Fault Detection in Manufacturing Equipment Using Unsupervised Deep Learning

by

Damien W. Martin

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2021

Author ....
Department of Electrical Engineering and Computer Science
January 15th, 2021

Certified by..
Duane S. Boning
Clarence J. LeBel Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by..
Jeffrey H. Lang
Vitesse Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Fault Detection in Manufacturing Equipment Using Unsupervised Deep Learning

by

Damien W. Martin

Submitted to the Department of Electrical Engineering and Computer Science
on January 15th, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We investigate the use of unsupervised deep learning to create a general purpose automated fault detection system for manufacturing equipment. Unexpected equipment faults can be costly to manufacturing lines, but data driven fault detection systems often require a high level of application specific expertise to implement and continued human oversight. Collecting large labeled datasets to train such a system can also be challenging due to the sparse nature of faults. To address this, we focus on unsupervised deep learning approaches, and their ability to generalize across applications without changes to the hyper-parameters or architecture. Previous work has demonstrated the efficacy of autoencoders in unsupervised anomaly detection systems. In this work we propose a novel variant of the deep auto-encoding Gaussian mixture model, optimized for time series applications, and test its efficacy in detecting faults across a range of manufacturing equipment. It was tested against fault datasets from three milling machines, two plasma etchers, and one spinning ball bearing. In our tests, the model is able to detect over 80% of faults in all cases without the use of labeled data and without hyperparameter changes between applications. We also find that the model is capable of classifying different failure modes in some of our tests, and explore other ways the system can be used to provide useful diagnostic information. We present preliminary results from a continual learning variant of our fault detection architecture aimed at tackling the problem of system drift.

Thesis Supervisor: Duane S. Boning
Title: Clarence J. LeBel Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Jeffrey H. Lang
Title: Vitesse Professor of Electrical Engineering and Computer Science

Acknowledgments

This project has been an incredibly rewarding experience and I've learned an enormous amount over its course. For that I want to thank my advisors, Professor Duane Boning and Professor Jeff Lang for their invaluable guidance throughout this project. As well as Vivek, Oliver, and the rest of the team at HARTING and Lam Research for their support and technical expertise.

For helping me get this far I want to thank Ian, Meagan, and my friends in the extended Senior House community for always being there when I needed it most. Mr. Oles for getting me started down this path. Most of all, thank you parents, for all of the above and more. I couldn't have done it without you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the manufacturing sector, unplanned downtime caused by unexpected machine faults can be highly costly and is estimated to reduce factory output by between 5% and 20% [8]. Automated fault detection (AFD) systems can help mitigate much of this damage by monitoring sensor data and flagging anomalies before problems become more serious. With advanced warning, maintenance can be scheduled ahead of any catastrophic failures, referred to as predictive maintenance (PdM). A 2017 survey by GE estimates over 80% of organizations had experienced an episode of unplanned downtime in the past three years, costing an average of $260,000 an hour, half of which were caused by unexpected hardware failures [3]. This suggests that there may be large efficiency gains to be made in manufacturing should these methods be more widely adopted.

In this work, we address three potential barriers to more widespread adoption of AFD systems. The first involves the difficulties of retrofitting sensors onto older machinery. For equipment that does not come with integrated sensors and monitors, invasive sensor installation can risk damaging the equipment in question, and in a factory setting, may require shutting down production lines. The second problem we address is the lack of large labeled datasets to train such systems. Faults are inherently a rare event and every failure may be different. The final barrier we hope to address is the high level of domain-specific expertise that can be required to build an effective AFD system. Tailor made solutions often rely on an intimate knowledge of the machinery in question and its failure modes. To address these three issues we frame our problem as a more general case of anomaly detection and leverage advances in unsupervised learning to build system models without labeled data. In this work, we propose and test a deep learning architecture capable of detecting faults across a wide range of manufacturing equipment without labeled data or application specific configuration.

1.1 Model Summary

In this thesis, we propose a new architecture for automated fault detection: the Recurrent Deep Autoencoding Gaussian Mixture Model (RDAGMM). The core of our model is a sliding window autoencoder. This architecture allows our system to model incoming sensor data in an unsupervised way, and provides the flexibility to generalize across applications. The autoencoder is trained to compress and reconstruct incoming windows of sensor data, with a series of convolutional layers. A recurrent unit in the final layer of the encoder allows the model to track long term trends outside the scope of a single window. The autoencoder outputs two vectors useful for anomaly detection, the latent state (compressed representation) and the residuals vector (reconstruction error). These two outputs are passed to a secondary classification network that is trained to minimize the negative log likelihood (NLL) of a Gaussian mixture model. The NLL is used as our final “anomaly score”: a measure of how aberrant the monitored equipment’s behavior is at any given moment in time. Setting a threshold on the anomaly score provides a binary label as to whether the system is performing normally or if there is a fault. Clustering the compressed latent state representations provides unsupervised labeling of different fault types useful for diagnostic purposes.

1.2 Evaluation

To test the ability of our model to generalize across applications, we collected a range of datasets containing sensor readings from bearings, milling machines, and circuit fabrication equipment. As well as covering a range of equipment, these datasets vary in terms of types of signals being recorded, the number of sensor channels available, and the sampling frequency. Half of our datasets exclusively use non-invasive sensors.

