Anomaly Detection in Scientific Workflows using End-to-End Execution Gantt Charts and Convolutional Neural Networks

Patrycja Krawczuk
krawczuk@isi.edu
University of Southern California
USA

George Papadimitriou
georgpap@isi.edu
University of Southern California
USA

Shubham Nagarkar
slnagark@usc.edu
University of Southern California
USA

Mariam Kiran
mkiran@es.net
Energy Sciences Network (ESnet)
Lawrence Berkeley National Lab
USA

Anirban Mandal
anirban@renci.org
RENCI, University of North Carolina
Chapel Hill
USA

Ewa Deelman
deelman@isi.edu
University of Southern California
USA

ABSTRACT

Fundamental progress towards reliable modern science depends on accurate anomaly detection during application execution. In this paper, we suggest a novel approach to tackle this problem by applying Convolutional Neural Network (CNN) classification methods to high-resolution visualizations that capture the end-to-end workflow execution timeline. Subtle differences in the timeline reveal information about the performance of the application and infrastructure's components. We collect 1000 traces of a scientific workflow's executions. We explore and evaluate the performance of CNNs trained from scratch and pre-trained on ImageNet [7]. Our initial results are promising with over 90% accuracy.

CCS CONCEPTS

• **Computing methodologies** → **Computer vision**; **Neural networks**; • **General and reference** → **Reliability**.

KEYWORDS

Scientific Workflows, Gantt Charts, Anomaly Detection, Convolutional Neural Networks

ACM Reference Format:
Patrycja Krawczuk, George Papadimitriou, Shubham Nagarkar, Mariam Kiran, Anirban Mandal, and Ewa Deelman. 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 (PEARC '21), July 18–22, 2021, Boston, MA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3437359.3465597

1 INTRODUCTION

Modern scientific experiments are conducted on complex, large-scale, distributed high-performance infrastructure like DOE Leadership Computing Facilities (e.g., OLCF [2]). Even though these systems are designed with reliability in mind [25], they can experience anomalies ranging from subtle (e.g. network performance degradation) to critical (e.g., file system integrity errors) [20], affecting the performance of the applications leveraging their resources and increasing the chances of failures.

Workflow management systems have emerged as very important tools for managing the execution of science applications. Automation, reproducibility, resiliency and monitoring are some of key advantages they can provide. However, anomalies remain a significant barrier to the reliable execution of scientific workflows at scale. It is still difficult to understand subtle anomalies that impact performance, and provide useful feedback to users and administrators.

In this article we introduce a novel approach to detect anomalies in scientific workflow traces, leveraging advances in computer vision. We make the following contributions: (1) introduce a method to compute and visualize end-to-end workflow execution timelines (Gantt charts) from workflow traces, (2) explore and evaluate different CNN architectures when applied to the Gantt chart anomaly detection problem, and (3) investigate whether transfer learning from pre-trained models on ImageNet [7] leads to better accuracy.

2 BACKGROUND

2.1 Pegasus Workflow Management System

Pegasus [6] is a popular workflow management system that enables users to design workflows at a high-level of abstraction. The workflow descriptions are independent of the resources available to execute the workflow tasks and are also independent of the location of data and executables. Pegasus transforms these abstract workflows into executable workflows that can be deployed onto distributed and high-performance computing resources such as DOE Leadership Computing Facilities (e.g., NERSC [1] and OLCF [2]), shared computing resources (e.g., XSEDE [29], OSG [18]), local clusters, and academic and commercial clouds. During the compilation process, Pegasus performs data discovery, locating input data files and executables. During execution, Pegasus offloads the workflow tasks to HTCondor [28], a comprehensive job management system.

2.2 Deep Learning Methods for Computer Vision

2.2.1 Convolutional Neural Network (CNN). Convolutional Neural Network is a type of deep learning architecture. CNNs automatically extract relevant hierarchical features from image data. The

PEARC '21, July 18–22, 2021, Boston, MA, USA

ACM ISBN 978-1-4503-8292-2/21/07.
https://doi.org/10.1145/3437359.3465597

network consists of sequences of layers like convolutions, pooling, and normalization layers with nonlinear units between them. The layers in the sequence learn progressively abstract representations of the input data [15]. CNNs progressively downsample the spatial resolution of an image while increasing the depth of their feature maps. The lower level layers identify simple aspects of training data like edges, curves, corners while the upper layers of a model combine these features and create complex and objective-specific representations. CNNs achieve excellent performance on a number of important computer vision tasks ([27], [31]).

