Refine: A Robust Approach to Unsupervised Anomaly Detection for Production HPC Systems

Efe Sencan
Boston University
Boston, MA, USA
esencan@bu.edu

Yin-Ching Lee
Boston University
Boston, MA, USA
leewill@bu.edu

Connor Casey
Boston University
Boston, MA, USA
ccasey@bu.edu

Benjamin Schwaller
Sandia National Laboratories
Albuquerque, NM, USA
bschwal@sandia.gov

Vitus J. Leung
Sandia National Laboratories
Albuquerque, NM, USA
vjleung@sandia.gov

Jim Brandt
Sandia National Laboratories
Albuquerque, NM, USA
brandt@sandia.gov

Brian Kulis
Boston University
Boston, MA, USA
bkulis@bu.edu

Manuel Egele
Boston University
Boston, MA, USA
megele@bu.edu

Ayse K. Coskun
Boston University
Boston, MA, USA
acoskun@bu.edu

***Abstract*—High-Performance Computing (HPC) systems are critical for many scientific applications, but they are often subject to performance variations due to ”anomalies”, which can lead to reduced efficiency and higher operational costs. To address this, machine learning (ML) techniques have been increasingly applied to automatically detect performance anomalies. However, traditional unsupervised anomaly detection methods assume that training datasets are free of anomalies. In real-world HPC systems, though, data is typically contaminated by anomalies caused by factors such as shared resource contention, software bugs, or hardware failures. These anomalies in the training data can significantly undermine the performance of ML models. To overcome this issue, we introduce *Refine*, a robust anomaly detection framework based on variational autoencoders (VAEs). *Refine* iteratively removes high-error samples during training in an unsupervised manner. By gradually reducing the proportion of anomalies in the training dataset based on reconstruction error, our approach enhances the model's robustness and overall performance. We evaluate *Refine* using data collected from a production HPC system, Eclipse, and demonstrate its effectiveness in handling varying levels of contamination. Even with up to 10% anomalies in the training dataset, *Refine* achieves an F1-score of 0.88, outperforming state-of-the-art unsupervised anomaly detection methods. Moreover, when applied to real production system data, *Refine* achieves 100% accuracy in detecting anomalies.**

***Index Terms*—High Performance Computing, Machine Learning, Anomaly Detection**

I. Introduction

High-Performance Computing (HPC) systems play a crucial role in advancing scientific and engineering research, powering applications such as drug discovery, climate modeling, and nuclear physics. These systems, comprising thousands of compute nodes, offer immense computational power to tackle complex problems. However, the advent of exascale computing has introduced a higher degree of heterogeneity and complexity into the HPC infrastructure [1]. This increased complexity, coupled with a higher degree of resource sharing, has made HPC systems more susceptible to performance variations.

Our repository is public at: https://github.com/PEACLab/Refine

Performance variations in HPC systems can be attributed to a range of factors, including hardware-related issues [2], shared resource contention [3], network congestion [4], or memory problems [5] such as memory leaks. While resource contention itself is expected in HPC systems, its severity can vary widely. In this work, we classify contention-related performance degradations as anomalies because, despite not being indicative of bugs or hardware failures, these degradations can substantially and unexpectedly affect application runtimes, reducing computational efficiency and increasing operational costs [2]. Subtle and transient anomalies of this kind are especially challenging to identify as they degrade system efficiency without causing outright failures or crashes. In this work, we focus on detecting anomalies that cause performance variations but do not necessarily lead to system failures or crashes.

Modern HPC systems leverage monitoring frameworks to gather extensive telemetry data, including performance metrics, system logs, and traces. However, the sheer volume of data generated by these systems—often reaching billions of data points daily—presents a significant challenge. Manual analysis of this data is impractical and prone to errors, making automated performance analysis techniques essential. Machine learning (ML) has emerged as a promising solution for anomaly detection and diagnosis, enabling system administrators to identify and mitigate performance issues more efficiently. Specifically, researchers have developed several techniques using supervised learning to detect and diagnose anomalies in HPC systems [6], [7]. These methods train models on labeled data to accurately identify performance issues. However, the telemetry data collected from monitoring systems is vastly unlabeled, and obtaining labels from domain experts is time-consuming, costly, and often infeasible. This lack of labeled data significantly limits the applicability of supervised learning approaches in real-world HPC environments. To relax the labeled data requirements, prior works have applied semi-supervised learning methods [8], [9], which use a combination of labeled and unlabeled data to improve model performance. These methods leverage a small amount of labeled data to guide the learning process, but they still face challenges when labeled data is scarce.

To address the lack of labeled data problem, recent works [10], [11], utilize *unsupervised learning* approaches. These existing unsupervised approaches either focus on detecting node failures or system crashes, which do not address the more challenging task of detecting performance anomalies, or they assume the training dataset is anomaly-free, which limits their usability in production HPC systems as real-world data is often not completely anomaly-free. To address the aforementioned problems, this paper introduces a novel robust variational autoencoder (VAE) [12] based training strategy that detects performance anomalies in compute nodes with high accuracy even when the training dataset is contaminated with anomalies. The *key novelty of our method* is its iterative data removal strategy, which progressively refines the training dataset by exploiting the difference in reconstruction errors between healthy and anomalous samples. This difference is most pronounced during the early stages of VAE training. By iteratively removing high-error samples, we reduce the anomaly ratio (number of anomalous samples divided by the total number of samples in the dataset) in the training dataset without requiring label information. This process enables the model to learn a more accurate representation of normal (healthy) system behavior, thereby improving its ability to distinguish between healthy and anomalous samples. Figure 1 provides an overview of *Refine* pipeline. Our specific contributions are as follows:

- Design of *Refine*, a robust unsupervised anomaly detection framework that detects performance anomalies under data contamination.
- Demonstration of our framework using real and proxy HPC applications. Our framework achieves an 0.88 F1-score for classifying synthetic anomalies, even when the training dataset initially has a 10% anomaly ratio.
- Application of our framework to real-world production system data. Our framework demonstrates significant accuracy improvement, achieving 100% accuracy in identifying real-world performance anomalies under data contamination.

II. Related Work

A. Unsupervised Anomaly Detection in HPC Systems

State-of-the-art methods for unsupervised anomaly detection in the HPC domain predominantly adopt reconstruction error-based approaches. RUAD [10] utilizes an LSTM-based autoencoder to detect system faults, training separate models for each compute node on raw time series data. However, maintaining separate models for each compute node may not be feasible as the system scales to thousands of nodes. Prodigy [11] uses a VAE-based framework to detect performance anomalies caused by shared resource contention. However, Prodigy assumes that the training dataset is anomaly-free, which may not hold in real-world scenarios.

