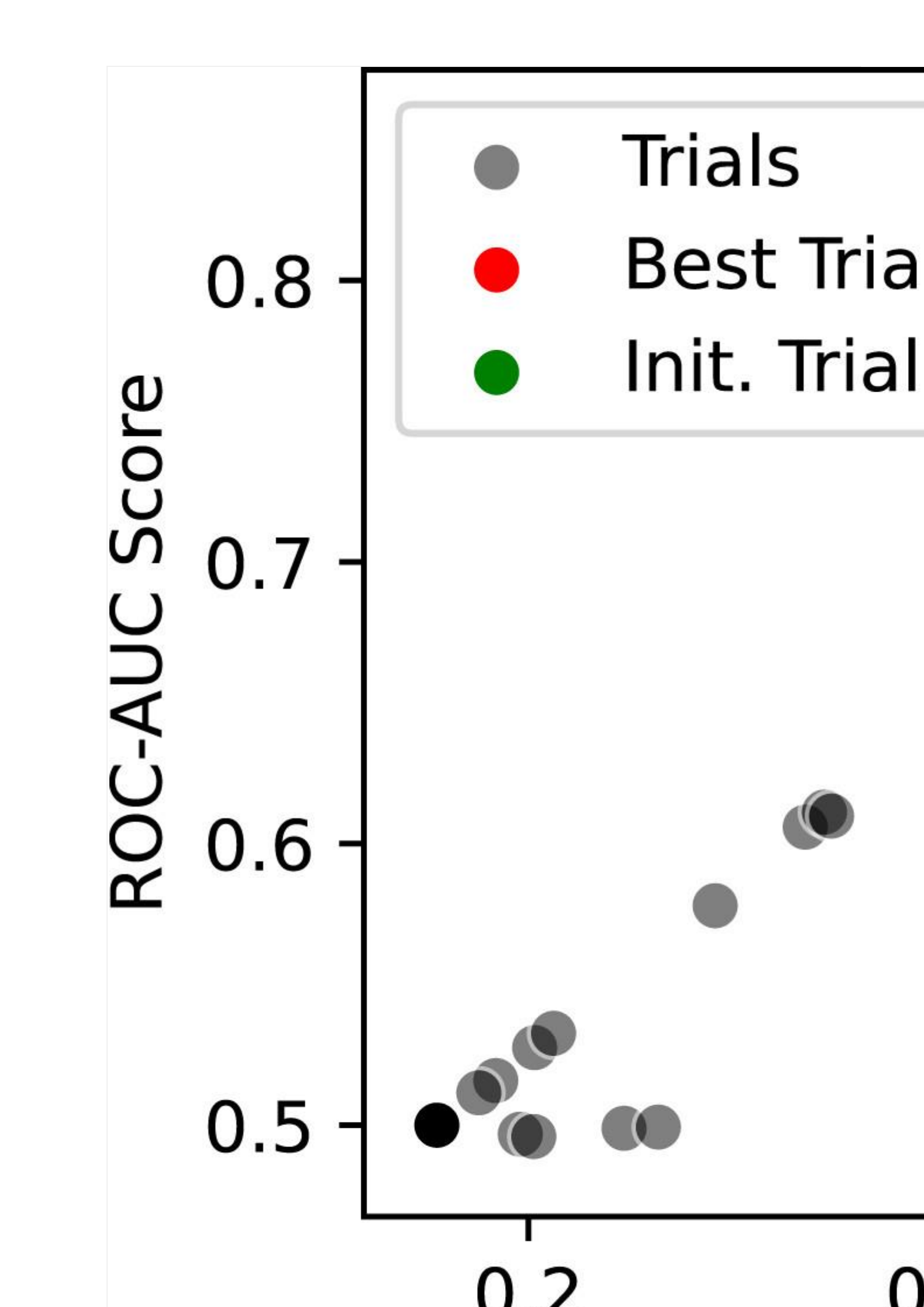
Trials
Best Tria
Init. Trial
ROC-AUC Score
0.8
0.7
0.6
0.5
0.2

Figure 11. HPS for both accuracy and ROC-AUC scores.Open in viewer

Explanation of GNNs

To explain the results from GNNs for anomaly detection, we apply the GNNExplainer to demonstrate both node-level and graph-level explanations. We use 1000 Genome B to provide the explanation of prediction on a real workflow. Recall in Section 3.2, we take advantage of GNNExplainer, which optimizes an upper bound of mutual information to identify the importance of both structure and feature information. To start, we pick node 23 of the workflow, which serves as a transfer node. Figure 12 shows its position and the 1-hop subgraph in the workflow. Figure 13 demonstrates the comparison between the normal job and anomaly job selected randomly from our collected data. Each column represents the raw node features, and its value is the log probabilities of the prediction associated with each feature. The higher value indicates a higher rate of importance in the prediction of anomalies. Figure 14, which is the feature-wise distribution between normal and anomaly of the same job across different simulations, supports our explanation from the GNNExplainer. For example, the *runtime* is the running time to complete the associated job. Intuitively, the job is more likely to be an anomaly when the runtime is higher than usual, which is identified by the explainer. This is essentially useful when we want to make decisions to optimize the workflow and reschedule the jobs.