All of the datasets contain labeled faults that either occurred naturally while the machine was in use, or were intentionally induced for testing purposes. In all of the above cases, we evaluate the accuracy with which our system can separate faulty from nominal behavior. We also investigate the ability of our system to classify different types of faults, attribute faults to specific sensor channels, and provide other information useful for fault diagnosis and predictive maintenance.

1.3 Thesis Outline

The next chapter of this thesis contains a review of unsupervised anomaly detection techniques in the literature, with an emphasis on methods used in this project. Chapter 3 then describes our model architecture in more detail, the methods used for fault detection and classification, and the details of the models used as a baseline for comparison. Chapter 4 details the various data sets and fault detection tasks we use to test our method. Chapter 5 contains the results of our tests. In Chapter 6, we discuss our preliminary work collecting more expansive datasets and testing improvements to our architecture. Finally, we conclude with a discussion of our results and suggestions for further work in Chapter 7.

Chapter 2

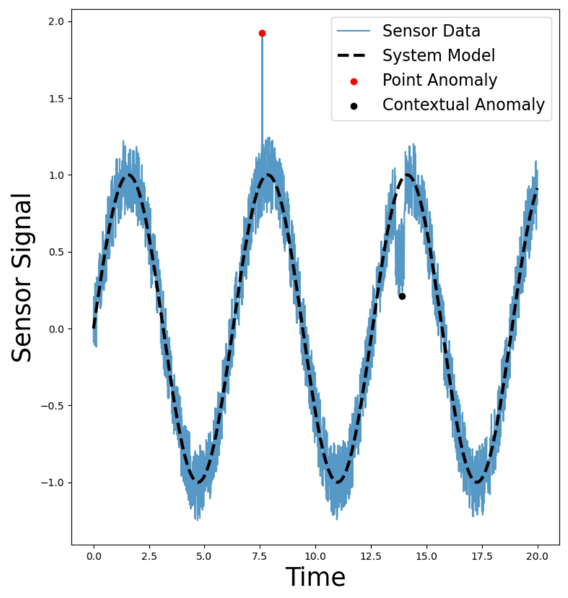
Theory & Literature Review

Our work draws on literature in the fields of anomaly detection, fault detection, and unsupervised learning. In this chapter, we summarize the relevant related research.

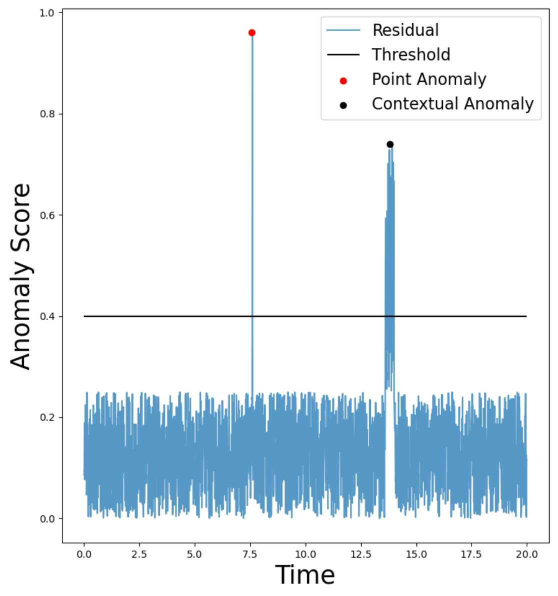
2.1 Anomaly Detection

Mechanical faults in machinery can often be detected and even predicted by finding the appearance of anomalies in the available sensor readings. While it is difficult to rigorously define anomalies, they can intuitively be thought of as observations that seem abnormal given historical data. In this paper, we will be using the definition of anomalies given by Hawkins as points which "deviate so significantly from other observations as to arouse suspicion that they were generated by a different mechanism" [14].

Anomalies can be further subdivided into two classes, point anomalies and contextual anomalies. While point anomalies can and often are detected as limit violations of normal operating parameters, contextual anomalies are only detectable with regards to surrounding values or readings from other sensors. Conceptually, they require some knowledge of what the signal should look like to compare against. For example, with an accurate model of nominal system behavior, an anomaly score can be calculated as the size of the residual; the absolute error between the model's predictions and the



(a) Sensor signal y and system model $\hat{y}$.



(b) Residual vector - $abs(y - \hat{y})$.

Figure 2-1: Residual based anomaly detection. Red and black points highlight two anomalies which are detectable as points with a residual value above a threshold.

recorded sensor data, as shown in Figure 2-1.

In this work, we focus on unsupervised anomaly detection for multivariate time series. Without access to expert knowledge of the system, we first learn a model of nominal system performance using historical data, and then detect anomalies as a function of the predictive accuracy of our model, our model's state, and other learned metrics.

2.2 Deep Unsupervised Modeling

In the field of deep learning, unsupervised methods used for anomaly detection can be roughly subdivided into three categories, based on their learning objective and anomaly scoring technique. These are time series forecasting, clustering techniques, and compression based methods. In this section, we review the theory and literature behind these methods. All three have close analogs to traditional anomaly detection methods, rely on the assumption that faults are relatively rare or not present in the training data, such that accurate models of nominal behavior can be trained without labels.