2.2.2 Transfer Learning with CNNs. Transfer Learning is a design methodology that transfers knowledge of a model trained on a large-scale, well-annotated dataset in one domain, to a target domain, where labeled data is scarce [32]. The pre-trained model can be used as a feature extractor by keeping the model's weights fixed (frozen) except for the final fully connected layers, which are randomly initialized. Alternatively, the model can be fine-tuned by initializing its weights based on pre-training and allowing some of the last layers in the architecture to remain updatable. These weights are updated during each training iteration to fit the dataset in a target domain. This method often proves to be faster than training a network with randomly initialized weights.

3 DATASET

To conduct our analysis we collect traces of the 1000 Genome Pegasus workflow [21], under normal and synthetic anomalous conditions, and we label each run with the corresponding type. We then create a methodology to parse the generated traces and produce detailed Gantt charts capturing the entire life-cycle of a workflow.

3.1 1000 Genome Pegasus Workflow

This Pegasus workflow 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 [4]. The workflow identifies mutational overlaps using data from the 1000 Genomes project in order to provide a null distribution for rigorous statistical evaluation of potential disease-related mutations [10]. Figure 1 depicts the workflow structure and is composed of five different 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 SNPs (single nucleotide polymorphisms) variants, as computed by the Variant Effect Predictor; (4) *pair overlap mutations* – measures the overlap in mutations (SNPs) among pairs of individuals; and (5) *frequency overlap mutations* – calculates the frequency of overlapping mutations across subsamples of certain individuals.

3.2 Execution Environment

To execute the 1000 Genome workflow we provision resources from the ExoGENI [5] testbed, which included 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 had 4 2.2 Ghz vCPUs, 10 GB RAM and 75 GB storage, and the

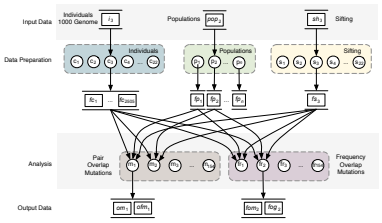

Figure 1: Overview of the 1000Genome Pegasus workflow.

connectivity between the two ExoGENI regions is established over a high speed layer 2 VLAN. To facilitate the workflow execution we configure our nodes with Pegasus and HTCondor, and on the data node we install a web server to serve files over HTTP.

3.3 Injecting Anomalies

To introduce synthetic network and I/O anomalies we use the Linux Traffic Control (TC) tool-set [12]. TC is able to replicate network anomalies such as delay, packet loss and jitter, by configuring the Linux kernel packet scheduler. Additionally, in order to reduce the performance of the worker nodes, we use Stress [30], a simple workload generator that can impose configurable amount of CPU, memory, I/O, and disk stress on the system.

3.4 Workflow Data Collection

For the data collection we use the Pegasus Panorama branch [22]. This Pegasus branch offers advanced monitoring capabilities over the stable Pegasus release. Panorama [17] 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 into an Elasticsearch instance [3].

During our data collection we generate 1000 traces of the 1000 Genome Pegasus workflow for 4 main classes as seen in Table 1. Examples of how these anomalies affect the workflow execution can be found in prior work [17].

Normal. No anomaly introduced - optimal conditions

CPU. 2 - 3 cores are occupied by stress tool on each worker node

HDD. 50 - 100 megabytes are continuously written by stress tool on each worker node

Loss. The network connection between the two ExoGENI regions is experiencing 0.1% - 5.0% of packet loss

3.5 Data Transformation into Gantt Charts

To generate our final dataset in the form of Gantt charts, we parse the generated Pegasus logs and the collected data in Elasticsearch, and we calculate delays, time spent transferring data, and time spent on computation for each individual job of every workflow run. More specifically each job timeline consists of the following phases:

Ready time. Timestamp since the beginning of the workflow, where all dependencies have been met and job can be dispatched

Pre script delay. Time spent on a script that is executed before job submission (if it exists)

WMS delay. Time spent by the workflow management system to prepare and submit the job

Table 1: Collected traces for each label.