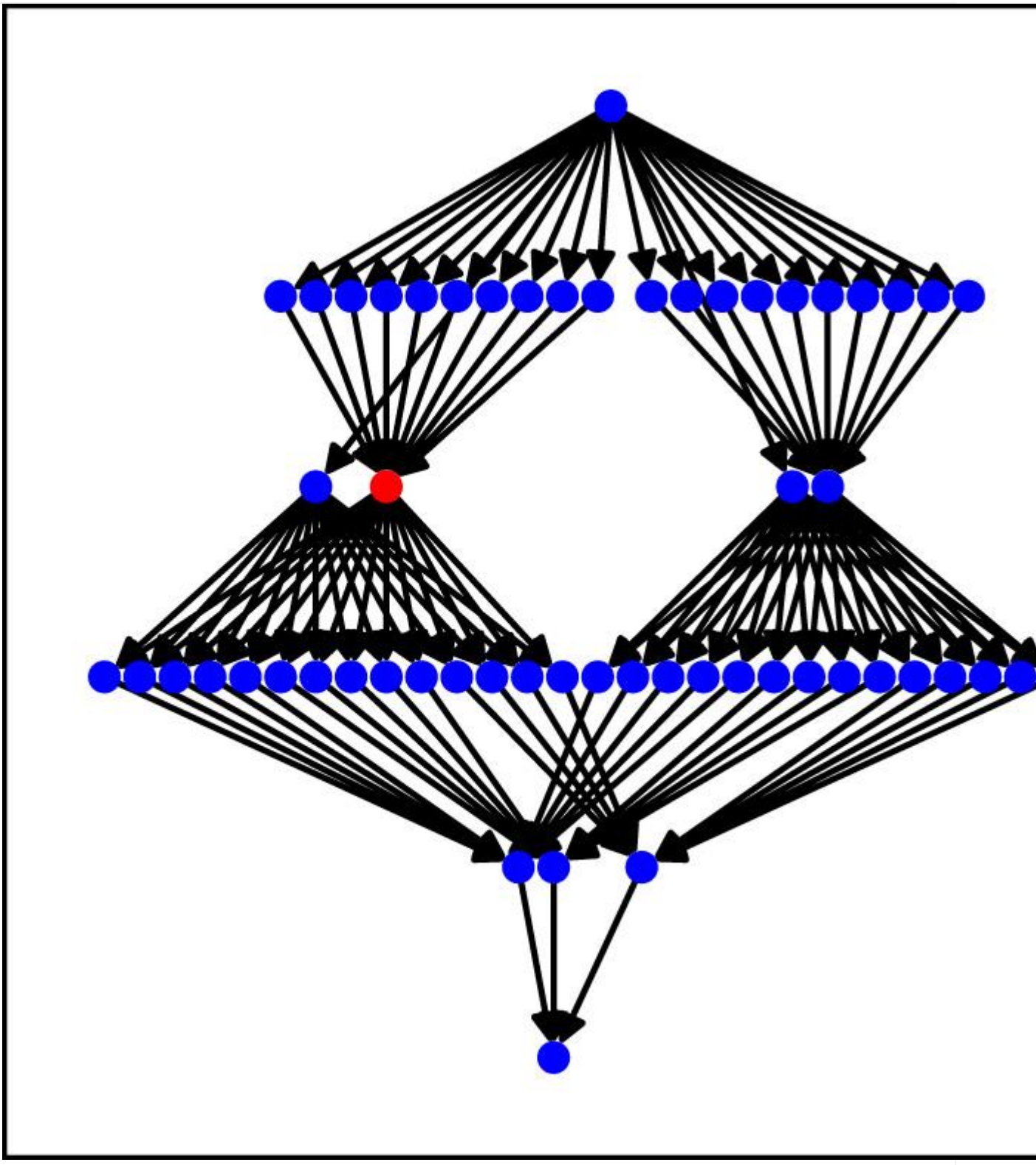

Figure 12. Demonstration of node 23 in 1000 Genome.Open in viewer

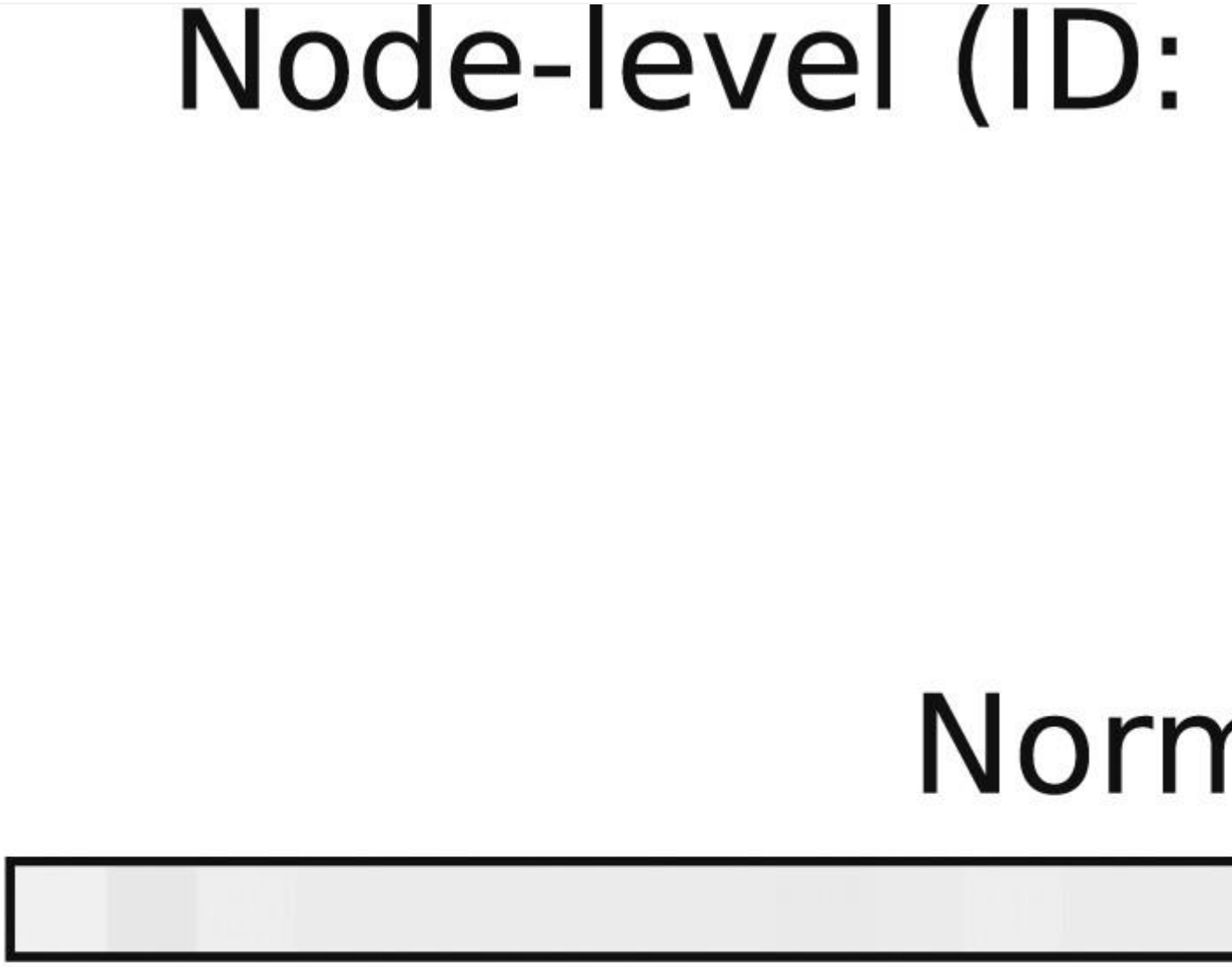
Node-level (ID:
Norm

Figure 13. Node-level explanation.Open in viewer

Normalized feature value
1.00
0.75
0.50
0.25
0.00
auxiliary
compute
transfer
is_clustered

Figure 14. Normalized features of 1000 Genome B (Job: 23).Open in viewer

Furthermore, to identify the anomalies of the entire workflow, that is, a single anomaly job in the workflow leads to abnormal behavior of the graph, we also pick the same node 23. Instead of investing its own, the explainer extracts a subgraph associated with it. Figure 15 demonstrates the difference between the normal workflow and anomaly workflow. Each row represents the job in the workflow, and each column represents the node feature. As we can see, the explainer identifies the anomaly node and the causes from its ancestors. In the scenario where a couple of its parent nodes are anomalous, it may affect the job itself, resulting in significant delays in collecting and transferring results to its successors. This aligns with our findings from the structure.