2.2.1 Time Series Forecasting

Time series forecasting involves building a model to predict future values of a time series, typically using a sliding window of past values. A classic method to solve this problem is to construct an autoregressive moving average (ARMA) models in which predicted future values $\hat{y}_t$ are computed as a linear combination of past values y_{t-i} and error terms ϵ_{t-i}, as defined in Equation 2.1:

$$\hat{y}_t = \sum_{i=1}^{p} \alpha_i \times y_{t-i} + \sum_{i=1}^{p} \beta_i \epsilon_{t-i} \tag{2.1}$$

where α_i and β_i represent the learned model parameters. Anomalies can then be detected as periods of abnormally large errors in the model predictions indicating a break from historical trends. as shown in Figure 2-2. For example, Qin et al. used this combination of ARMA and a statistical threshold on the residuals vector to detect faults in satellite systems [31].

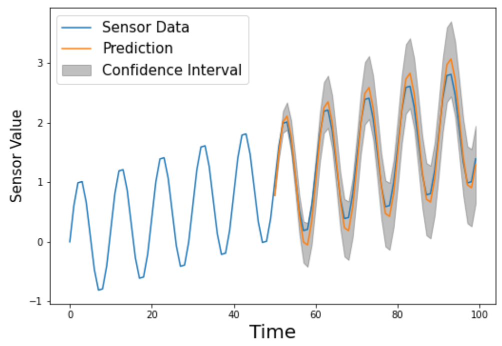
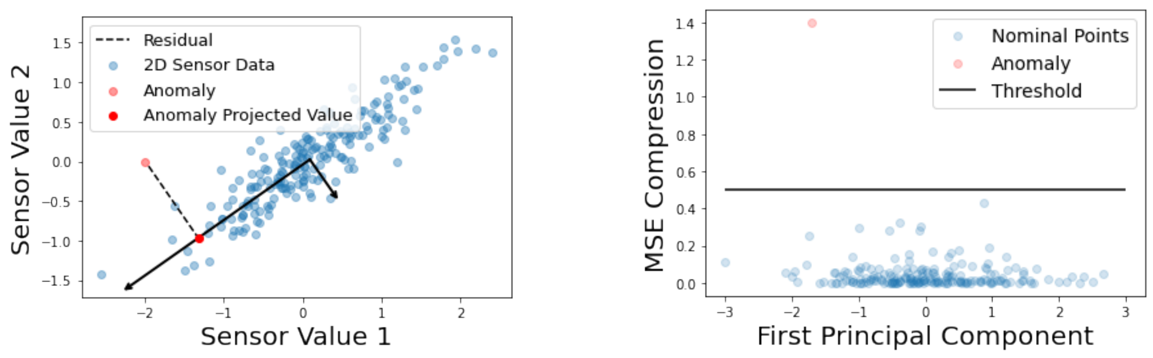


Figure 2-2: ARIMA Forecast. After fitting a model on historical data, future values are predicted on a rolling basis. Points which fall outside the projected confidence interval and labeled as anomalies.

In the field of deep learning, a common architecture for time series predictions are RNNs (Recurrent Neural Networks), particularly LSTMs (Long Short-Term Memory) and GRUs (Gated Recurrent Units). These networks maintain an internal state and at each time step, produce an output and updated state as a function of both the prior state and current input. In the field of anomaly detection, this method has proven effective at detecting faults in cyber-physical systems [27] as well as in satellite systems [10], in both cases improving on industry standard techniques.

2.2.2 Compression

Compression based techniques such as principal component analysis (PCA) learn to compress segments of the incoming time series to a lower dimensional representation by learning common factors of variation. In PCA, this involves decomposing historical data using singular value decomposition and reducing the dimension of the dataset using the vectors associated with the k largest singular values, known as the principal components. This technique produces a linear model that explains the maximum variance. Signals are then compressed and reconstructed by projection onto the principal components, as shown in Figure 2-3.



(a) Raw sensor data.   (b) MSE after compression.

Figure 2-3: Anomaly detection via compression. In this example 2d sensor data is projected onto the first PCA component, a 1d representation. The square of the distance between the original points location and the projected value after compression is used as the anomaly score.

As with forecasting methods, anomalies can then be detected as increases in the magnitude of the residual vector between the original incoming signal and the compressed version of this signal [25], [17]. This is due to the fact anomalies will be poorly compressed, as a consequence of failing to follow common factors of variation found in the nominal training data.

In the field of deep learning, unsupervised compression is often performed with autoencoders [12], [11], [37], [26], [16], [30], [13]. These neural networks reduce the dimension of the data to k hidden variables, known as the latent state, and are trained to reconstruct the original inputs using SGD (Stochastic Gradient Descent) or an equivalent optimization algorithm. With a nonlinear activation function between layers such as RELU (Rectified Linear Units), this is directly analogous to a non-linear form of PCA.

2.2.3 Clustering

Clustering based anomaly detection involves grouping incoming data into classes using algorithms such as k-means or Gaussian mixture models [9] [36] [38] [39]. Anomalies can then be detected as points which fall outside of known clusters, or as points that fall into clusters that are known to be associated with faults. These methods tend to perform poorly on high dimensional data sets, and thus are often combined with methods like PCA to first project the data to a lower dimensional feature space before clustering.

In unsupervised anomaly detection, one popular technique is OC-SVDD (One class - Support Vector Data Description) [33]. These systems attempt to separate nominal points into a single class by finding the smallest possible hyper-sphere which encapsulates the majority of training data as shown in Figure 2-4.