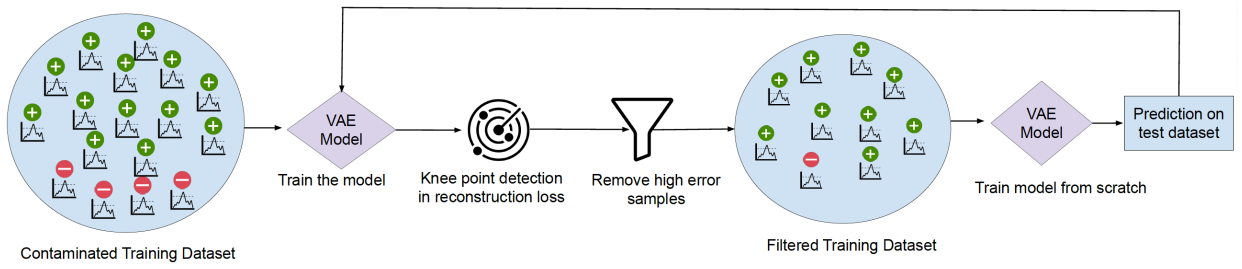
B. Unsupervised Anomaly Detection with Contaminated Training Dataset

In practice, it is unrealistic to assume that training datasets are entirely free of anomalies. Even a small proportion of anomalous samples in the training dataset can significantly degrade the performance of unsupervised anomaly detection models. Therefore, researchers have developed various methods to handle contaminated training dataset in the context of multivariate time series anomaly detection, which can be broadly categorized into three types:

The first category modifies reconstruction-based methods to enhance robustness. For instance, Normality-Calibrated Autoencoder (NCAE) [13] leverages a joint generative adversarial network (GAN) [14] to adversarially generate high-confidence normal samples from a low-entropy latent space. These samples serve as references to identify anomalous data and mitigate their impact by maximizing their reconstruction errors. Similarly, NegCo [15] employs a GAN to generate representative normal samples from a Gaussian distribution in the latent space. Instead of directly suppressing anomalous samples, NegCo uses these representative samples to model the negative correlation between morphological similarity and reconstruction consistency, effectively recalibrating biased anomaly measurements in contaminated datasets. This soft contamination calibration strategy significantly improves performance, especially in heavily contaminated datasets, without relying on hard thresholds. RDSSM [16] addresses the challenge of contaminated training dataset in multivariate time-series by combining robust statistics with deep state space modeling. It introduces a novel decoder structure employing a Gamma prior and t-distribution, enabling robust training and adaptive anomaly scoring even in the presence of anomalies.

Moving beyond reconstruction-based methods, some techniques use more robust base models and methods to account for contamination. COUTA [17] employs Deep Support Vector Data Description [18], which is less affected by contamination, and incorporates synthetic anomaly injection and uncertainty modeling to assess anomaly likelihood, reducing the weight of likely contaminated samples. Latent Outlier Exposure (LOE) [19] employs two coupled loss functions: one optimized for normal samples and the other for anomalous ones. By jointly learning model parameters and inferring binary labels (normal vs. anomalous) for the training data, LOE alternates between these updates using block coordinate descent. This approach not only learns to identify anomalies but also uses insights from anomalous data to improve the model's representation of normal behavior. MTGFlow [20] builds on the assumption that anomalous samples exhibit sparser densities compared to normal ones and employs dynamic graph structure learning to model evolving interdependencies among entities. Additionally, MTGFlow incorporates entity-aware normalizing flows to estimate fine-grained, entity-specific density distributions.

The final category adds enhancements to existing models for better performance with contaminated data. The IAD [21]



Contaminated Training Dataset
VAE Model
Train the model
Knee point detection in reconstruction loss
Remove high error samples
Filtered Training Dataset
VAE Model
Train model from scratch
Prediction on test dataset

Fig. 1: Overview of the *Refine* pipeline. The process begins with a contaminated training dataset and analyzes reconstruction errors during training to identify the knee point, the epoch where the model starts to overfit anomalous samples. *Refine* removes high-error samples based on the 95th percentile threshold of reconstruction errors calculated at this knee point. The framework then retrains the model on the refined dataset and uses the trained model to predict anomalies in the test dataset, applying a threshold derived from the 99th percentile of reconstruction errors in the refined training data. This iterative process repeats until the framework achieves the desired performance or reaches a predefined iteration limit.

framework introduces an iterative learning process for anomaly detection that dynamically adjusts importance weights for each sample based on their likelihood of being normal, refined at every iteration using anomaly scores. This approach, which is model-agnostic and robust to varying contamination ratios, avoids reliance on hyperparameter-sensitive pseudo-labeling. By employing a novel termination criterion that tracks convergence in anomaly score rankings, IAD effectively enhances the robustness of base models, achieving improved performance across diverse datasets. The authors introduce a novel approach, RejEx [22], that leverages a stability-based confidence metric to identify and reject low-confidence predictions, enabling robust decision-making without the need for labeled data.This method enhances the reliability of predictions by systematically filtering out uncertain outputs, thereby improving the overall performance of anomaly detection systems.

Our work differs from prior approaches by implementing an iterative filtering mechanism that removes high-error samples from the training set in an unsupervised manner, without making any assumptions about the anomaly ratio in the training dataset. Additionally, our method detects performance anomalies in HPC applications arising from shared resource contention, which are often challenging to identify.

III. Background

This section provides background on VAEs and their role in the *Refine* framework, which leverages reconstruction error properties of VAEs to detect performance anomalies in HPC systems.

A VAE is a generative model that encodes input data into a latent space and decodes it back to the original space. It consists of two main components: an encoder and a decoder. The encoder compresses the input $\mathbf{x}$ into a lower-dimensional latent representation, producing the mean $\mathbf{z_{mean}}$ and log variance $\mathbf{z_{log\_var}}$ of the latent variables. The decoder reconstructs the original input from the latent variables $\mathbf{z}$, sampled using the reparameterization trick:

$$\mathbf{z} = \mathbf{z_{mean}} + \exp(0.5 \times \mathbf{z_{log\_var}}) \times \epsilon, \quad \epsilon \sim \mathcal{N}(0,1). \tag{1}$$

The reparameterization trick plays a crucial role by reformulating the stochastic sampling operation as a differentiable transformation, enabling gradient-based optimization and allowing us to train the VAE end-to-end.

We train the VAE to minimize a combined loss function that includes a reconstruction loss and a Kullback-Leibler (KL) [23] divergence loss (L_{KL}). The reconstruction loss (L_{recon}) measures how well the decoder reconstructs the input data and is given by:

$$L_{recon} = \sum_{i=1}^{N} (\mathbf{x}_i - \hat{\mathbf{x}}_i)^2, \tag{2}$$

where N is the number of data points, $\mathbf{x}_i$ represents the input data, and $\hat{\mathbf{x}}_i$ represents the reconstructed output.

The L_{KL} regularizes the latent distribution to be close to a standard normal distribution:

$$L_{KL} = -0.5 \sum_{j=1}^{D} \left(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right), \tag{3}$$

where D represents the dimensionality of the latent space, μ_j is the mean, and σ_j is the standard deviation of the latent variables.

The total loss function is:

$$L_{total} = L_{recon} + \beta L_{KL}, \tag{4}$$

where β is a hyperparameter that controls the trade-off between reconstruction fidelity and the smoothness of the latent space.

In our framework, *Refine*, the goal of the VAE is to learn the characteristics of healthy application runs. The latent space encodes patterns that capture underlying normal behavior. When the VAE processes anomalous samples, their deviations from the learned normal patterns result in higher reconstruction errors. We use this property to detect anomalies. Specifically, we compute the reconstruction error for each sample as the Mean Absolute Error (MAE) between the input x and the

TABLE I: The list of applications we run on Eclipse and Volta for data collection.