Labels	Normal	CPU		HDD						Loss				
		2	3	50	60	70	80	90	100	0.1%	0.5%	1.0%	3.0%	5.0%
# Traces	250	125	125	67	37	35	31	31	49	50	50	50	50	50

Queue delay. Time spent in the queue waiting for resources
Stage in delay. Time spent transferring input data
Runtime. Time spent during computation
Stage out delay. Time spent transferring data to the intermediate scratch directory or final output directory
Post script delay. Time spent on a script executed after job indent exits (e.g., WMS parses stdout and exit code)
Completion time. Timestamp marking job completion, since beginning of workflow

Finally, using the above job timelines, we create one Gantt chart for each workflow executed (1000 Gantt charts) and we visualize them as shown in Figure 2 as high-resolution images (3875 by 3875 pixels).

4 APPROACH

Having generated the high-resolution visualizations of the Gantt charts we aim to leverage advances in computer vision, such as CNN and their ability to automatically extract relevant hierarchical features from images, and to classify the different types of anomalies we introduced. We also investigate the transferability of natural image features to our classification problem via transfer learning.

4.1 Data Manipulation

Image Preprocessing. Throughout our experiments we resize all the generated Gantt chart images to 512 by 512 pixels using bilinear interpolation, in order to reduce processing time and memory requirements.

Data Split. We keep a consistent data split across all of the experiments: the training set contains 80% of the data, while the validation and the testing sets contain 10% each.

Data Augmentation. To increase the robustness of our models and prevent overfitting we explore a number of data augmentation techniques [23]. We experiment with rotations, flips, shifts, and add noise. However, in our final experiments (Section 5), we only apply horizontal flips and jitter, since these transformations keep the semantic information intact and do not dramatically degrade the performance of our models.

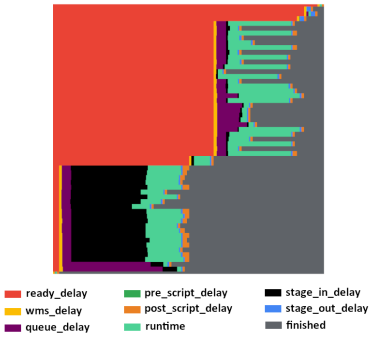

Figure 2: A visualized Gantt chart from one of the 1000 Genome workflow executions.

Table 2: Performance of simple CNNs trained from scratch.

Model	Acc.	Recall	Prec.	F-score	Time (s)
our CNN	**0.900**	**0.900**	**0.907**	**0.900**	77.09
our CNN+aug	0.870	0.870	0.885	0.869	77.85

4.2 Architectures

4.2.1 Simple Convolutional Neural Network (CNN). We design a simple CNN architecture and train it from scratch. Our network consists of a sequence of *5 convolutional layers* with Leaky Rectified Linear Unit (LeakyReLU) activations and 2D batch normalization layers between them. The head of the network comprises of *3 linear layers* interwoven with ReLU activations and dropout layers [26]. We initialize the weights of the model with random Gaussian distribution. We employ the Adam [13] optimizer and early stopping with patience of 4. The other hyperparameters are learning rate:$10^{-4}$and dropout rate: 0.4.

4.2.2 CNN with Transfer Learning. In this set of experiments, we investigate whether CNNs architectures trained on a large-scale, well-annotated dataset of natural images (ImageNet [7]) can support the classification of anomalies in the Gantt charts. We explore three progressively more complex models: AlexNet [14], VGG-16 [24], and ResNet-18 [11]. First, we replace the last classification layers in the architectures with *two fully connected layers* separated by ReLU activations, 1D batch normalization, and dropout layers. We adjust the architecture to fit our classification problem, with the final output of 4 classes. During training, we vary the number of transferred layers in which weights are frozen.