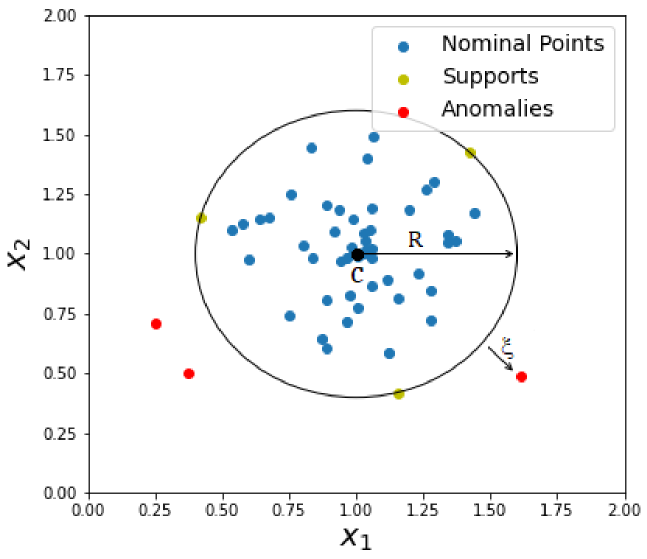


Figure 2-4: Anomaly detection by clustering. R denotes the radius to the decision boundary and c is the cluster center.

This is accomplished by minimizing the constrained optimization problem formulated in Equation 2.2:

$$\begin{aligned} \min R^2 + \frac{1}{vn}\sum_i \xi_i \\ s.t. ||\Phi_k(x_i) - c||^2 < R^2 + \xi_i \\ \xi_i \geq 0, \forall i \end{aligned} \tag{2.2}$$

where R is the radius of the hyper-sphere, c is the cluster center, Φ_k is the encoding function, and ξ_i is a slack variable denoting the radial distance of anomalous point from the boundary of the hyper-sphere. In this framework, the anomaly score is calculated as the radial distance of every point from the cluster center, and the threshold is determined by the hyper-parameter $0 < v < 1$ which acts as a prior on the expected percentage of anomalies within the training data set [33].

2.2.4 Hybrid Approaches

Several recent state of the art approaches to unsupervised anomaly detection are based on combining and tightly integrating the three aforementioned approaches to anomaly detection. Forecasting, clustering, and compression based anomaly detection complement each other in a variety of ways, and combinations of these approaches often prove effective and robust. For example, as mentioned above, clustering techniques such as k-means, GMMs, and SVDDs are often used on the compressed representation from PCA or on the latent state of a trained autoencoder, because the reduction in input dimension can reduce training time, improve generality, and prevent overfitting. Making use of this synergy, Deep Autoencoding Gaussian Mixture Models (DAGMMs), proposed by Zong et al. [40], simultaneously train an autoencoder to compress and reconstruct the inputs, as well as an auxiliary estimation network to predict sample class based on the autoencoders latent state and the mean squared residual error. After empirically determining the parameters of the cluster centers, anomalies are detected as a function of the sample data point energy, analogous to the negative log likelihood under the GMM framework, as shown in Equation 2.3. This method effectively combines clustering based anomaly detection with the residual error from reconstruction:

$$E(z) = -log(\sum_{k=1}^{K} \hat{\phi}_k \frac{exp(-2(z-\mu_k)^T \Sigma_k^{-1}(z-\mu_k)/2)}{\sqrt{|2\pi\Sigma_k|}}) \tag{2.3}$$

where K represents the number of clusters, μ_k and Σ_k define the cluster means and variances, ϕ_k defines the mixture probability, and z is the latent state vector. Autoencoders are key to many hybrid approaches to fault detection because they can very naturally be used to complement both forecasting and clustering methods. OC-neural networks in particular can be thought of as a single end-to-end method of training both the compressive, feature-crafting stage, and the clustering step. Similarly, by shifting the target vector in time, autoencoders can be trained to both compress and predict future values, allowing for better detection of contextual anomalies that may be missed by compression alone [22].

2.2.5 Variational Autoencoders

Variational inference is the practice of using optimization algorithms to efficiently find approximations of unknown distributions, when calculating exact maximum likelihood solutions may be intractable [2]. In the case of VAEs (Variational Autoencoders), we begin by assuming that our input data x is generated by a random process as a function of an unobserved state z, where z is drawn from an unknown distribution $p_{\theta^*}(z)$ and x from $p_{\theta^*}(x|z)$. A deep encoding neural network is trained to parameterize a distribution $q_\phi(z|x)$ (typically Gaussian) to approximate the true $p_{\theta^*}(z|x)$. The decoder network is trained to approximate $p_{\theta^*}(x|z)$ and reconstruct the input from the latent state. To this end, given an input x, our encoding network produces a mean and standard deviation for the distribution $q_\phi(z|x)$. A value for the latent state is then sampled from this distribution, where $z \sim \mathcal{N}(\mu, \sigma)$, and the decoder produces an approximate value x_r for the sampled value, as defined in Equation 2.4:

$$\begin{aligned} &\mu_z, \sigma_z \leftarrow Enc(x) \\ &z \sim q_\theta(z|x) = \mathcal{N}(\mu_z, \sigma_z) \\ &x_r \leftarrow Dec(z) \end{aligned} \tag{2.4}$$

where *Enc* and *Dec* represent the encoding and decoding networks, respectively.