Eclipse		
	Application	**Description**
Real Applications	LAMMPS	Molecular dynamics
	HACC	Cosmological simulation
	SW4	Seismic modeling
ECP Proxy Suite	EXAMINIMD	Molecular dynamics
	SWFFT	3D Fast Fourier Transform
	SW4LITE	Numerical kernel optimizations
Volta		
	Application	**Description**
NAS	BT	Block tri-diagonal solver
	CG	Conjugate gradient
	FT	3D Fast Fourier Transform
	LU	Gauss-Seidel solver
	MG	Multi-grid on meshes
	SP	Scalar penta-diagonal solver
Mantevo	MINIMD	Molecular dynamics
	COMD	Molecular dynamics
	MINIGHOST	Partial differential equations
	MINIAMR	Stencil calculation
Other	KRIPKE	Particle transport

reconstructed output $\hat{x}$ using it to identify anomalous samples based on a defined threshold.:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|x_i - \hat{x}_i| \,. \tag{5}$$

IV. EXPERIMENTAL METHODOLOGY

The goal of our experiments is to evaluate the robustness and effectiveness of *Refine* in detecting performance anomalies in HPC systems. We achieve this by running diverse HPC applications on two systems, Eclipse and Volta, collecting telemetry data during their execution, injecting synthetic anomalies to mimic real-world conditions, and assessing the performance of *Refine* in detecting these anomalies. This section provides details about the HPC systems, applications, injected anomalies, and the telemetry data collection process.

A. HPC System & Applications

We use two different HPC systems, Eclipse and Volta, to run various HPC applications, from which we gather the telemetry data necessary for our framework. Eclipse is a production HPC cluster consisting of 1,488 compute nodes and 53,568 cores, located at Sandia National Laboratories. It has a peak performance of 1.800 petaflops. Volta, also from Sandia National Laboratories, is a Cray XC30m testbed supercomputer with 52 compute nodes. It features 52 compute nodes organized across 13 interconnected switches, with each switch supporting four nodes. We run 6 applications on the Eclipse system and 11 applications on the Volta system. Table I lists the applications we use and the respective HPC systems they run on. On Eclipse, we run three real-world applications and three from the Exascale Computing Project (ECP) Proxy Suite. These applications are representative of production workloads, ranging from molecular dynamics to seismic and numerical simulations. On Volta, we run six benchmarks from the NAS Parallel Benchmark Suite, four applications from the Mantevo mini-app suite, and one particle transport application (*Kripke*). These benchmarks and mini-apps are widely used for testing and optimizing HPC system performance. We execute each application on 4, 8, and 16 compute nodes, using 3 different input decks for each configuration. The runtime for each application ranges from 20 to 40 minutes, depending on the application and the specific input deck. After running the applications and injecting synthetic anomalies—by executing anomaly-generating code on the application's nodes to simulate contention—we collect 24,566 samples (6,325 healthy) from the Eclipse system and 20,915 samples (18,980 healthy) from the Volta system. To create our train-test split, we divide the collected samples into a 20-80% ratio, where 20% of the samples are allocated for training and 80% for testing. Each sample represents the feature-extracted telemetry data of a single application run on a specific compute node. We repeat this train-test split process five times using stratified sampling to ensure that the class distribution in the training and test datasets matches the distribution in the entire dataset. Repeating this process exposes the model to different portions of the data during training and testing, providing a more comprehensive evaluation of its performance. We allocate 20% of the data for training due to the high proportion of anomalous samples in the Eclipse dataset. This allocation ensures a smaller training dataset, maintaining an initial anomaly ratio of 10% in the training dataset. We choose this ratio based on observed outlier behavior in Eclipse, where anomalies naturally occur at rates ranging from 2% to 7%. For experiments involving lower initial anomaly ratios (e.g., below 10%) in the training dataset, we reduce anomalous samples from the training dataset in a stratified manner, preserving the proportional class distribution from the original 10% setup.

B. Synthetic Anomalies

Obtaining a large dataset of ground truth labels (e.g., thousands of labels of known anomalies) for real-world anomalies is challenging; therefore, we use the High Performance Anomaly Suite (HPAS) [24] to generate labeled data for our controlled experiments. HPAS is an open-source framework that replicates common performance anomalies across key subsystems, including the CPU, cache, memory, network, and shared storage. While executing an application on multiple compute nodes, we inject synthetic anomalies into every node the application uses for the Eclipse dataset. In the Volta dataset, we inject anomalies into only one compute node per application run. If an anomaly is injected, the telemetry data collected from that node is labeled with the anomaly type; otherwise, it is labeled as healthy. The specific anomalies and their configurations are detailed in Table II. Each anomaly type is configured with parameters that simulate performance issues:

- CPU Contention (cpuoccupy): Simulates high CPU utilization by occupying a specified percentage of a core's capacity. This anomaly mimics scenarios where processes

TABLE II: The configuration details of the injected anomalies.

Anomaly type	Configuration
cpuoccupy	-u 100%, 80%
cachecopy	-c L1,-m 1 / -c L2 -m 2
membw	-s 4K, 8K, 32K
memleak	-s 1M, -p 0.2 / -s 3M -p 0.4 / -s 10M -p 1

compete for CPU resources, leading to reduced performance.

  - Utilization (-u): Specifies the percentage of CPU capacity to be occupied, with values set to 100% and 80%. This simulates situations where the CPU is either fully saturated or under significant load, common in HPC environments with computationally intensive tasks.

- Cache Contention (cachecopy): Generates cache read and write operations to induce contention at different cache levels.
  - Cache Level (-c): Indicates the targeted cache level, either L1 or L2.
  - Multiplier (-m): Determines the intensity of cache operations. A higher multiplier (e.g., 2) results in more intense operations compared to a lower multiplier (e.g., 1).
- Memory Bandwidth Contention (membw): Creates memory write operations to stress memory bandwidth, simulating scenarios where memory-bound applications experience bottlenecks.
  - Buffer Size (-s): Specifies the size of the memory buffer used in operations, with values set to 4 KB, 8 KB, and 32 KB.
- Memory Leak (memleak): Simulates a memory leak by continuously allocating and filling memory without releasing it.
  - Buffer Size (-s): Defines the size of each memory allocation, set to 1 MB, 3 MB, and 10 MB. Larger buffer sizes simulate more aggressive memory consumption.
  - Allocation Period (-p): Specifies the time interval between successive memory allocations, with values set to 0.2 seconds, 0.4 seconds, and 1 second. Shorter intervals simulate faster memory leaks, where resources are consumed at an accelerated rate.

We select these configurations to represent a range of performance anomalies that occur in HPC environments, allowing us to evaluate the robustness and effectiveness of our framework under various stress conditions.

C. Monitoring Framework

We use the Lightweight Distributed Metric Service (LDMS) [25] to collect telemetry data from various subsystems and performance counters on our compute nodes with minimal overhead and 1Hz sampling frequency. The collected telemetry from each application run on a given compute node is in the form of numeric multivariate time series data, represented as $X \in \mathbb{R}^{t \times m}$, where t denotes the time dimension and m represents the number of collected metrics. We collect 806 metrics from the Eclipse system and 721 metrics from the Volta system. These metrics include memory usage (e.g., free, active memory), CPU activity (e.g., user and idle time), network traffic (e.g., number of received/transmitted packets), shared file system operations (e.g., open, close, and write counts), and Cray performance counters (e.g., power consumption, write-back counters). Due to significant fluctuations in per-core metrics for the same application run, we exclude them and retain only node-level CPU metrics along with other sampled metrics, resulting in a total of 156 metrics in our framework.