For all the pre-trained experiments we employ the Adam optimizer, early stopping with patience of 4 and batch size of 16. We fine-tuned the pre-trained layers with learning rate of 10^{-8} and train fully connected layers with learning rate of 10^{-6}.

5 EXPERIMENTAL RESULTS

In this section, we present an evaluation of our anomaly detection approach using the CNN models described in Section 4.2. Besides performance statistics like accuracy, recall or precision we record training time until convergence for all of our experiments.

5.1 Simple CNN Trained from Scratch

The results of experiments with simple CNN architecture trained from scratch are displayed in Table 2. We observe that adding data augmentation methods like horizontal flips and jitter lowers the performance of the model. As shown in Figure 4, the validation error continues to decrease with the training error until epoch 8 for training without data augmentation and epoch 7 for experiment

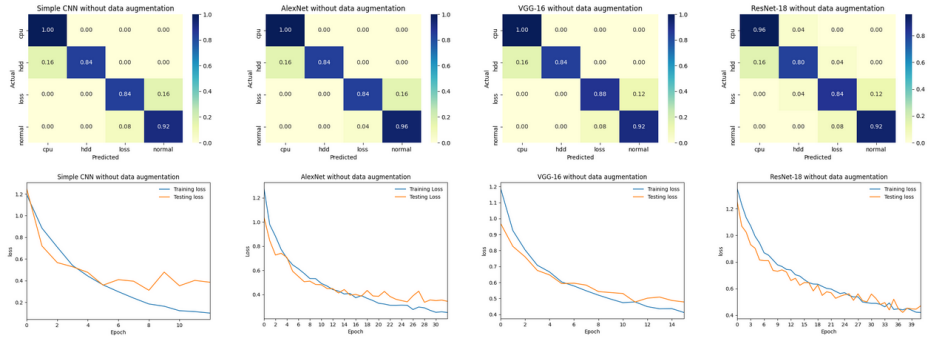

Figure 3: Confusion matrices and training curves for the simple model trained from scratch and pre-trained models.

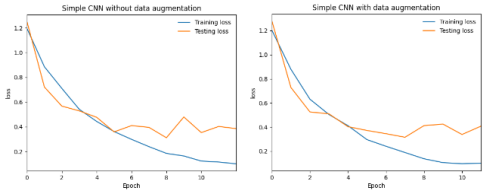

Figure 4: Loss curves for simple CNN model train without and with data augmentation.

with data augmentation. The loss curves for experiment with augmented data is slightly smoother but it ultimately leads to lower accuracy. This behavior could be attributed to small inter-class differences and the highly specific structure of our Gantt chart dataset.

Table 3: Performance of pre-trained models (fine-tuned).

Model	Acc.	Recall	Prec.	F-score	Time (s)
Without Data Augmentation					
AlexNet	**0.910**	**0.910**	**0.918**	**0.910**	222.75
VGG-16	**0.910**	**0.910**	**0.916**	**0.910**	283.42
ResNet-18	0.880	0.880	0.881	0.879	320.43
With Data Augmentation					
AlexNet	0.910	0.910	0.918	0.910	268.83
VGG-16	**0.930**	**0.930**	**0.916**	**0.930**	288.41
ResNet-18	0.890	0.890	0.900	0.890	325.52

5.2 CNN with Transfer Learning

In Table 3 we present results for AlexNet, VGG-16 and ResNet-18. The models are pre-trained on the ImageNet dataset, which is diverse and distinctly different from our synthetically developed Gantt chart images. We perform experiments with different hyperparameters and vary the number of transferred layers. We are able to achieve a 1% increase in accuracy compared to training the simple CNN from scratch. Among the pre-trained models, ResNet-18 is the most complex with 11M parameters. We notice a slight degradation in the performance of ResNet-18 in comparison with AlexNet or VGG-16. This is due to the limited quantity of data that is insufficient to utilize bigger architectures.

In contrast to our simple CNN trained from scratch, the addition of horizontal flips and jitter does not degrade the performance of the pre-trained models. The data augmentation does not affect the accuracy of AlexNet and ResNet-18, however, it improves the accuracy of VGG-16. The pre-trained models are more robust and have a better understanding of data transformations due to their extensive prior knowledge.