This method has multiple benefits when compared to deterministic autoencoders. Because standard autoencoders place no restrictions on how the latent state is represented internally, small changes in the latent state can lead to drastic changes in the output. By contrast, in a VAE, stochastically sampling the latent state from a distribution forces the system to encode similar samples nearby in latent space. This allows for more robust clustering and more interpretable results when analyzing the effect of individual dimensions of the latent state. It also allows for the decoder to function as a generative model by sampling from within the prior distribution of states.

For time series modeling this method can be further improved by using variational methods to train recurrent RNNs, allowing for the prior distribution to be updated sequentially, conditional on the previous state [28, 22, 16].

Chapter 3

Methods

In this chapter, we first describe the architecture of the proposed FDS (Fault Detection System) in more depth, and the methods we used to evaluate the performance of our system. At a high level there are four main components to the FDS tested in this project: a normalization layer, a symmetrical encoding-decoding scheme, an information bottleneck, and an anomaly scoring mechanism, as shown in Figure 3-1.

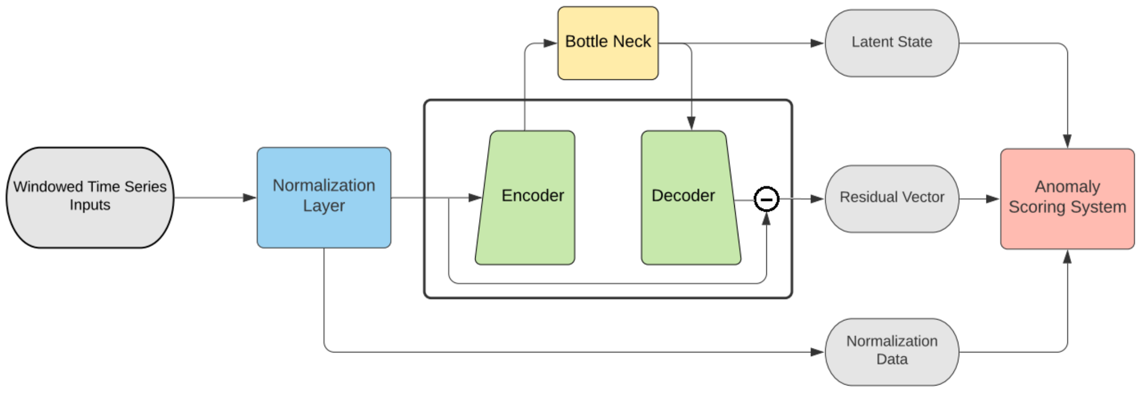


Figure 3-1: Fault detection system overview.

The normalization layer serves to standardize the scale and variance of each input channel. When working with multivariate time series, values from various sensors can span many orders of magnitude causing some signals to have an outsized effect on the loss function.

The encoder-decoder system represents the function learned to compress incoming windows of data and to reconstruct said inputs from the latent state. The bottleneck defines how information is compressed between the encoding and decoding network. The anomaly scoring system defines the clustering, residual processing, and other methods used to compute a final anomaly score. For binary fault detection, this involves reducing the incoming sensor data to a single value, representing the anomaly score, and setting a threshold on this score to detect faults. For fault classification, this involves clustering the data into an unknown number of classes.

3.1 Neural Network Architecture

With the exception of preprocessing steps such as normalization, our system is implemented as a single end-to-end differentiable deep neural network. Our neural network architecture builds off of the DAGMM, with modifications to optimize the architecture performance on time series applications. Our network architecture can be seen in Figure 3-2, highlighting the convolutional and recurrent components and the class estimation network.

First, a CNN (convolutional neural network) is used the encode the incoming time series window. A GRU (gated recurrent unit) tracks these encoded windows, and outputs a latent state vector as a low dimensional representation of the current state of the machine. A transposed convolutional network then reconstructs an approximation of the original input window from the latent state vector. Finally, our estimation network assigns a class label to each window, based on the features from the residual vector and the latent state, with the goal of finding a clustering that maximizes the likelihood of generation by a Gaussian mixture model.

This architecture was chosen to maximize the ability of our system to generalize across applications by combining many aspects of anomaly detection systems that have shown success previously. To allow it to detect a wide range of anomalies, it combines both a residual and clustering objective with optimizations specific to time series applications. To improve robustness without tuning, it features a number of