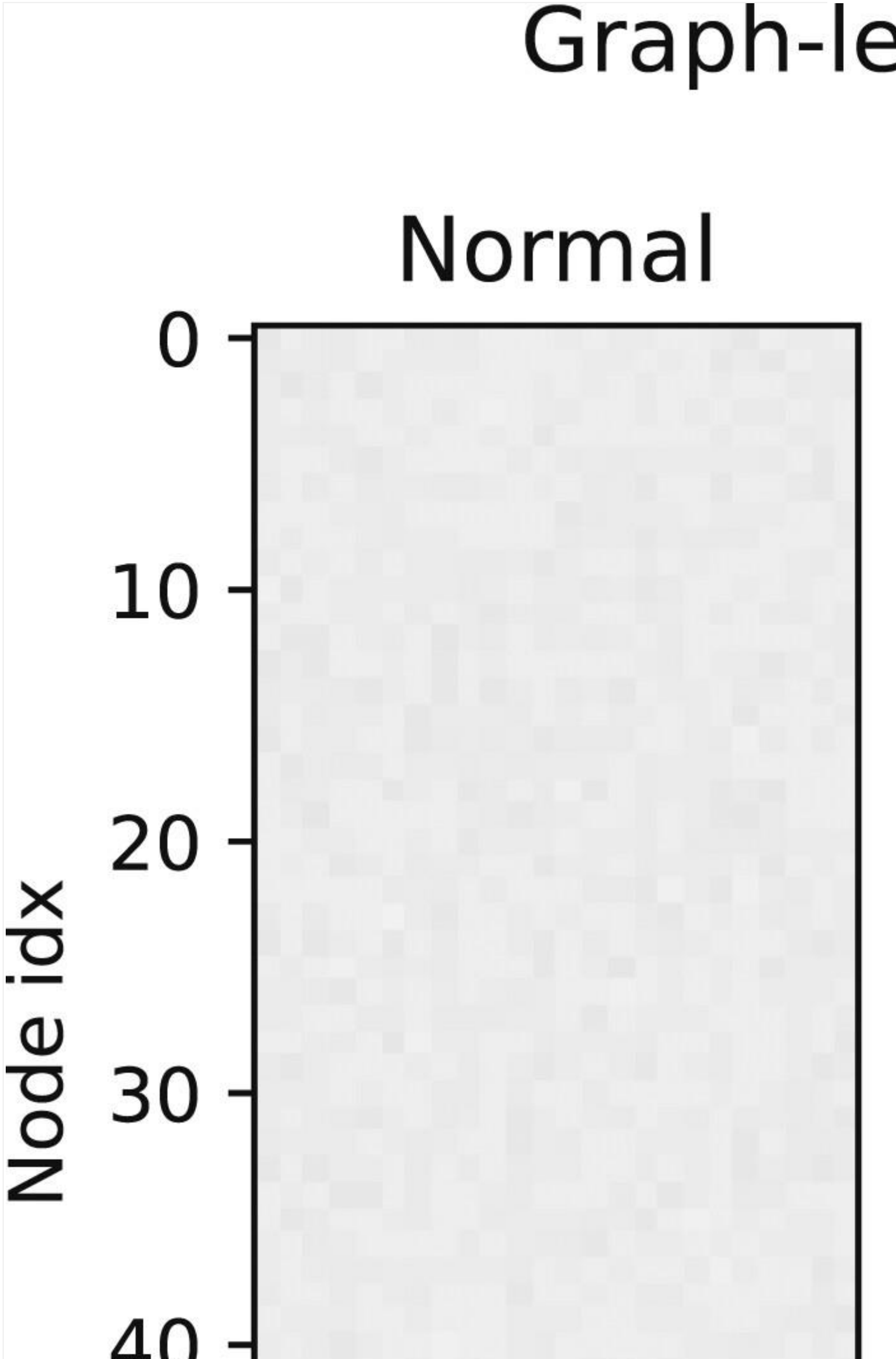
Graph-le
Normal
0
10
20
30
40
Node idx

Figure 15. Graph-level explanation.Open in viewer

Conclusion and future work

In this study, we examined anomaly detection in workflows by representing them as directed acyclic graphs and utilizing graph neural networks (GNNs) to identify anomalous workflows. By utilizing both node features and local structural information, our GNN models outperformed traditional machine learning and computer vision techniques. In addition, we boost the model performance by tuning hyperparameters in the GNN model and integrate explainable approaches to provide insights on job and graph level anomalies. In the future, we plan to investigate the efficiency of larger scientific workflows and apply our models to new workflow datasets. Additionally, since creating intentional anomalies can be time consuming, we plan to use generative models to generate synthetic anomaly data.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: 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.

