EPJ Web of Conferences 337, 01120 (2025)
CHEP 2024
https://doi.org/10.1051/epjconf/202533701120

Decode the Workload: Training Deep Learning Models for Efficient Compute Cluster Representation

Ahmed Hossam Mohammed[1,*], *Mark* Jones[1], *Diana* McSpadden[1], *Malachi* Schram[1,2], *Bryan* Hess[1], and *Kishansingh* Rajput[1,3]

[1]Data Science Department, Thomas Jefferson National Accelerator Facility, Newport News, Virginia, 23606, USA
[2]Computer Science Department, Old Dominion University, Norfolk, Virginia, 23539, USA
[3]Department of Computer Science, University of Houston, Houston, TX 77204, USA

Abstract. In this study, we address the mounting challenge of monitoring high throughput computing clusters running computationally intensive jobs, which increasingly strains system administrators. We develop autoencoders that analyze traces of Linux kernel CPU metrics to capture salient system features by producing robust compressed embeddings for various downstream tasks. In addition, we employ graph neural networks to incorporate contextual information from surrounding CPUs and assess their performance. We also demonstrate the enhanced job differentiation achieved by increasing the sampling rate of these traces. Our models are evaluated based on their ability to generate meaningful latent representations, detect anomalies, and distinguish between different job types, marking a preliminary step towards self-supervised, large-scale foundation models for computing centers.

1 Introduction

Increased demand for computational resources has expanded the scale of High Throughput Computing (HTC) clusters, thereby exacerbating issues related to resource contention, misconfigurations, and poor job submissions. Consequently, system administrators encounter increased difficulties in tracking cluster health (i.e., anomaly detection and prediction tasks) and intervening when necessary (i.e., control task) before the situation deteriorates further. The sheer number of components and metrics renders manual monitoring by system administrators impractical. Additionally, it is challenging to determine how the trace patterns of these metrics will evolve under different malfunction scenarios.

Inspired by the successes of foundation models that can be customized for diverse applications [1], we employ Machine Learning (ML) models to generate compact embeddings that capture the most salient input features. These embeddings underpin downstream tasks such as anomaly detection—where alerts are triggered whenever the system exhibits behavior outside its learned norm—and the training of Reinforcement Learning (RL) agents that can make informed decisions based on the model's outputs.

Many efforts have explored using AutoEncoders (AEs) to predict anomalies. For example, in [2], a separate AutoEncoder (AE) was trained on each compute node in the cluster

*e-mail: ahmedm@jlab.org

using 166 metrics to detect node-level anomalies, with the premise that anomalous behavior would disrupt inter-metric correlations. The latent vector's dimensionality was set to ten times that of the input, counter to the typical preference for lower-dimensional embeddings in hierarchical models. Additionally, the architecture did not account for the time-series nature of the data. It was demonstrated in [3] that training an AE on time-series data yielded a higher ROC-AUC (Area Under the Receiver Operating Characteristic Curve) in anomaly detection compared to a dense AE that ignores temporal correlations. The same authors explored using a node-classifying Graph Neural Network (GNN) to predict anomalies within a future time window, modeling compute nodes as graph nodes connected based on physical proximity [4].

In light of the presented previous work, we examine the trade-offs inherent in the properties of the studied time series rather than focusing solely on the performance metrics for the anomaly detection task. Owing to the uniform characteristics of the compute nodes and their consistent task assignments, the studied architectures are trained using data aggregated from all nodes—thereby eliminating the need for node-specific models and reducing training overhead. These architectures include autoencoders designed to model the core traces of compute nodes; additionally, we evaluate augmenting these autoencoders with a GNN module that facilitates information sharing across the CPUs within a compute node. The models are assessed based on their ability to capture the salient features of the input, detect anomalies, and distinguish between different kinds of jobs run at Jefferson Lab's computational cluster.

The manuscript is organized as follows: Section 2 details the data collection process and its characteristics. Section 3 describes the structure of the employed models, while section 4 reports their performance according to the established criteria. Section 5 concludes the study and outlines future research directions.

2 Data

2.1 Collection & Preparation

At Jefferson Lab, the computing cluster primarily processes Nuclear Physics (NP) experimental jobs. The Research and Development (R&D) sandbox cluster comprises five compute nodes, each equipped with 16 dual-threaded cores across two sockets; Message Passing Interface (MPI) jobs are supported for enhanced parallelization. This dedicated environment enables the execution of multiple jobs, including anomalous ones, without impacting the lab's production cluster. Table 1 outlines the various job types submitted to the sandbox cluster.