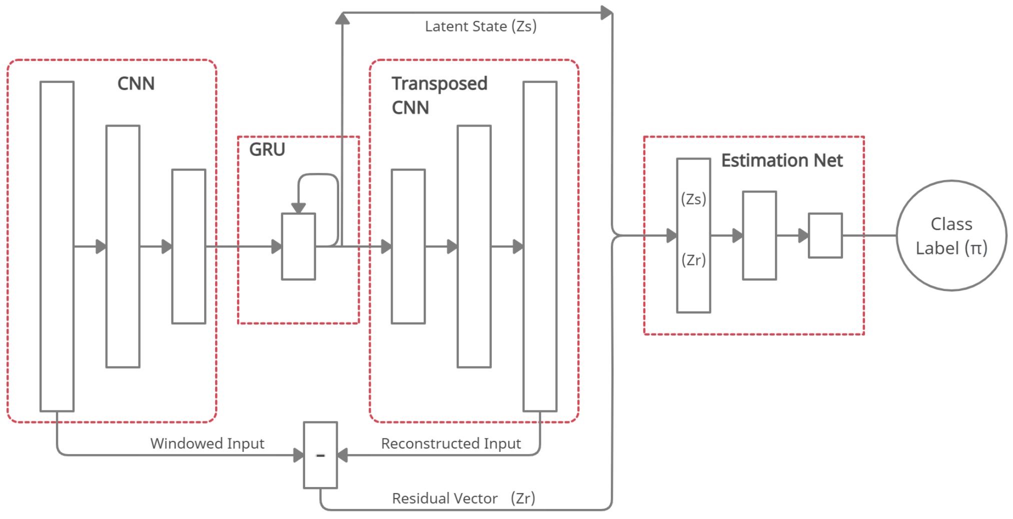


Figure 3-2: Schematic for the full neural network architecture. Convolutional layers are used for compression and reconstruction. The latent state is calculated by a recurrent unit. A probability distribution over class labels is output by the estimation network. Not pictured above, throughout training the empirical mean and variance of each class is tracked. The anomaly score is computed as the NLL of new data point being generated by the GMM parameterized by the empirical distribution.

auxiliary losses to regularize the network as a whole. It was also chosen to maximize the potential for future work in predictive maintenance and diagnostics, due to its rich output space including the per channel residuals vector, compressed latent state, and cluster centers. The following sections describe each sub-network in more detail. All networks were implemented using tensorflow, keras, and alibi-detect [35].

3.1.1 Autoencoder

The core of our fault detection system is a sliding window autoencoder. To allow for easier recognition of time-independent features, our architecture uses 1D convolutional and transposed convolutional layers as the encoder and decoder, respectively, with all convolutions taking place over the time axis. The signal is compressed leading up to the bottleneck by using a convolution stride of two, effectively halving the window size at each encoding layer. The de-convolutional layers in the decoder then up-sample the latent state using transposed convolutions until the output matches the original dimension of the input window, as shown in Figure 3-3. All of the models tested in this work use four layers in both the encoder and decoder with a window size of 128 and 64 kernels each of size 16. This method has the added benefit of scaling the network size and number of parameters with the number of input channels as this dimension is held constant through compression and reconstruction.

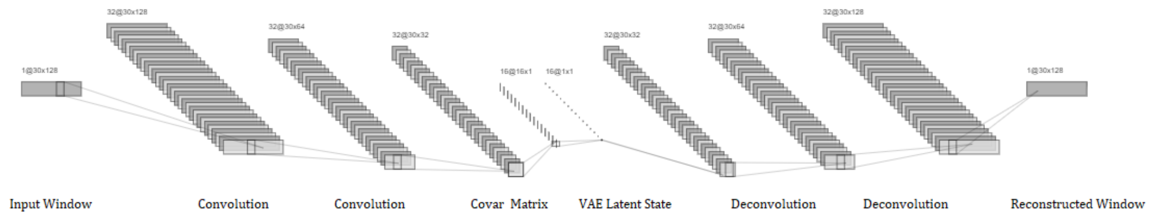


Figure 3-3: Scale diagram of the convolutional layers in the encoder and decoder networks. Window length is shown as the width of each filter and as pictured is halved during each encoding step. Channel count is shown is shown as the height. Filters are stacked in z.

3.1.2 Bottleneck

The bottleneck of our neural network controls the final layer of compression and how the latent state is represented internally. In our architecture, we use a variational GRU. At every timestep t the GRU outputs a mean $\mu_{z,t}$ and variance $\sigma_{z,t}$ parameterizing a posterior distribution over latent state values $q(z_t|x_t, z_{t-1})$. It takes as input the current output from the encoder x_t, and the distribution over latent state values from the previous time step $\mu_{z,t}$ and $\sigma_{z,t}$ effectively performing a Bayesian update on the distribution over latent state values. A value for the latent state z_t is then sampled from the distribution $q_t(z_t|x_t, z_{t-1})$. This recursive element allows our system to track long term trends beyond the length of our input window when calculating the current latent state. The Kullback-Leibler divergence between the distribution q and an independent normal is then added to the loss as described in Section 2.2.5. The variational objective further improves performance by adding a form of regularization to the size of the information bottleneck as well as improving the interpretability of the final latent state values.

3.1.3 Estimation Network

The estimation network assigns class labels to every input window as a function of the latent state and the residuals vector. It takes as input the sampled value for the latent state z_{ls} and features from the residual vector z_r such as the MSE and cosine similarity, and returns a probability distribution over class labels. These two features were used in line with the original DAGMM paper [40], but the method is agnostic to the choice of residual features. These two vectors define the size of the clustering space for the network. The size of the latent state vector and the number of clusters are both set as hyperparameters in advance. In this work, our model uses a latent state vector of size ten, with two cluster centers across all datasets. We found our detection accuracies and classification accuracies to be fairly robust to changes in these hyperparameters. Note that the class labels calculated here are not the ones used for calculating the classification accuracy; the final labels are computed by a separate procedure described in Section 3.2.3. The estimation network itself is implemented as a fully connected dense network with two layers ending with a softmax activation function.

3.1.4 Training and Loss Functions

The network is trained to minimize two loss functions. The first is the reconstruction loss, defined as the MSE (Mean Squared Error) between the normalized input window x_i and the reconstructed input window x'_i after compression, as shown in Equation 3.1:

$$\mathcal{L}_{rec} = \frac{1}{N}\sum_{i=0}^{N}(||x_i - x'_i||_2^2) \tag{3.1}$$

where N is the number of points in a training batch. This objective pushes the encoder to compress the input as efficiently as possible.