ORCID iDs

Hongwei Jin
George Papadimitriou
Ewa Deelman

References

ELK stack (2018) https://www.elastic.co/elk-stack

Go to Reference

Google Scholar

Collaborative Adaptive Sensing of the Atmosphere (2020). http://www.casa.umass.edu/

Google Scholar

National Energy Research Scientific Computing Center (NERSC) (2022)

https://www.nersc.gov

Go to Reference

Google Scholar
Oak Ridge Leadership Computing Facility (OLCF) (2022) https://www.olcf.ornl.gov
Google Scholar
1000 Genomes Project Consortium (2015) A global reference for human genetic variation. *Nature* 526(7571): 68–74.
Go to Reference
Crossref
PubMed
Web of Science
Google Scholar
Balaprakash P, Salim M, Uram TD, et al. (2018) Deephyper: Asynchronous hyperparameter search for deep neural networks. In: 2018 IEEE 25th International Conference on High Performance Computing (HiPC). Bengaluru, India, 17-20 December 2018.
Go to Reference
Crossref
Google Scholar
Baldin I, Chase J, Xin Y, et al. (2016) ExoGENI: A multi-domain infrastructure-as-a-service testbed. In: McGeer R, Berman M, Elliott C, et al. (eds), *The GENI Book*. Springer International Publishing, pp. 279–315.
Go to Reference
Crossref
Google Scholar
Chen D, Lin Y, Li W, et al. (2020) Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *Proceedings of the AAAI Conference on Artificial Intelligence* 34: 3438–3445.
Go to Reference
Crossref
Google Scholar
Deelman E, Carothers C, Mandal A, et al. (2017) Panorama: An approach to performance modeling and diagnosis of extreme-scale workflows. *The International Journal of High Performance Computing Applications* 31(1): 4–18.
Crossref
Web of Science
Google Scholar
Deelman E, Mandal A, Jiang M, et al. (2019) The role of machine learning in scientific workflows. *The International Journal of High Performance Computing Applications* 33(6): 1128–1139.
Crossref
Web of Science
Google Scholar
Deelman E, Vahi K, Juve G, et al. (2015) Pegasus: a workflow management system for science automation. *Future Generation Computer Systems* 46: 17–35.

Crossref
Web of Science
Google Scholar
Docker (2022) Docker https://docs.docker.com/
Go to Reference
Google Scholar
Ferreira da Silva R, Filgueira R, Deelman E, et al. (2019) Using simple PID-inspired controllers for online resilient resource management of distributed scientific workflows. *Future Generation Computer Systems* 95: 615–628.
Go to Reference
Crossref
Google Scholar
Gaikwad P, Mandal A, Ruth P, et al. (2016) Anomaly detection for scientific workflow applications on networked clouds. In 2016 International Conference on High Performance Computing & Simulation (HPCS). Innsbruck, Austria, 18-22 July 2016, pp. 645–652.
Go to Reference
Crossref
Google Scholar
Huang W, Zhang T, Rong Y, et al. (2018) Adaptive sampling towards fast graph representation learning. *Advances in Neural Information Processing Systems* 31: 11.
Go to Reference
Google Scholar
Hubert B, Graf T, Maxwell G, et al. (2002) *Linux advanced routing & traffic control*. Ottawa Linux Symposium, volume 213.
Go to Reference
Google Scholar
Wyatt MR, Herbein S, Gamblin T, et al. (2022) AI4IO: A suite of ai-based tools for io-aware scheduling. *The International Journal of High Performance Computing Applications* 36(3): 370–387.
Crossref
Web of Science
Google Scholar
Jamieson K, Talwalkar A (2016) Non-stochastic best arm identification and hyperparameter optimization. *Artificial Intelligence and Statistics*. PMLR, pp. 240–248.
Go to Reference
Google Scholar
Jin H, Raghavan K, Papadimitriou G, et al. (2022) Workflow anomaly detection with graph neural networks. In: *2022 IEEE/ACM Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, pp. 35–42.
Go to Reference
Crossref
Google Scholar