During the evaluation, our simple CNN, AlexNet, and VGG-16 achieve 100% accuracy when detecting CPU-related anomalies. All of the models can identify the normal runs (without any anomalies) with an accuracy of at least 92%. The misclassification errors are most common between normal runs and packet loss injected anomalies as seen in Figure 3. This interference influences the characteristics of the traces subtly and it is hard to identify by our model. More data might be needed to improve performance.

Our simple CNN network is lightweight and takes only 77.09 seconds to train on GeFore RTX 3090 GPU with 24 GB memory. The pre-trained models are able to achieve slightly higher accuracy and exhibit better robustness to data transformations. However, fine-tuning VGG-16 or ResNet-18 on high-resolution images (512 by 512 pixels) is 3 to 4 times slower (Table 3) and more resource intensive.

Based on our results, both training from scratch and transfer learning are promising methods that should be futher tested on larger and more diverse Gantt chart dataset.

6 RELATED WORK

Many statistical and deep learning based anomaly detection methods have been proposed ([16], [19], [8], [9]). In [19], the authors propose hierarchical temporal memory based method for performance anomaly detection. Du et al. [9], utilize Long Short-Term Memory (LSTM) deep learning network to identify anomalies based on data in log files. In [8], the authors develop RAMP (Real-Time Aggregated Matrix Profile) for real-time anomaly detection. RAMP uses time series data generated by events in scientific workflows and a technique called Matrix Profile to detect anomalous behaviours. Our work is the first to apply computer vision methods and identify anomalies in scientific workflow execution traces.

7 FUTURE WORK

We plan to increase the size of our dataset by collecting end-to-end execution timelines of other scientific workflows. We aim to explore more advanced deep learning methods like Convolutional Autoencoders or contrastive learning and investigate correctness of our models through explainable AI methodologies.

ACKNOWLEDGMENTS

This work was funded by DOE contract #DESC0012636M, "Panorama 360: Performance Data Capture and Analysis for End-to-end Scientific Workflows".

REFERENCES

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

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

[3] ELK Stack. https://www.elastic.co/elk-stack.

[4] 1000 Genomes Project Consortium. 2012. A global reference for human genetic variation. *Nature* 526, 7571 (2012), 68–74.

[5] Ilya Baldin, Jeff Chase, Yufeng Xin, Anirban Mandal, Paul Ruth, Claris Castillo, Victor Orlikowski, Chris Heermann, and Jonathan Mills. 2016. ExoGENI: A Multi-Domain Infrastructure-as-a-Service Testbed. In *The GENI Book*, Rick McGeer, Mark Berman, Chip Elliott, and Robert Ricci (Eds.). Springer International Publishing, 279–315.

[6] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. 2015. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* 46 (2015), 17–35. https://doi.org/10.1016/j.future.2014.10.008

[7] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[8] J. Dinal Herath, C. Bai, G. Yan, P. Yang, and S. Lu. 2019. RAMP: Real-Time Anomaly Detection in Scientific Workflows. In *2019 IEEE International Conference on Big Data (Big Data)*. 1367–1374. https://doi.org/10.1109/BigData47090.2019.9005653

[9] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 14. https://doi.org/10.1145/3133956.3134015

[10] Rafael Ferreira da Silva, Rosa Filgueira, Ewa Deelman, Erola Pairo-Castineira, Ian Michael Overton, and Malcolm Atkinson. 2019. Using Simple PID-inspired Controllers for Online Resilient Resource Management of Distributed Scientific Workflows. *Future Generation Computer Systems* 95 (2019), 615–628. https://doi.org/10.1016/j.future.2019.01.015

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. arXiv:cs.CV/1512.03385

[12] Bert Hubert, Thomas Graf, Greg Maxwell, Remco van Mook, Martijn van Oosterhout, P Schroeder, Jasper Spaans, and Pedro Larroy. 2002. Linux advanced routing & traffic control. In *Ottawa Linux Symposium*, Vol. 213.