The second function is the clustering loss under the GMM framework. To calculate the clustering loss, the empirical means μ_k and co-variance matrices Σ_k, and mixture components ϕ_k for each class k (collectively referred to as Θ_{emp}) are tracked and updated throughout training. The rate at which the parameters defining every cluster are changed after every training batch is controlled by a hyperparameter β as shown in Equation 3.2, where Z is the collection of encoded windows in a single training batch.

$$\begin{aligned} \mu_k &= \beta * \mu_k + (1-\beta) * \frac{\sum_{z_i \in Z} p(k|z_i) z_i}{\sum_{z_i \in Z} p(k|z_i)} \\ \Sigma_k &= \beta * \Sigma_k + (1-\beta) * \frac{\sum_{z_i \in Z} p(k|z_i)(z_i - \mu_k)^2}{\sum_{z_i \in Z} p(k|z_i)} \end{aligned} \tag{3.2}$$

This update method ensures that the cluster means and variances change smoothly between batches and makes network training more stable. The clustering loss is then calculated as the negative likelihood of a batch of data being generated by a GMM with the aforementioned parameters, as shown in Equation 3.3:

$$\mathcal{L}_{gmm} = -log\left(\sum_{k \in K}\sum_{z_i \in Z} \phi_k \frac{exp(-.5(z_i - \mu_k)^T \Sigma_k^{-1}(z_i - \mu_k))}{(|2\pi\Sigma_k|)^{.5}}\right) \tag{3.3}$$

where k is the number of clusters.

3.1.5 Network Optimization

Several design choices are made to improve the performance and robustness of our models. Every network makes use of L2 weight regularization. Optimization during training was performed using ADAM. One of the goals of this project was to design the system to be robust to hyperparameter tuning. All hyperparameters in our system were initially chosen using values from similar architectures in the literature. After finding a set of parameters that performed well on a synthetically generated dataset, all hyperparameters were kept fixed for all further tests on all of our other datasets. It remains an area of future work to determine how much performance can be improved with careful tuning or how to optimize hyperparameters in an automated fashion by incorporating some form of meta-learning.

3.2 Anomaly Scoring System

This section details how our model is used to detect faults and to provide diagnostic information about the type and causes of different faults. Pseudocode for generating fault labels and class labels using a trained model is shown below in Algorithm 1.

Algorithm 1 Fault Detection & Classification

```
 1: procedure AFD($\phi_{enc}, \phi_{dec}, \phi_{est}, \Theta_{emp}, \tau, X$)    ▷ Input: Trained NN weights $\phi$;
    GMM parameters $\Theta$; threshold $\tau$, and test dataset $X$
 2:     $\vec{z}_{ls} \leftarrow \phi_{enc}(X)$
 3:     $X' \leftarrow \phi_{dec}(ls)$
 4:     $\vec{z}_r \leftarrow X - X'$
 5:     $\vec{z} \leftarrow concatenate(\vec{z}_r, \vec{z}_{ls})$
 6:     $\vec{\pi} \leftarrow \phi_{est}(\vec{z})$
 7:     $Anomaly\_Score \leftarrow NLL(\vec{z}, \vec{\pi}|\Theta_{emp})$
 8:     $Anomaly\_Score \leftarrow (Anomaly\_Score - \mu_{AS}(X_d)) / \sigma_{AS}(X_d)$
 9:     $Fault\_Labels \leftarrow (Anomaly\_Score > \tau)$
10:     $BIC_{min} \leftarrow Inf$
11:     for $i$ in range($N$) do
12:         $\Theta_i = argmax_{\Theta}\,(p(\vec{z}|\Theta))$    ▷ Refit GMM with $i$ clusters using EM
13:         if $BIC_{min} > BIC(\vec{z}, \Theta_i)$ then
14:             $\Theta^* \leftarrow \Theta_i$
15:             $BIC_{min} \leftarrow BIC(\vec{z}, \Theta_i)$
16:     $Class\_Labels = \Theta^*(z)$
17:     return Fault_Labels, Class_Labels, $\Theta^*$
```

3.2.1 Anomaly Score

The anomaly score for every window of sensor data is equivalent to the clustering loss $\mathcal{L}_{gmm}$, calculated as the negative log likelihood with reference to the empirical GMM as shown in Equation 3.3. As values of the probability density function defined by the GMM, the ratio of two anomaly scores can be used to calculate the relative likelihood of seeing two given windows of sensor data. For datasets that are further discretized into runs, such as a single wafer etch in the plasma etcher database, the mean anomaly score of all windows contained in its duration is used to represent the entire run. More generally, for continuous data without clear boundaries between runs, anomalies can be found on different time scales by taking the moving average of the anomaly score over the given duration.

3.2.2 Fault Labels and Thresholding

This section describes how thresholds are set and runs are assigned a final label of faulty or anomalous using a validation dataset X_v. First, anomaly scores are computed for every run in the validations dataset AS. For datasets containing discrete variables X_d, defining separate run types, such as the etch recipe being performed, we further compute the mean $\mu_{AS}(X_d)$ and standard deviation $\sigma_{AS}(X_d)$ for runs of each type independently. We use these statistics to normalize the anomaly scores for each run type. The threshold to detect faults is then set by default at two standard deviations above the mean.