Prometheus [5], a monitoring system for time-series data, along with a custom CGroup-v2 exporter, collects Linux kernel CPU metrics ("iowait", "system", and "user") at a 1 Hz rate. Remaining metrics are aggregated into an "other" category so that the CPU rates sum to one at each timestep. Notably, no data-driven scaling was applied, as metric distributions during production inference may differ; this approach forces the ML models to prioritize metrics that vary significantly during training.

Training, validation, and test datasets are typically split randomly to ensure similar underlying distributions; however, naïve random splitting of time-series data may lead to data leakage. To prevent this, we propose running the jobs from Table 1 in campaigns, with each campaign exclusively assigned to one dataset. Since each campaign contains the same set of jobs, the resulting datasets maintain comparable distributions.

2.2 Sampling Analysis

Building upon prior research demonstrating the superiority of time-series models over single-timestep models, this study seeks to determine the optimal sequence length that captures the

Job Type	**Description**
Nuclear Physics (NP)	Likelihood fit of partial wave analysis [6]
Bad IO (omitted during training)	Synthetic job spawning a writer per thread that generates write, seek, and flush operations in a difficult-to-buffer way
Good IO	Synthetic job spawning a writer per thread, adhering to modern C conventions where file operations are managed by the operating system
MPI N	CPU-bounded MPI job (version: MPICH) running on N threads for a random amount of time before executing Allreduce
Idle	Refers to any thread not running any job

Table 1: Campaign job types submitted to the sandbox cluster

strongest temporal correlations in the data while minimizing computational costs. Unlike previous work that addressed this question within specific tasks like anomaly detection [3], this research focuses on the data itself to identify an appropriate sequence duration. Hence, Welch's Spectral Density Estimation (SDE) was employed to analyze various job waveforms, aiming to identify the lowest dominant frequency components. This method involves dividing a waveform into overlapping segments, applying Fast Fourier Transform (FFT) to each, and averaging the squared magnitudes of the FFT results [7]. Visual analysis revealed that the lowest dominant frequency was 0.01 Hz, corresponding to a 100-second period. It is important to note that this approach is data-dependent, as different jobs produce varying waveforms.

After determining the sequence duration, the following question becomes: *How many points should be fitted within that duration (i.e., sampling rate)?* Increasing the sampling fidelity by which the data is collected comes with a significant computational overhead, and hence needs to be justified. Principal Component Analysis (PCA) was applied on the waveforms of the jobs defined in Table 1 at different sampling intervals (i.e., sampling rate reciprocal) to evaluate the effect of the sampling interval on the job distinction ability. Figure 1a illustrates the clustering of traces in the PCA domain under 1 Hz sampling.

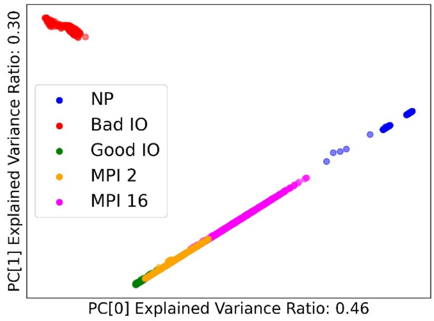
Cluster separability can be assessed by the silhouette score metric which measures how well each data point fits within its own cluster compared to the nearest other cluster. Values near 1 indicate clear separation, values around 0 suggest overlap, and negative values signal potential misassignments. The overall score is the average of individual point scores.

Due to the quadratic computational cost of calculating silhouette scores, we computed the score on a sampled subset of waveforms, repeating the process 30 times per sampling interval (i.e., an ensemble size of 30). As shown in Figure 1b, increasing the sampling rate improves the separation of job traces. Consequently, all analyses use the native 1 Hz data—its maximum available fidelity—without further down-sampling.

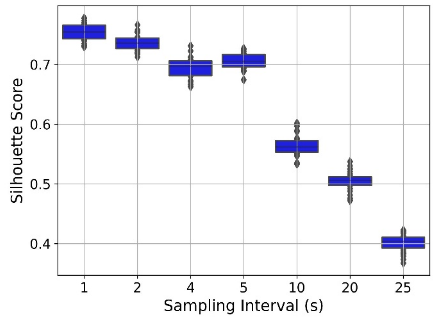
3 Models

3.1 Vanilla AutoEncoder (AE)

Leveraging the benefits of modeling temporal correlations [3], the encoder processes time-series data while omitting the redundant "other" metric to preserve feature independence. Local features are extracted via 1D convolutional layers (Conv1D) that apply learnable filters along the temporal axis, and these are downsampled using 1D max-pooling layers (MaxPool1D) to produce feature maps at different abstraction levels. The encoder comprises



(a) PCA applied on flattened traces.