[13] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521, 7553 (2015), 436–444. https://doi.org/10.1038/nature14539

[16] A. Mandal, P. Ruth, I. Baldin, D. Krol, G. Juve, R. Mayani, R. F. Da Silva, E. Deelman, J. Meredith, J. Vetter, V. Lynch, B. Mayer, J. Wynne, M. Blanco, C. Carothers, J. Lapre, and B. Tierney. 2016. Toward an End-to-End Framework for Modeling, Monitoring and Anomaly Detection for Scientific Workflows. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1370–1379. https://doi.org/10.1109/IPDPSW.2016.202

[17] George Papadimitriou, Cong Wang, Karan Vahi, Rafael Ferreira da Silva, Anirban Mandal, Liu Zhengchun, Rajiv Mayani, Mats Rynge, Mariam Kiran, Vickie E. Lynch, Rajkumar Kettimuthu, Ewa Deelman, Jeffrey S. Vetter, and Ian Foster. 2021. End-to-End Online Performance Data Capture and Analysis for Scientific Workflows. *Future Generation Computer Systems* 117 (2021), 387–400. https://doi.org/10.1016/j.future.2020.11.024 Funding Acknowledgments: DOE DE-SC0012636.

[18] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, Ian Foster, Rob Gardner, Mike Wilde, Alan Blatecky, John McGee, and Rob Quick. 2007. The open science grid. *Journal of Physics: Conference Series* 78 (jul 2007), 012057. https://doi.org/10.1088/1742-6596/78/1/012057

[19] Maria A. Rodriguez, Ramamohanarao Kotagiri, and Rajkumar Buyya. 2018. Detecting performance anomalies in scientific workflows using hierarchical temporal memory. *Future Generation Computer Systems* 88 (2018), 624–635. https://doi.org/10.1016/j.future.2018.05.014

[20] Mats Rynge, Karan Vahi, Ewa Deelman, Anirban Mandal, Ilya Baldin, Omkar Bhide, Randy Heiland, Von Welch, Raquel Hill, William L. Poehlman, and F. Alex Feltus. 2019. Integrity Protection for Scientific Workflow Data: Motivation and Initial Experiences. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)* (Chicago, IL, USA) *(PEARC '19)*. ACM, New York, NY, USA, Article 17, 8 pages. https://doi.org/10.1145/3332186.3332222 Funding Acknowledgments: NSF 1642070, NSF 1642053, NSF 1642090. Best Paper in Advanced Research Computing Software and Applications Track. The Phil Andrews Most Transformative Contribution Award.

[21] SciTech. Pegasus 1000 Genome Workflow. https://github.com/pegasus-isi/1000genome-workflow.

[22] SciTech. Pegasus Panorama. https://github.com/pegasus-isi/pegasus/tree/panorama.

[23] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data* 6, 1 (2019), 1–48.

[24] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:cs.CV/1409.1556

[25] Marc Snir, Robert W Wisniewski, Jacob A Abraham, Sarita V Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, Andrew A Chien, Paul Coteus, Nathan A Debardeleben, Pedro C Diniz, Christian Engelmann, Mattan Erez, Saverio Fazzari, Al Geist, Rinku Gupta, Fred Johnson, Sriram Krishnamoorthy, Sven Leyffer, Dean Liberty, Subhasish Mitra, Todd Munson, Rob Schreiber, Jon Stearley, and Eric Van Hensbergen. 2014. Addressing Failures in Exascale Computing. *Int. J. High Perform. Comput. Appl.* 28, 2 (May 2014), 129–173. https://doi.org/10.1177/1094342014522573

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958.

[27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. arXiv:cs.CV/1409.4842

[28] Douglas Thain, Todd Tannenbaum, and Miron Livny. 2005. Distributed computing in practice: the Condor experience. *Concurr. Comput.* 17, 2-4 (Feb. 2005), 323–356.

[29] 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. 2014. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering* 16, 05 (sep 2014), 62–74. https://doi.org/10.1109/MCSE.2014.80

[30] Amos Waterland. stress, POSIX workload generator.

[31] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, and Yizhou Yu. FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation. arXiv:cs.CV/1903.11816

[32] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. 2021. A Comprehensive Survey on Transfer Learning. *Proc. IEEE* 109, 1 (2021), 43–76. https://doi.org/10.1109/JPROC.2020.3004555