Algorithm 1 Iterative Robust VAE Training

```
Input: Contaminated Dataset $\mathcal{D}$, VAE model parameters
Output: Trained VAE model
  Initialize VAE model parameters.
  Define hyperparameters:
    $\epsilon = 95^{\text{th}}$ percentile threshold for removing high-error samples from the training set.
    $N_{\text{epochs}}$ = Number of epochs for each training cycle.
    $\alpha$ = Desired test accuracy.
    $N_{\text{max\_iter}}$ = Maximum number of iterations.
  Initialize iteration counter $t \leftarrow 0$.
  while $t < N_{\text{max\_iter}}$ and test accuracy $< \alpha$ do
    Train VAE model for $N_{\text{epochs}}$ epochs on dataset $\mathcal{D}$.
    Calculate reconstruction errors for all samples in $\mathcal{D}$.
    Determine knee point $N_{\text{knee}}$ of the reconstruction loss curve.
    Train VAE model from scratch up to $N_{\text{knee}}$ epochs on dataset $\mathcal{D}$.
    Remove samples from the training set that have reconstruction errors higher than $\epsilon^{\text{th}}$ percentile of reconstruction errors in $\mathcal{D}$.
    Train VAE model from scratch for $N_{\text{epochs}}$ epochs on the filtered dataset $\mathcal{D}$.
    Make predictions on the test dataset and evaluate accuracy.
    $t \leftarrow t + 1$
  end while
  return Trained VAE model.
```

V. Robust Unsupervised Anomaly Detection

Our primary objective is to accurately detect performance anomalies in HPC systems, even when the training data is contaminated. We focus on identifying whether applications running on compute nodes are healthy or anomalous, particularly aiming to detect anomalies that cause performance variability without resulting in program errors or premature termination. We design our robust VAE training framework to enhance anomaly detection accuracy despite the challenges posed by contaminated training datasets.

The framework includes several stages: feature extraction, feature selection, feature scaling, unsupervised training, and

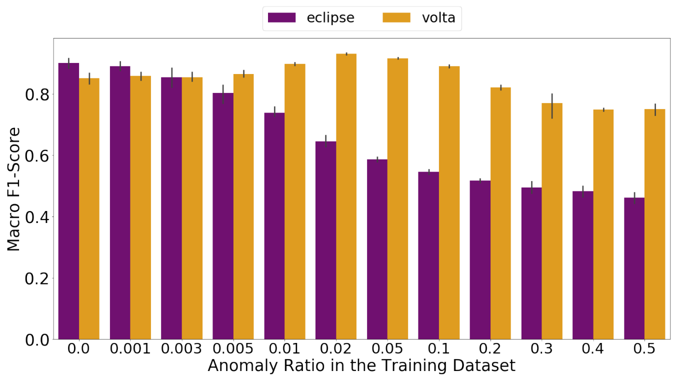


Fig. 2: The effect of anomalous samples on test dataset performance with Prodigy. When the anomaly ratio in the training dataset increases to 10%, the F1-score in the Eclipse dataset drops from 0.90 to below 0.55.

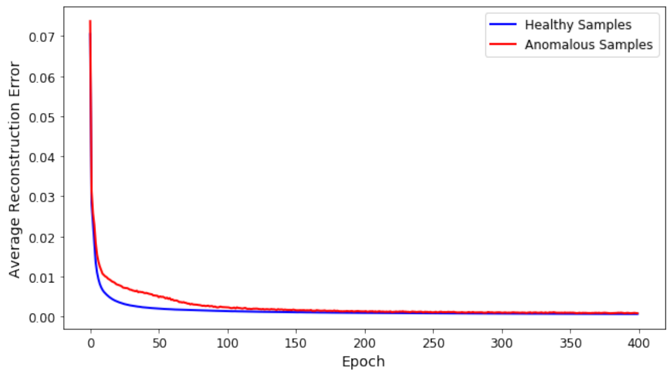


Fig. 3: Average reconstruction errors of healthy and anomalous samples during the VAE training stage in the Eclipse dataset, with a 1% anomaly ratio in the training dataset.

anomaly detection. We collect telemetry data from compute nodes running both normal and anomalous workloads. During the feature extraction stage, we generate a comprehensive set of features from the raw telemetry data. In the feature selection stage, we identify the most discriminative features to reduce dimensionality and improve classification performance. Feature scaling stage ensures that all features are normalized to a consistent range, which helps improve the model's training stability. The unsupervised training stage involves training a VAE using the selected features. To enhance robustness, we iteratively identify and remove samples with high reconstruction errors, retrain the model, and repeat the process until we achieve the desired anomaly detection accuracy on the test dataset or reach a predefined number of iterations.

A. Feature Extraction

In order to obtain different time series characteristics, we apply feature extraction using the TSFRESH [26] open-source toolkit. TSFRESH computes 794 features for each metric based on 63 distinct time series characterization methods. These extracted features provide a detailed description of the time series, often revealing underlying patterns and behaviors that are not immediately apparent. The features include basic descriptive statistics (e.g., min, max, mean) as well as more advanced measures such as absolute energy of time series, Benford correlation [27], and the variation coefficient.

B. Feature Selection & Scaling

To reduce the dimensionality of the dataset and improve classification performance, we use the Chi-square [28] feature selection method. During the feature selection process, we use one labeled sample for each application and anomaly (and healthy) pair, resulting in 30 labeled samples (both healthy and anomalous) for the Eclipse dataset and 55 labeled samples for the Volta dataset, thus requiring minimal supervision. We experiment with different numbers of features (250, 500, 1000, 2000, and 4000) and find that our model performs best with the top 2000 features. After feature selection, we apply a MinMax scaler to normalize the feature values within the range of 0 and 1. We first fit the scaler on the training dataset and then apply the same scaler to the test dataset.

C. Iterative Robust VAE Training

When the training dataset is anomaly-free, Prodigy achieves an F1-score of over 0.80 in both the Eclipse and Volta datasets. To ensure a fair comparison in Figure 2, we adjust both the Eclipse and Volta test datasets to have a consistent 10% anomaly ratio by performing stratified sampling during the train-test split. However, as the anomaly ratio in the training dataset increases, performance in the Eclipse dataset declines significantly, with macro average F1-scores dropping below 55%. We further explore the impact of higher anomaly ratios in the test datasets on model performance with varying anomaly ratios in the training dataset. Notably, when the anomaly ratio in the test dataset reaches 90%, the macro average F1-score in the Eclipse dataset falls below 20%, and in the Volta dataset, it drops below 50% when the anomaly ratio in the training dataset is 10%. In contrast, when the training datasets are anomaly-free, both datasets maintain an F1-score above 0.80, regardless of the anomaly ratio in the test dataset.