(b) Better separability with higher fidelity.

Figure 1: (a) Shows the clustering of a sample from different job traces. (b) Silhouette score is computed on each of the samples. The figure shows the enhanced separability denoted by the higher silhouette score accompanied by the higher sampling rate (lower sampling interval).

stacked blocks that combine Conv1D followed by MaxPool1D, with additive residual connections [8] to enhance gradient flow and performance. The output tensor is then flattened and passed to a Multi-Layer Perceptron (MLP) to generate a compressed embedding z. The decoder roughly mirrors this architecture—replacing max-pooling with upsampling—to reconstruct the traces, with a softmax activation at the final layer across the channel dimension ensuring that the reconstructed rates sum to one at each timestep.

3.2 Graph Neural Network-Assisted AutoEncoder (GNN-AE)

Unlike the vanilla autoencoder (AE) in subsection 3.1, which processes each CPU's traces independently, we hypothesized that modeling inter-CPU interactions yields a more effective representation of salient features for multi-threaded jobs. Accordingly, we adopted the GNN-AE architecture where graph nodes represent CPU threads. In this framework, $z_i^{(k)}$ denotes the embedding of the i^{th} thread after k message-passing iterations. An encoder compresses each thread's traces to produce $z_i^{(0)}$, after which the GNN module updates the embeddings as:

$$z_i^{(1)} = f\left(z_i^{(0)}, \sum_{j \in J} \alpha_{i,j} \, z_j^{(0)}\right), \tag{1}$$

where f is a learnable function represented as an MLP, and J denotes the set of neighboring threads to thread i on the same compute node. An MLP takes $z_i^{(0)}$ and $z_j^{(0)}$ as inputs and outputs a scaling factor, $\alpha_{i,j}$, that assigns a weight to the importance of information received from neighboring thread j by thread i. We explored two variants of this architecture: **Node-GNN-AE**, modeling all CPU threads within a compute node as a graph, and **Job-GNN-AE**, representing only CPU threads executing the same job as a graph. Given that all graphs are complete, a single GNN message-passing iteration suffices for each thread to capture the entire graph's activity. This approach not only reduces GNN training and inference time but also mitigates the common issue of over-smoothing in GNNs [9]. Finally, a decoder, with an architecture similar to that in subsection 3.1, reconstructs the traces of CPU_i from $z_i^{(1)}$. Consistent with the vanilla AE architecture, the same encoder and decoder modules are utilized for all threads, hence the designation GNN-assisted AE. This architecture draws inspiration from the success of GNNs in various fields, including the study cited in [4], where GNNs facilitated information sharing across co-located compute nodes. Our interest lies in investigating the potential benefits of applying GNNs at a more granular level, specifically at the CPU level, thereby paving the way for hierarchical modeling of the compute cluster at varying resolution levels.

4 Results & Discussion

4.1 Capturing Salient Features

The primary criterion for evaluating different ML architectures is the ability to capture the salient features of the input traces which is quantified in terms of the model's ability to reconstruct the traces of the job types witnessed during the training phase (i.e., normal jobs). Figure 2 illustrates an example of how well the trained models can replicate the original input traces of the user variable collected from different CPUs within the cluster.

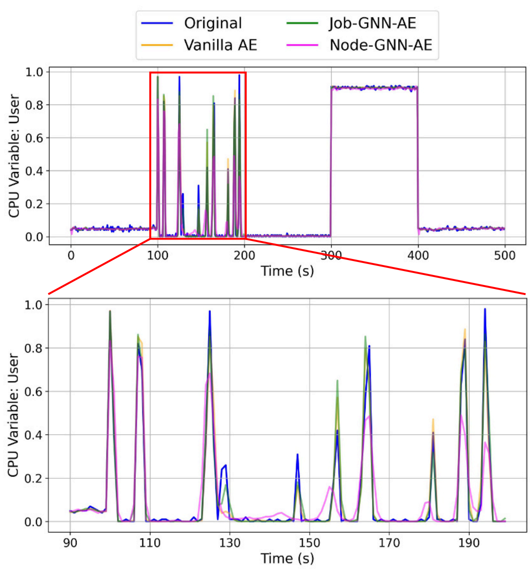


Figure 2: Original and reconstructed traces by the different models.