Keahey K, Anderson J, Zhen Z, et al. (2020) Lessons learned from the chameleon testbed. In Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20). 2020, USENIX Association.
Go to Reference
Google Scholar
Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In International Conference on Learning Representations.
Go to Reference
Google Scholar
Krawczuk P, Papadimitriou G, Nagarkar S, et al. (2021) Anomaly detection in scientific workflows using end-to-end execution Gantt charts and convolutional neural networks. In *Practice and Experience in Advanced Research Computing*, pp. 1–5.
Crossref
Google Scholar
Li F, Song F (2019) Building a scientific workflow framework to enable real-time machine learning and visualization. *Concurrency and Computation: Practice and Experience* 31(16): e4703.
Crossref
Google Scholar
Kernel Organization L (2023) *Control groups v2* https://docs.kernel.org/admin-guide/cgroup-v2.html
Google Scholar
Lyons E, Papadimitriou G, Wang C, et al. (2019) 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), San Diego, CA, 24-27 September 2019, pp. 67–76.
Go to Reference
Crossref
Google Scholar
Papadimitriou G, Deelman E (2018) *Pegasus panorama*. https://github.com/pegasus-isi/pegasus/tree/panorama
Google Scholar
Papadimitriou G, Kiran M, Wang C, et al. (2019) Training classifiers to identify TCP signatures in scientific workflows. In: 2019 *IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*. Los Alamitos, CA: IEEE Computer Society, pp. 61–68. https://doi.ieeecomputersociety.org/10.1109/INDIS49552.2019.00012
Go to Reference
Crossref
Google Scholar
Papadimitriou G, Wang C, Vahi K, et al. (2021) End-to-end online performance data capture and analysis for scientific workflows. *Future Generation Computer Systems* 117: 387–400. http://www.sciencedirect.com/science/article/pii/S0167739X20330570

Crossref
Google Scholar
Pordes R, Petravick D, Kramer B, et al. (2007) The open science grid. *Journal of Physics: Conference Series* 78: 012057. https://doi.org/10.1088%2F1742-6596%2F78%2F1%2F012057
Go to Reference
Google Scholar
Rodriguez MA, Kotagiri R, Buyya R (2018) Detecting performance anomalies in scientific workflows using hierarchical temporal memory. *Future Generation Computer Systems* 88: 624–635.
Go to Reference
Crossref
Google Scholar
Ruzanski E, Chandrasekar V (2015) Weather radar data interpolation using a kernel-based Lagrangian nowcasting technique. *IEEE Transactions on Geoscience and Remote Sensing* 53(6): 3073–3083.
Go to Reference
Crossref
Google Scholar
Singh A, Altintas I, Schram M, et al. (2018) Deep learning for enhancing fault tolerant capabilities of scientific workflows. In: 2018 IEEE International Conference on Big Data (Big Data). Seattle, WA, 10-13 December 2018, IEEE, pp. 3905–3914.
Go to Reference
Crossref
Google Scholar
Stevens R, Taylor V, Nichols J, et al. (2020) *AI for Science*. Argonne, IL (United States): Argonne National Lab.(ANL). Technical report.
Go to Reference
Google Scholar
Taufer M (2021) AI4IO: A suite of ai-based tools for io-aware HPC resource management. In 2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC). Bengaluru, India, 17-20 December 2021.
Go to Reference
Crossref
Google Scholar
Taylor IJ, Deelman E, Gannon DB, et al. (2014) *Workflows for E-Science: Scientific Workflows for Grids*. Springer Publishing Company
Go to Reference
Google Scholar
Towns J, Cockerill T, Dahan M, et al. (2014) XSEDE: Accelerating scientific discovery. *Computing in Science & Engineering* 16(05): 62–74.
Go to Reference

Crossref
Google Scholar
Wang C, Papadimitriou G, Kiran M, et al. (2020) Identifying execution anomalies for data intensive workflows using lightweight ML techniques. In 2020 IEEE High Performance Extreme Computing Conference (HPEC). Waltham, MA, 22-24 September 2020, pp. 1–7.
Go to Reference
Crossref
Google Scholar
Waterland A (2013) *POSIX Workload Generator*.
Go to Reference
Google Scholar
Welling M, Kipf TN (2016) Semi-supervised classification with graph convolutional networks. In J. International Conference on Learning Representations. *ICLR 2017)*.
Go to Reference
Google Scholar
Ying R, Bourgeois D, You J, et al. (2019) GNNExplainer: Generating explanations for graph neural networks. *Advances in Neural Information Processing Systems* 32: 9240–9251.
Go to Reference
PubMed
Google Scholar

Biographies

Hongwei Jin received his Ph.D. in computer science from the University of Illinois at Chicago in 2022. Before that, he got his M.S. in applied mathematics from the Illinois Institute of Technology. He had a couple of works that have been published in top-tier machine learning conferences including NeurIPS, UAI, IJCAI, ECML-PKDD, etc.
Krishnan Raghavan received his Ph.D. in computer engineering from Missouri University of Science and Technology in May 2019. He has co-authored several papers on deep neural networks with applications to big data.
George Papadimitriou is a Computer Science PhD candidate at the University of Southern California, and a Graduate Research Assistant in the Science Automation Technologies group at the USC Information Sciences Institute. His research interests lie within the intersection of Data Intensive Applications and Distributed Computing. He received his BS in Electrical and Computer Engineering from the National Technical University of Athens.
Cong Wang is a senior network and systems researcher at RENCI, University of North Carolina at Chapel Hill. His research focuses on cloud computing, networking, and distributed systems. He obtained his PhD in the department of Electrical and Computer Engineering at University of Massachusetts Amherst.
*Anirban Mandal*serves as the Assistant Director for network research and infrastructure at Renaissance Computing Institute (RENCI) at University of North Carolina, Chapel Hill. He leads several efforts in cyberinfrastructure research in support of science. His research interests lie in the areas of distributed systems, cloud