To address the performance degradation caused by anomalies in the training dataset, we introduce an iterative filtering mechanism that progressively refines the training dataset by removing samples with high reconstruction errors to mitigate the negative impact of anomalous samples. During the VAE training process, we analyze the reconstruction errors for healthy and anomalous samples at each epoch to understand their behavior. As shown in Figure 3, anomalous samples exhibit higher reconstruction errors compared to healthy samples, especially in the early training stages. However, as training progresses, the model starts to overfit these anomalous samples, and the difference in reconstruction errors begins to diminish. Leveraging this insight, we identify the knee point on the overall reconstruction error curve, which includes all training samples, indicating the epoch where the model stops learning effectively and begins to overfit. While the knee point might not perfectly align with the exact moment when the

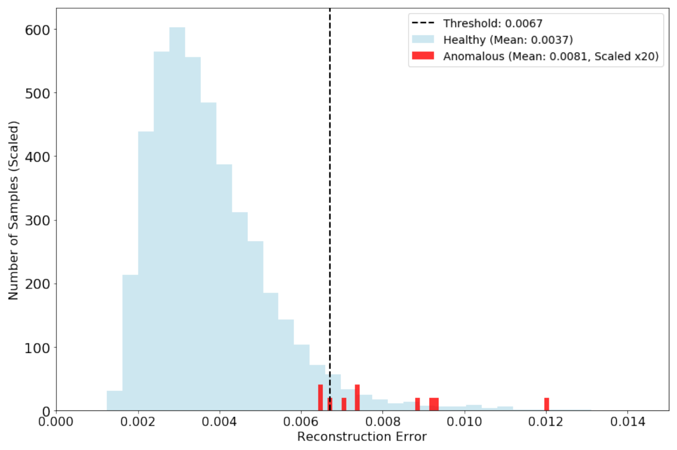


Fig. 4: Reconstruction error distribution at epoch 20 (knee point) in the Eclipse dataset with a 1% anomaly ratio in the training dataset. To enhance visual clarity, the plot scales the number of anomalous samples by a factor of 20.

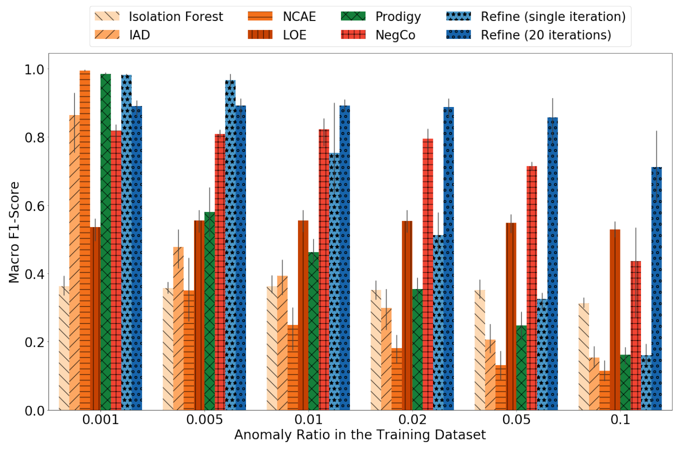


Fig. 5: Performance of different methods in the Eclipse dataset. *Refine* outperforms other baselines when the anomaly ratio in the training dataset is above 0.005.

difference in average reconstruction errors between healthy and anomalous samples diminishes, it reliably indicates when the model starts to overfit anomalous samples. For instance, in this case, we find the knee point to occur at epoch 20, which aligns with the observed behavior in the individual curves of healthy and anomalous samples. As detailed in Algorithm 1, we then retrain the VAE from scratch up to this knee point and subsequently compute the 95th percentile of reconstruction errors in the training dataset. We then remove samples with reconstruction errors exceeding this threshold from the training dataset.

Figure 4 illustrates the reconstruction error distribution at epoch 20 (knee point) in the Eclipse dataset with a 1% anomaly ratio in the training dataset. Since anomalous samples are fewer in number, we scale their count by a factor of 20 to make their bar heights more prominent for visual clarity. The dashed line represents the threshold for removing high-error samples, calculated as the 95th percentile of average reconstruction errors in the training dataset. At this epoch, the average reconstruction error for anomalous samples is 0.0081, while the average reconstruction error for healthy samples is 0.0037. The figure demonstrates that, after just one iteration, the framework can successfully remove most anomalous samples from the training dataset, although some healthy samples are also eliminated during this process. We determine the 95th percentile threshold empirically to balance the removal of anomalous data while retaining sufficient healthy samples for effective learning. We then retrain the VAE model with the refined training dataset and make predictions on the test dataset. To determine whether the sample in the test dataset is healthy or anomalous, we first make predictions on the available training dataset and set the threshold as the 99th percentile of the reconstruction errors from the training dataset. We establish this threshold based on empirical evaluation to maximize anomaly detection performance. If the reconstruction error of a test sample exceeds this threshold, we classify the sample as anomalous; otherwise, we classify it as healthy. This process of training, identifying, and removing high-error samples followed by making predictions on the test dataset constitutes a single iteration. *Refine* repeats this process until it reaches the desired test accuracy (if labeled samples are available for evaluation) or meets the predefined iteration limit.

D. Implementation Details

We use Python 3.6.x and TensorFlow 2.6.2 for our VAE implementation with a fixed random seed (`np.random.seed(42)`) for reproducibility. The encoder consists of an input layer matching the dimensionality of the feature set, followed by a dense layer that reduces the input dimensionality by half using ReLU activation. The encoder maps the input to a latent space with dimensionality set to one-third of the input dimensions. The decoder mirrors the encoder structure, taking the latent variables as input, passing them through a dense layer with ReLU activation, and reconstructing the input using a sigmoid activation in the output layer. This ensures that the reconstructed output remains within the normalized range of the input data. We use the Adam optimizer with a learning rate of 0.0001 and a gradient clipping value of 0.5 to stabilize training. We train the model for 400 epochs with a batch size of 256. We use the `KneeLocator` from the `kneed` [29] library for knee point detection.

VI. Evaluation

We evaluate the performance of our framework using the macro average F1-score, macro average precision, and macro average recall. These metrics are particularly useful for imbalanced datasets, where the distribution of class samples is skewed. Macro average precision is calculated by determining the precision for each class individually and then averaging the results. Similarly, macro average recall computes the recall

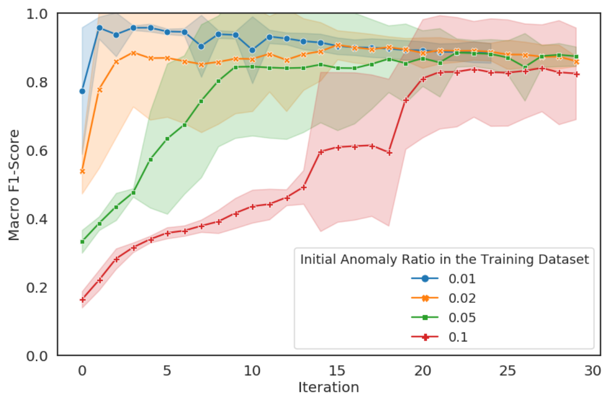
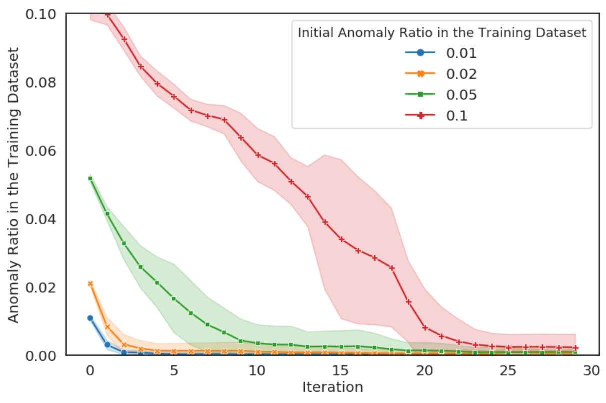
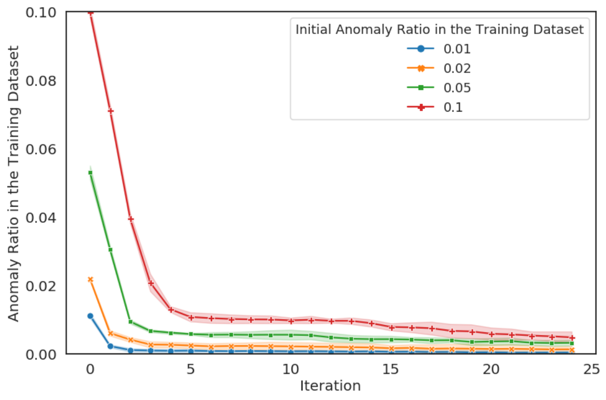


Fig. 6: Performance of *Refine* across multiple iterations increases rapidly from 0.19 to 0.88 when the initial anomaly ratio in the training dataset is 10% for the Eclipse dataset.