Table 2 presents a comparison of different architectures by evaluating the test dataset on an ensemble of randomly initialized models for each architecture. Both Vanilla AE and Job-GNN-AE achieved comparable reconstruction error on normal jobs. This suggests that incorporating a GNN block does not necessarily enhance the autoencoder's ability to capture salient features. One explanation could be that the latent representation $z_i^{(0)}$ in vanilla AE already contains sufficient information for reconstructing CPU_i's traces. Furthermore, the current sampling rate may be insufficient to adequately capture inter-thread correlations. GNNs may prove more beneficial in predictive tasks where information from neighboring CPU threads aids in forecasting relevant features. Their adaptability and scalability are advantageous, especially given the heterogeneous nature of hardware architectures and the variability in thread assignments across jobs.

	Vanilla AE	Node-GNN-AE	Job-GNN-AE
RE - Normal	0.0064 ± 0.0002	0.0122 ± 0.0033	0.0069 ± 0.0003
RE - Anomalous	0.3543 ± 0.0434	0.3642 ± 0.0065	0.3733 ± 0.0151
Silhouette Score	0.7503 ± 0.0207	0.6849 ± 0.0373	0.6694 ± 0.0232

Table 2: Reconstruction Error (RE) and Silhouette Score statistics are collected over an ensemble of 10 models for each architecture.

4.2 Anomaly Detection

We evaluated the models' ability to detect anomalies by manually injecting Bad I/O jobs, as described in Table 1. These jobs—representative of those commonly arising from inefficient coding practices that lead to resource contention—exhibit markedly different behavior from the normal jobs used for training, resulting in significantly higher reconstruction errors. Figure 3 illustrates that all three architectures detect these anomalous jobs. Although the GNN-AE architectures yield a larger margin between anomalous and normal reconstruction errors compared to the vanilla AE (as shown in Table 2), the vanilla AE still yields perfect separation between the two job types, likely because the designed anomalies are sufficiently extreme to be trivially detected by any architecture.

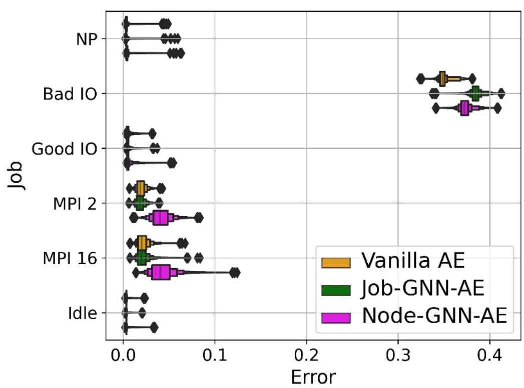
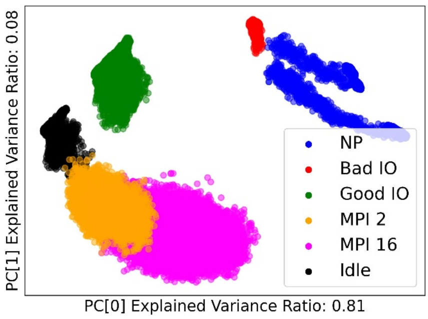
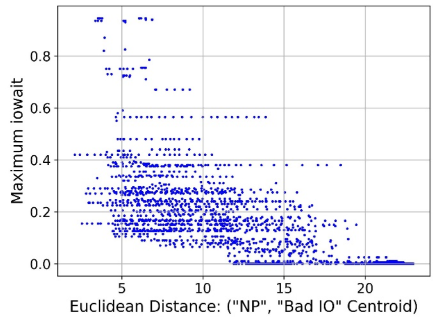


Figure 3: Reconstruction error yielded on the test dataset for the different job types. Bad IO jobs that were not included in the training set yield a significantly larger reconstruction error indicating the model's ability to identify the out-of-distribution jobs.

In our previous work [10], we introduced a continual learning framework for cluster monitoring, in which nightly retraining is performed on the previous day's data. After retraining, a daily competition selects a champion model—based on the minimal reconstruction error on normal data—to be deployed the following day for anomaly detection. A threshold, derived from the normal reconstruction error distribution, is used to flag anomalous events with minimal false positives, and these events are validated by system administrators. If an anomaly is confirmed during the day, the nightly retraining is halted to preserve the model's sensitivity to future anomalous events of the same kind.



(a) Embeddings scatter plot



(b) Maximum iowait decay

Figure 4: Vanilla AE model: (a) Shows PCA scatter plot of different job types. PCA is fitted on the training dataset (not including Bad IO jobs). Notably, some NP samples overlap with BadIO. (b) Maximum iowait of NP samples decays as the euclidean distance distance from the Bad IO centroid increases. Hence, samples lying close to Bad IO suffer from high iowait.