computing, networking, and data-driven scientific workflows. His research deals with resource provisioning, scheduling, performance analysis, machine learning, and anomaly detection for large scale scientific cyberinfrastructures, next generation networks and experimental testbeds. Prior to joining RENCI, he earned his PhD degree in Computer Science from Rice University in 2006 and a Bachelor's degree in Computer Science & Engineering from IIT Mumbai, India in 2000.

Mariam Kiran is a research scientist with shared positions with Energy Sciences Network and the Scientific Data Management (SDM) group in Computational Research Division. Her work specifically concentrates on using advanced software and machine learning techniques to advance system architectures, particularly high-speed networks such as DOE networks. Her current work is exploring reinforcement learning, unsupervised clustering and classification techniques to optimally control distributed network resources, improving high-speed big data transfers for exascale science applications and optimize how current network infrastructure is utilized. Kiran is the recipient of the DOE ASCR Early Career Award in 2017. Before joining LBNL in 2016, Kiran held positions as a lecturer and research fellow at the Universities of Sheffield and Leeds in the UK. She earned her undergrad and PhD degree in software engineering and computer science from the University of Sheffield, UK in 2011.

Prasanna Balaprakash is the director of AI Program and Distinguished R&D Scientist at Oak Ridge National Laboratory. His research interests span the areas of artificial intelligence, machine learning, optimization, and high-performance computing. Currently, he seeks to deliver foundational, scalable, and applied AI/ML capabilities supporting Oak Ridge National Laboratory's broad mission and provides world-class solutions in computer and computational science, neutron science, materials science, biology and health science, nuclear engineering, isotopes, manufacturing, energy, and climate science. He is a recipient of U.S. Department of Energy 2018 Early Career Award. He is the machine-learning team lead and data-understanding team co-lead in RAPIDS, the SciDAC Computer Science institute. Prior to ORNL, he was the R&D Group Leader and Computer Scientist in the Mathematics and Computer Science Division with a joint appointment in the Leadership Computing Facility at Argonne National Laboratory.

Ewa Deelman received her PhD in Computer Science from the Rensselaer Polytechnic Institute in 1998. Following a postdoc at the UCLA Computer Science Department she joined the University of Southern California's Information Sciences Institute (ISI) in 2000, where she is serving as a Research Director and is leading the Science Automation Technologies group. She is also a Research Professor at the USC Computer Science Department and an AAAS and IEEE Fellow. Dr. Deelman's research interests include the design and exploration of collaborative, distributed scientific environments, with particular emphasis on workflow management as well as the management of large amounts of data and metadata. At ISI, Dr. Deelman is leading the Pegasus project, which designs and implements workflow mapping techniques for large-scale applications running in distributed environments. Pegasus is being used today in a number of scientific disciplines, enabling researchers to formulate complex computations in a declarative way.

Similar articles:

- Restricted access

  Explainable spatially explicit geospatial artificial intelligence in urban analytics

  Show details Hide details

  Pengyuan Liu and more ...

  Environment and Planning B: Urban Analytics and City Science

  Sep 2023

- Restricted access

  PANORAMA: An approach to performance modeling and diagnosis of extreme-scale workflows

  Show details Hide details

  Ewa Deelman and more ...

  The International Journal of High Performance Computing Applications

  Jul 2015

- Available access

  The role of machine learning in scientific workflows

  Show details Hide details

  Ewa Deelman and more ...

  The International Journal of High Performance Computing Applications

  May 2019

- Open Access

  Understanding COVID-19 vaccine hesitancy of different regions in the post-epidemic era: A causality deep learning approach

  Show details Hide details

  Yang Liu and more ...

DIGITAL HEALTH

Sep 2024

- Restricted access

  Improved GNN based on Graph-Transformer: A new framework for rolling mill bearing fault diagnosis

  Show details Hide details

  Dongxiao Hou and more ...

  Transactions of the Institute of Measurement and Control

  Jul 2024

- Open Access

  OSG-GEM: Gene Expression Matrix Construction Using the Open Science Grid

  Show details Hide details

  William L. Poehlman and more ...