(a) Eclipse dataset



(b) Volta dataset

Fig. 7: *Refine* removes anomalies in the training dataset over iterations (see Algorithm 1). (a) Eclipse dataset: The anomaly ratio decreases gradually over multiple iterations. (b) Volta dataset: The anomaly ratio decreases rapidly within a few iterations and stabilizes.

for each class and averages the values. The macro average F1-score provides a balanced assessment by first calculating the F1-score for each class (the harmonic mean of precision and recall for that class) and then averaging these F1-scores across all classes.

We compare our method *Refine* with various baselines, including Prodigy, Isolation Forest (IF) [30], Local Outlier Factor (LOF) [31], NegCo, LOE, NCAE, and IAD. We include Prodigy because it closely relates to our work on detecting performance anomalies in HPC systems. IF detects anomalies by constructing an ensemble of random decision trees and isolating samples based on the number of splits required; samples requiring fewer splits are flagged as anomalies due to their deviation from the majority. LOF identifies anomalies by comparing the density of a sample to the density of its nearest neighbors, flagging samples with significantly lower density as anomalous. While both methods are widely used for anomaly detection, we include only IF in Figure 5, as it outperforms LOF on our dataset. The other methods represent state-of-the-art techniques for unsupervised anomaly detection that specifically address the challenges of contaminated datasets and are applicable to our dataset format. We report the macro average F1-scores based on 5-fold cross-validation for all approaches. In Figure 5, error bars represent the 95th percentile confidence intervals, providing a visual indication of the variability in the model's performance across different folds. Shaded areas in Figures 6, 7, 9, and 8 represent 95th percentile confidence intervals. They show variability in the metrics: the macro average F1-score in Figure 6, anomaly ratio in the training dataset in Figure 7, and macro average recall and precision in Figures 9 and 8.

In Figure 5, we observe that the *Refine* with a single iteration achieves an F1-score above 0.80 up to a 1% anomaly ratio in the training set, but its performance drops below 0.60 when the anomaly ratio in the training dataset exceeds 2%. The baselines, including Prodigy, NCAE, and IAD, show a notable decrease in performance as the anomaly ratio in the training dataset increases. This decline is primarily because these methods rely on autoencoders, and the presence of anomalies in the training dataset distorts the training by disproportionately influencing the reconstruction error, leading to suboptimal latent representations. The quality of NCAE's generated high-confidence normal data suffers due to the distorted latent space caused by contamination, which undermines its contamination mitigation strategy. In IAD, the weights assigned to contaminated samples are intended to be lower than those for normal samples. However, these weights often remain inconsistent, sometimes resulting in similar or even higher weights for anomalous data. This inconsistency limits the effectiveness of such re-weighing strategies, particularly at higher contamination levels, motivating the need for alternative approaches that systematically reduce contamination. The classification performance of IF and LOE remains poor and stable across all anomaly ratios. LOE's poor performance is due to its transformation layers' inability to effectively handle the complex structure of our dataset, making it difficult for the model to differentiate between healthy and anomalous samples.

The anomaly ratio difference between the training and test datasets causes the poor performance of IF. In the Eclipse dataset, we initially set a 10% anomaly ratio in the training

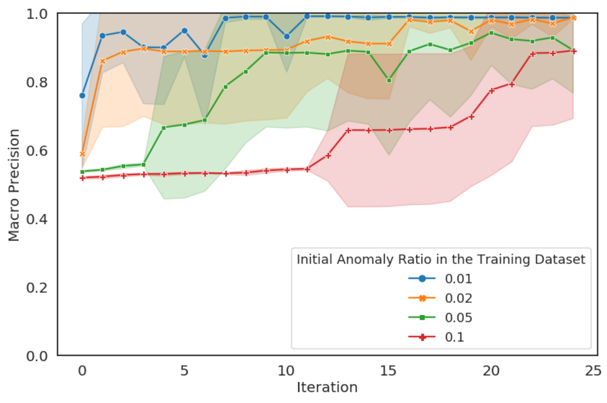


Fig. 8: Macro average precision achieved by *Refine* across iterations for varying initial anomaly ratios in the training dataset.

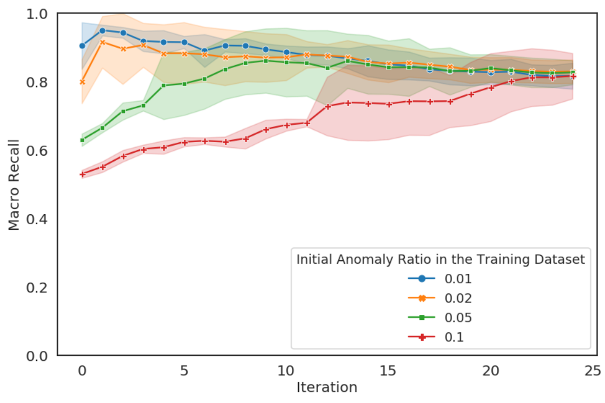


Fig. 9: Macro average recall achieved by *Refine* across iterations for varying initial anomaly ratios in the training dataset.

dataset to reflect realistic conditions. However, due to our initial data collection strategy, which includes more anomalous samples, the test dataset naturally ends up with a 90% anomaly ratio. This imbalance affects IF's performance, as it relies on a specified contamination ratio parameter during training, which does not align well with the actual distribution in the test dataset.

NegCo performs well when the anomaly ratio in the training dataset is below 5%. With only a small number of anomalous samples, the model effectively learns the semantic representations of true normal behavior, and the Gaussian distribution of the feature space closely aligns with this normal behavior. This allows the framework to generate reliable representative normal samples and maintain a distinct separation between normal and anomalous data. However, as the anomaly ratio reaches 10%, the presence of anomalous samples during training increasingly distorts the learned feature space, causing the Gaussian distribution to shift away from representing true normal behavior. This distortion undermines the framework's ability to generate accurate representative normal samples and reduces the reliability of the morphological similarity measure, ultimately impairing the model's performance in distinguishing normal and anomalous samples.

To mitigate the performance degradation that occurs at higher anomaly ratios in the training dataset, we run *Refine* across multiple iterations, testing with initial anomaly ratios of 1%, 2%, 5%, and 10% in the training dataset.