4.3 Separability

Besides anomaly detection, we evaluated the models' ability to generate latent embeddings that effectively separate the job types listed in Table 1. Mirroring the sampling rate analysis in section 2, we computed silhouette scores directly on the latent embeddings since all architectures yield embeddings of the same size; Table 2 indicates that the vanilla AE achieves the highest separability. PCA was then fitted on the training embeddings and applied to transform the test embeddings. Figure 4a illustrates the resulting clustering for the vanilla AE. Notably, some NP job samples appear near the Bad IO cluster in the PCA domain. To investigate this further, we plotted the Euclidean distance from each NP sample to the Bad IO centroid against the corresponding maximum iowait—the latter serving as a proxy for worst-case performance, given the observed dominance of the iowait mode in Bad IO jobs. As shown in Figure 4b, maximum iowait decreases with increasing distance from the Bad IO centroid. This two-dimensional diagnostic tool thus enables the identification of problematic NP samples and could be used to trigger alarms when new samples fall within regions previously associated with anomalous behavior.

5 Conclusion & Future Work

This study presents a framework for monitoring computational clusters. Initially, we demonstrated that increasing the sampling fidelity of Linux CPU performance metric traces improves the separability of different job types. Subsequently, various autoencoder-based architectures were compared in terms of their ability to represent cluster status, detect anomalies, and distinguish job types in the latent space. Despite the larger reconstruction error margins between anomalous and normal samples yielded by GNN-assisted architectures, the vanilla autoencoder more effectively captures salient features and differentiates job samples while still maintaining perfect separation between normal and anomalous jobs. Thus, for the tasks examined, the additional complexity of GNN-aided models does not improve performance.

We aim to investigate settings in which the GNN can yield significant improvements, such as predicting critical events (e.g., job failure) at a sampling rate that captures the tight coupling between threads. Potential extensions include developing hierarchical foundation models that integrate heterogeneous components (e.g., computing, storage, networking). For instance, one could generate a compressed latent representation of a compute node's status using a foundation model, and then use this representation as input to a higher-level model for the entire cluster. Such architectures could facilitate multi-resolution cluster monitoring and enable control mechanisms—such as automatic job termination—to optimize a reward function that balances cluster health with throughput.

6 Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-AC05-06OR23177. The research described in this paper was conducted under the Laboratory Directed Research and Development program at Thomas Jefferson National Accelerator Facility for the U.S. Department of Energy.

References

[1] J. Zhu, Y. Xia, L. Wu, D. He, T. Qin, W. Zhou, H. Li, T. Liu, Incorporating BERT into Neural Machine Translation, CoRR **abs/2002.06823** (2020), `2002.06823`.

[2] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, Anomaly detection using autoencoders in high performance computing systems, in *Proceedings of the AAAI Conference on artificial intelligence* (2019), Vol. 33, pp. 9428–9433

[3] M. Molan, A. Borghesi, D. Cesarini, L. Benini, A. Bartolini, RUAD: Unsupervised anomaly detection in HPC systems, Future Generation Computer Systems **141**, 542 (2023). https://doi.org/10.1016/j.future.2022.12.001

[4] M. Molan, J. Ahmed Khan, A. Borghesi, A. Bartolini, Graph Neural Networks for Anomaly Anticipation in HPC Systems, in *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering* (Association for Computing Machinery, New York, NY, USA, 2023), ICPE '23 Companion, p. 239–244, ISBN 9798400700729, `https://doi.org/10.1145/3578245.3585335`

[5] B. Rabenstein, J. Volz, Prometheus: A Next-Generation Monitoring System (Talk) (USENIX Association, Dublin, 2015)

[6] M. Jones, P. Hurck, W. Phelps, C.W. Salgado, PyPWA: A software toolkit for parameter optimization and amplitude analysis, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **1062**, 169150 (2024). https://doi.org/10.1016/j.nima.2024.169150

[7] P. Welch, The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms, IEEE Transactions on Audio and Electroacoustics **15**, 70 (1967). 10.1109/TAU.1967.1161901

[8] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016)

[9] T.K. Rusch, M.M. Bronstein, S. Mishra, A Survey on Oversmoothing in Graph Neural Networks (2023), `2303.10993`, `https://arxiv.org/abs/2303.10993`

[10] D. McSpadden, M. Jones, A.H. Mohammed, B. Hess, M. Schram, Establishing MLOps for Continual Learning in Computing Clusters, IEEE Software pp. 1–8 (2024). 10.1109/MS.2024.3424256