  Bioinformatics and Biology Insights

  Aug 2016

- Open Access

  TVGCN: Time-varying graph convolutional networks for multivariate and multifeature spatiotemporal series prediction

  Show details Hide details

  Feiyan Sun and more ...

  Science Progress

  Sep 2024

- Restricted access

  Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments

  Show details Hide details

Ewa Deelman

The International Journal of High Performance Computing Applications

Dec 2009

- Open Access

  Forecasting Traffic Speed during Daytime from Google Street View Images using Deep Learning

  Show details Hide details

  Junfeng Jiao and more ...

  Transportation Research Record

  May 2023

View more

Sage recommends:

- **SAGE Knowledge**

  Entry

  Web Geoprocessing Workflows

  Show details Hide details

  Bastian Schaeffer

  Encyclopedia of Geography

  2010

- **SAGE Research Methods**

  Entry

  Machine Learning

  Show details Hide details

  Machine Learning

  2020

- **SAGE Research Methods**

  Whole book

  Introducing Social Networks

  Show details Hide details

  Alain Degenne and more...

  Introducing Social Networks

  1999

- **SAGE Research Methods**

  Data

  Learn About Consensus Clustering in R With Data From Zacharyâ□□s Karate Club (1977)

  Show details Hide details

  Feng Shi and more...

  SAGE Research Methods Datasets

  2019

- **SAGE Knowledge**

  Entry

  Network Simulations

  Show details Hide details

  Kathleen M. Carley and more...

  Encyclopedia of Social Networks

  2011

- **SAGE Research Methods**

  Whole book

  Social Network Analysis

Show details Hide details

Song Yang and more...

Social Network Analysis

2017

- **SAGE Research Methods**

  How to Visually Analyse Networks Using Gephi

  Show details Hide details

  Axel Bruns

  How to Visually Analyse Networks Using Gephi

  2022

- **SAGE Knowledge**

  Book chapter

  Modeling in a Broader Context

  Show details Hide details

  Stephan Lewandowsky and more...

  Computational Modeling in Cognition: Principles and Practice

  2011

- **SAGE Research Methods**

  Book chapter

  Introduction

  Show details Hide details

  Nick Crossley

  Social Network Analysis for Ego-Nets

  2015

View more

Sage recommends:

- **SAGE Knowledge**

  Entry

  Web Geoprocessing Workflows

  Show details Hide details

  Bastian Schaeffer

  Encyclopedia of Geography

  2010

- **SAGE Research Methods**

  Entry

  Machine Learning

  Show details Hide details

  Machine Learning

  2020

- **SAGE Research Methods**

  Whole book

  Introducing Social Networks

  Show details Hide details

  Alain Degenne and more...

  Introducing Social Networks

  1999

- **SAGE Research Methods**

  Data

  Learn About Consensus Clustering in R With Data From Zacharyâ□□s Karate Club (1977)

Show details Hide details

Feng Shi and more...

SAGE Research Methods Datasets

2019

- **SAGE Knowledge**

  Entry

  Network Simulations

  Show details Hide details

  Kathleen M. Carley and more...

  Encyclopedia of Social Networks

  2011

- **SAGE Research Methods**

  Whole book

  Social Network Analysis

  Show details Hide details

  Song Yang and more...

  Social Network Analysis

  2017

- **SAGE Research Methods**

  How to Visually Analyse Networks Using Gephi

  Show details Hide details

  Axel Bruns

  How to Visually Analyse Networks Using Gephi

  2022

- **SAGE Knowledge**

Book chapter

Modeling in a Broader Context

Show details Hide details

Stephan Lewandowsky and more...

Computational Modeling in Cognition: Principles and Practice

2011

- **SAGE Research Methods**

  Book chapter

  Introduction

  Show details Hide details

  Nick Crossley

  Social Network Analysis for Ego-Nets

  2015

View more

Also from Sage

- CQ Library Elevating debateopens in new tab
- Sage Data Uncovering insightopens in new tab
- Sage Business Cases Shaping futuresopens in new tab
- Sage Campus Unleashing potentialopens in new tab
- Sage Knowledge Multimedia learning resourcesopens in new tab
- Sage Research Methods Supercharging researchopens in new tab
- Sage Video Streaming knowledgeopens in new tab
- Technology from Sage Library digital servicesopens in new tab

Back to top

About

- Sage microsites

Information for

- Authors
- Editors
- Librarians
- Promoters / Advertisers
- Researchers
- Reviewers
- Societies
- Frequently asked questions

The International Journal of High Performance Computing Applications

- ISSN: 1094-3420
- Online ISSN: 1741-2846

- About Sage
- Contact us
- CCPA - Do not sell my personal information
- CCPA
- Privacy Policy