Figure 6 shows the F1-score we achieve on the test dataset after each iteration. As expected, with higher anomaly ratios in the training dataset, the starting F1-score is lower because a higher anomaly ratio results in a lower F1-score on the test dataset. For instance, with a 10% anomaly ratio in the training dataset, the starting F1-score is 0.19. However, after 20 iterations, *Refine* achieves an F1-score above **0.80**, representing a **400%** improvement. When we apply *Refine* iteratively, it outperforms all other baselines, regardless of the initial anomaly ratio in the training dataset, demonstrating its effectiveness across different levels of data contamination. Figure 7 shows that using the iterative method, it takes approximately 20 iterations to remove all anomalous samples at a 10% train anomaly ratio for the Eclipse dataset, 10 iterations at 5%, 4 iterations at 2%, and just 2 iterations at 1%. In contrast, for the Volta dataset, the anomaly ratio decreases rapidly within the first few iterations and typically stabilizes after 5 iterations, regardless of the initial anomaly ratio in the training dataset. These results highlight that the rate at which the anomaly ratio decreases with *Refine* iterations can vary across different datasets, likely due to differences in the characteristics of healthy and anomalous samples. It is important to note that after each iteration, we remove not only anomalous samples but also inevitably some healthy samples from the training dataset. While this increases the false alarm rate (proportion of healthy samples incorrectly classified as anomalies), the reduction in anomaly miss rate (proportion of anomalous samples incorrectly classified as healthy) is significantly greater, leading to overall improvements in the F1-score. For example, at an initial 1% contamination level (4,048 healthy and 46 anomalous samples), after three iterations, *Refine* reduces anomalous samples to only 3 while retaining 3,673 healthy samples, increasing the F1-score from 0.66 to 0.98. Similarly, at a higher 10% initial contamination level (4,048 healthy and 404 anomalous samples), after 25 iterations, *Refine* removes all anomalous samples while retaining approximately 2,400 healthy samples, again substantially improving the F1-score despite this reduction.

We also analyze the precision and recall trends of *Refine* across iterations. Figure 8, displays the macro average precision, which generally increases as iterations progress. For lower initial anomaly ratios (e.g., 1% and 2%), the increase in precision is sharper because the contamination is quickly mitigated, leaving a cleaner training dataset. However, at higher initial anomaly ratios, such as 10%, precision remains stable for the first 10 iterations before starting to increase and eventually reaching 85%. This stability can be attributed to the iterative refinement process, where the model becomes better at detecting anomalies (reducing false negatives) over iterations. Since macro-average precision considers precision for both classes equally, the improvement in anomaly detection

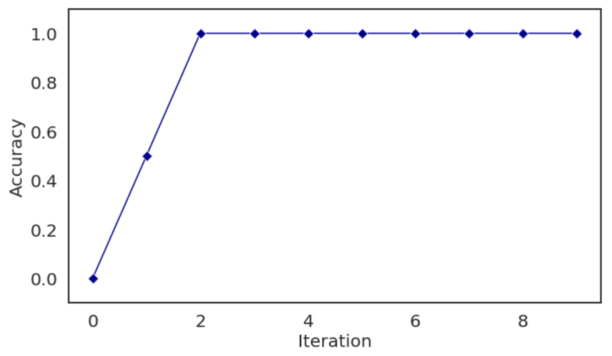


Fig. 10: Performance improvement in the contaminated Empire dataset (Anomalous Subset 2) with *Refine* iterations.

outweighs the increasing false positive rate, leading to an overall increase in precision.

In contrast, the recall trends in Figure 9 show different dynamics. When the initial anomaly ratio in the training dataset is higher (e.g., 10%), the recall increases consistently as iterations progress. The consistent reduction in anomaly miss rates outweighs the impact of the increasing false alarm rates, as the healthy class is underrepresented in the test dataset. For smaller initial anomaly ratios (e.g., 1%, 2%), recall peaks within the first few iterations because anomalies are quickly removed from the training dataset, leading to an immediate improvement in anomaly detection performance and a stabilization of recall in later iterations.

To validate our method with real-world production data without synthetic anomalies, we collaborate with a plasma physics expert at Sandia National Labs. The expert identifies performance degradation in runs of the Empire application. We execute this application multiple times on 4 compute nodes with the same input parameters, observing that 7 jobs complete in around 60 minutes (labeled as healthy) and 2 jobs take 50-60% longer (labeled as anomalous). When we train the VAE model with the 28 available healthy samples, it achieves 100% accuracy in identifying the 8 anomalous jobs in the test dataset. We have 2 anomalous jobs, one of which takes 90 minutes and the other 97 minutes to complete. To simulate contamination, we divide these jobs into two subsets: Anomalous Subset 1 (jobs that take 97 minutes) and Anomalous Subset 2 (jobs that take 90 minutes), each containing 4 samples. We then conduct experiments by adding these anomalous samples into the training dataset, contaminating it, and testing the model on the remaining anomalous samples. For instance, including samples from Anomalous Subset 1 in the training dataset maintains 100% accuracy. However, including samples from Anomalous Subset 2 results in 0% accuracy, indicating that these samples severely degrade the model's performance. We then apply *Refine* with multiple iterations to the contaminated dataset that results in poor performance, aiming to improve the model's classification accuracy. Figure 10 shows that after two iterations, *Refine* successfully removes all anomalous samples from the training dataset, restoring the model's performance to 100% accuracy.

TABLE III: Summary of Execution Time for only one Refine Iteration.

Statistic	Initial Training (s)	Train Until Knee Point (s)	Retraining (s)	Inference (s)
Mean	777.93	18.04	386.27	7.43
Std	37.34	2.94	27.55	0.19

TABLE IV: Summary of Execution for 25 Refine iterations.

Statistic	Initial Training (s)	Train Until Knee Point (s)	Retraining (s)	Inference (s)
Mean	14543.82	284.41	6962.93	179.92
Std	727.49	33.74	433.46	4.03

A. *Execution Time Analysis*

Execution time is crucial for assessing *Refine*'s practicality in large-scale HPC environments, where efficient processing of vast telemetry data with minimal overhead is essential for real-world integration. While *Refine*'s iterative approach involves multiple training and filtering stages, it does not impose significant computational overhead. We summarize the execution times for different stages of the *Refine* pipeline in Tables III and IV, detailing both a single iteration and the cumulative execution times after 25 iterations.

We conduct the experiments on a compute node equipped with an Intel Xeon E5-2670 processor (16 cores, 2.6 GHz). The pipeline includes four main stages: (1) the initial VAE training stage, (2) training the VAE model up to the knee point and identifying high-error samples, (3) retraining the VAE on the filtered dataset after removing high-error samples, and (4) inference on the test dataset. We report the execution times averaged across 20 runs of the Refine framework. These runs include four different initial anomaly ratios in the training dataset (1%, 2%, 5%, and 10%) and use a 5-fold cross-validation setup, as explained in the methodology section. From Table III, we observe a mean execution time of 777.93 seconds for the initial VAE training, 18.04 seconds for training up to the knee point and removing high-erorr samples from the training dataset, 386.27 seconds for retraining, and 7.43 seconds for inference during a single iteration. The inference time includes predictions for all 18,947 test samples. When we evaluate 25 iterations (Table IV), we find that the cumulative execution time does not increase linearly for training stages. As iterations progress, we remove high-error samples, which reduces the training dataset size and consequently decreases the training and retraining times. Specifically, the total time for initial training, training up to the knee point and retraining exhibits a sub-linear growth due to this iterative refinement. Inference time remains consistent across iterations, as it involves evaluating the same number of test samples (18,947) regardless of the training dataset size.

For comparison, Prodigy, executed on a compute node with two 14-core Intel Xeon E5-2680v4 processors (2.4 GHz), requires approximately 330 seconds for model training. In contrast, a single full iteration of *Refine* averages around 1182 seconds. Although *Refine*'s iterative filtering introduces additional computational overhead, this overhead remains acceptable within HPC environments, where job runtimes typically span hours or days. Furthermore, *Refine* achieves an inference time of 7.43 seconds for 18,947 samples, compared to Prodigy's 3.28 seconds on the same dataset, indicating that it remains suitable for real-time and batch anomaly detection scenarios. Note that although we use 20 iterations for performance evaluation in Figure 5, we report execution times for 25 iterations here to provide a more exhaustive view of cumulative overhead. Beyond 20 iterations, we observed negligible additional gains indicating that 20–25 iterations are practically sufficient in our scenario.

VII. Conclusion and Future Work

In this work, we introduce *Refine*, a robust unsupervised anomaly detection framework that enhances detection accuracy in contaminated datasets through an iterative filtering process. *Refine* achieves a macro average F1-score exceeding 0.80, even with a 10% anomaly ratio in the training dataset, significantly outperforming state-of-the-art methods in both synthetic and real-world HPC datasets. Its iterative filtering mechanism effectively removes anomalous samples from the training dataset in an unsupervised manner, improving the model's robustness against contamination. Moreover, *Refine* is ML model agnostic, allowing any reconstruction loss-based method to adopt its iterative refinement strategy without modifying the underlying model architecture.

Despite these achievements, several open challenges remain. One key challenge involves determining the optimal stopping point for the iterative process, particularly when ground-truth labels for the test dataset are unavailable. Without this information, deciding when to halt iterations becomes nontrivial, as continued removal of samples risks excluding healthy data, potentially increasing false alarm rates. Potential methods to address this issue include monitoring the stabilization of reconstruction error distributions, assessing convergence of reconstruction error thresholds, or calculating the Wasserstein distance [32] between reconstruction error distributions across iterations. These metrics could serve as proxies for determining whether additional iterations would yield meaningful improvements.

Future deployment of *Refine* in production HPC environments requires adaptability to evolving workloads, diverse applications, and emerging anomaly patterns. To address these dynamics, *Refine* must incorporate mechanisms for handling previously unseen applications and user-specific workloads, as well as for detecting novel anomaly types. Strategies such as application-specific training and periodic retraining using labeled telemetry data from nightly runs can enable *Refine* to continuously update its model and detection thresholds, ensuring its effectiveness over time.

Lastly, extending *Refine*'s applicability to a broader range of real-world workloads, including GPU-accelerated applications, is an important direction for future work. Conceptually, *Refine*'s iterative filtering remains directly applicable once GPU-specific telemetry metrics replace or complement existing CPU-focused features. Additionally, exploring how varying the size and architectural parameters of the VAE impacts detection performance could further enhance *Refine*'s effectiveness and generalizability. While our current evaluation includes real anomalies from Empire applications, demonstrating *Refine*'s effectiveness across diverse real-world scenarios and heterogeneous computing platforms will further validate its practical robustness.

References

[1] S. Fiore, M. Bakhouya, and W. W. Smari, "On the road to exascale: Advances in high performance computing and simulations—an overview and editorial," pp. 450–458, 2018.

[2] D. Skinner and W. Kramer, "Understanding the causes of performance variability in hpc workloads," in *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.* IEEE, 2005, pp. 137–149.

[3] A. Jokanovic, G. Rodriguez, J. C. Sancho, and J. Labarta, "Impact of inter-application contention in current and future hpc systems," in *2010 18th IEEE Symposium on High Performance Interconnects.* IEEE, 2010, pp. 15–24.

[4] Y. Zhang, T. Groves, B. Cook, N. J. Wright, and A. K. Coskun, "Quantifying the impact of network congestion on application performance and network metrics," in *2020 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, 2020, pp. 162–168.

[5] A. Agelastos, B. Allan, J. Brandt, A. Gentile, S. Lefantzi, S. Monk, J. Ogden, M. Rajan, and J. Stevenson, "Toward rapid understanding of production hpc applications and systems," in *2015 IEEE International Conference on Cluster Computing.* IEEE, 2015, pp. 464–473.

[6] J. Klinkenberg, C. Terboven, S. Lankes, and M. S. Müller, "Data mining-based analysis of hpc center operations," in *2017 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, 2017, pp. 766–773.

[7] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Online diagnosis of performance variation in hpc systems using machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 883–896, 2018.

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

[9] B. Aksar, E. Sencan, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, B. Kulis, and A. K. Coskun, "Albadross: Active learning based anomaly diagnosis for production hpc systems," in *2022 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, 2022, pp. 369–380.

[10] M. Molan, A. Borghesi, D. Cesarini, L. Benini, and A. Bartolini, "Ruad: Unsupervised anomaly detection in hpc systems," *Future Generation Computer Systems*, vol. 141, pp. 542–554, 2023.

[11] B. Aksar, E. Sencan, B. Schwaller, O. Aaziz, V. J. Leung, J. Brandt, B. Kulis, M. Egele, and A. K. Coskun, "Prodigy: Towards unsupervised anomaly detection in production hpc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–14.

[12] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *Advances in neural information processing systems*, vol. 31, 2018.

[13] J. Yu, H. Oh, M. Kim, and J. Kim, "Normality-calibrated autoencoder for unsupervised anomaly detection on data contamination," 2021.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[15] X. Lin, Z. Li, H. Fan, Y. Fu, and X. Chen, "Exploiting negative correlation for unsupervised anomaly detection in contaminated time series," *Expert Systems with Applications*, vol. 249, p. 123535, 2024.

[16] L. Li, J. Yan, Q. Wen, Y. Jin, and X. Yang, "Learning robust deep state space for unsupervised anomaly detection in contaminated time-series," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 6, pp. 6058–6072, 2022.

[17] H. Xu, Y. Wang, S. Jian, Q. Liao, Y. Wang, and G. Pang, "Calibrated one-class classification for unsupervised time series anomaly detection," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[18] S. Kim, Y. Choi, and M. Lee, "Deep learning with support vector data description," *Neurocomputing*, vol. 165, pp. 111–117, 2015.

[19] C. Qiu, A. Li, M. Kloft, M. Rudolph, and S. Mandt, "Latent outlier exposure for anomaly detection with contaminated data," in *International conference on machine learning*. PMLR, 2022, pp. 18 153–18 167.

[20] Q. Zhou, S. He, H. Liu, J. Chen, and W. Meng, "Label-free multivariate time series anomaly detection," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[21] M. Kim, J. Yu, J. Kim, T.-H. Oh, and J. K. Choi, "An iterative method for unsupervised robust anomaly detection under data contamination," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[22] L. Perini and J. Davis, "Unsupervised anomaly detection with rejection," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[23] T. Kim, J. Oh, N. Kim, S. Cho, and S.-Y. Yun, "Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation," 2021.

[24] E. Ates, Y. Zhang, B. Aksar, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Hpas: An hpc performance anomaly suite for reproducing performance variations," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[25] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden *et al.*, "The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 154–165.

[26] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)," *Neurocomputing*, vol. 307, pp. 72–77, 2018.

[27] F. Benford, "The law of anomalous numbers," *Proceedings of the American philosophical society*, pp. 551–572, 1938.

[28] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900.

[29] "Kneed documentation," https://kneed.readthedocs.io/en/stable/index.html, 2024, accessed: 2024-08-26.

[30] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[31] Z. Cheng, C. Zou, and J. Dong, "Outlier detection using isolation forest and local outlier factor," in *Proceedings of the conference on research in adaptive and convergent systems*, 2019, pp. 161–168.

[32] V. M. Panaretos and Y. Zemel, "Statistical aspects of wasserstein distances," *Annual review of statistics and its application*, vol. 6, no. 1, pp. 405–431, 2019.