3.2.3 Fault Classification

For data sets with multiple types of faults, we also test the ability of our model to classify each type of fault. To use our model as an unsupervised classifier, the test data is passed through our trained model to compute the mean residual vector and latent state representation for each run. An unsupervised Gaussian mixture model is then fit to cluster the concatenated residuals and latent state. The number of clusters in the Gaussian mixture model is chosen automatically by iterating over a range of values and choosing the model with lowest BIC (Bayesian Information Criterion) score [5]. The BIC is a measure of how well a model fits a given dataset with an added penalty term for the number of parameters in the model, as shown in Equation 3.4:

$$\text{BIC} = k \ln n - 2 \ln \hat{L} \tag{3.4}$$

where k is the number of fitted parameters in the model, $\hat{L}$ is the loss of the fitted model, and n is the number of data points in the training set. Assuming the test data contains both faulty and normal data, not all clusters will be associated with faults. Separating true fault clusters can be accomplished by one of three methods. Firstly, clusters associated with faults are likely to have a significantly higher anomaly score than usual. Secondly, if any known faults exist in the test data these few labels can be used to determine the meaning of different clusters. Finally clusters representing a healthy machine state will fall close to the empirical cluster centers found during the training.

3.3 Preprocessing Steps

Before passing the raw data to our neural network model we perform two preprocessing steps. The first is shifting the raw sensor data to the frequency domain and the second is normalization. For the first step, every channel is broken into 10 frequencies using the absolute value of the short term Fourier transform, equivalent to the square root of the PSD (Power Spectral Density). The PSD was then centered and scaled to zero mean and unit variance. The test and validation datasets were similarly transformed and normalized using the mean and standard deviation from the training dataset. The normalized PSD was then used as the input to our neural network.

3.4 Model Evaluation

To evaluate the performance of our model, we test its ability to detect faults across all of our datasets. For datasets with multiple types of faults, we tested the ability of the model to accurately classify different fault modes. The following sections outline how our models are tested and the evaluation metrics used to judge model performance.

3.4.1 Fault Detection

The primary metric we use to judge our model is the overall detection accuracy. For this purpose, on all datasets, the model is trained on approximately 70% of the available known good data. A further 10% of the known good data is used for threshold setting, and the remaining data including all instances of faults are used as our test data. The trained model is then used to calculate anomaly scores for all runs in the test dataset. All runs in the test dataset with an anomaly score higher than the threshold are classified as faults, and the remaining runs are classified as nominal. We evaluate our model by calculating the true positive rate, true negative rate, and overall accuracy. The theoretical effectiveness of our model to detect faults independent of the threshold setting method is measured using the area under the ROC (Receiver Operating Characteristic) curve, or AUC.

3.4.2 Baselines

To test how significantly the performance of our system is affected by the addition of the estimation network and the recurrent unit, we test our method against two baseline models with these pieces disabled. To isolate the effect of these pieces, besides the changes explicitly listed in this section, all hyperparameters concerning the network size, training time, and input pipeline are kept the same as our final model. The first baseline method is a DAGMM without a recurrent unit following the final encoding layer. Instead, the final layer of the encoder is used as the bottleneck. To this end, the final layer of the encoder was used to parameterize the normal distribution from which the latent state was sampled directly. This change makes the DAGMM stateless, such that the anomaly score assigned at any given moment in time is independent of all sensor data outside of the current input window. The second baseline model is a variational autoencoder without an estimation network. Instead of using the GMM clustering objective to calculate the anomaly score, this model uses the MSE loss from compression and reconstruction. Together these two baselines are used to establish how important the clustering and forecasting elements are to successful fault detection on each dataset.

3.4.3 Classification Accuracy

To evaluate the performance of the clustering method, we compare the ground truth labels to the predicted labels and use the NMI (Normalized Mutual Information) between the two sets of labels as our metric. The NMI is a measure of the agreement between two sets of labels that is invariant to permutations and falls within the range $[0, 1]$, with one corresponding to identical labels and zero indicating that there is no correlation between the two sets of labels [24]. The definition of the NMI is shown below in Equation 3.5, where Y is the vector of true class labels, C is the vector of predicted cluster labels, I is the mutual information, and H is the entropy.

$$\begin{aligned} H(X) &= -\sum_{x \in X} \times p(X = x) log(p(x = X)^{-1}) \\ I(Y; C) &= H(Y) - H(Y|C) \\ NMI(Y, C) &= \frac{2 \times I(Y; C)}{H(Y) + H(C)} \end{aligned} \tag{3.5}$$

3.4.4 Fault Detection With Hidden Information

During operation, some details of how a machine is being used may not be recorded electronically. For example, the material being cut by a milling machine is rarely tracked. In the event our model does not have access to this information, it will need to be inferred to accurately gauge how anomalous a given window of sensor data is. To evaluate the robustness of our model when missing information, on datasets with multiple run types, we also measure the detection accuracy without normalizing the anomaly scores by run type. We also test the ability of our model to classify runs with different system settings, using the same process as described in Section 3.4.3.

3.4.5 Diagnostics and Predictive Maintenance

In this section, we describe the methods we explore to extract diagnostic information from our model and a method we propose to predict faults in advance. The two goals of this method are to narrow down the moments in time and the channels most relevant to detecting a particular fault. This level of interpretability can help engineers determine the true cause of different faults and speed up repair efforts. To do so, we isolate channels and windows with the highest anomaly scores above the empirical average on runs that were labeled as faults. We then overlay the known good and faulty runs in the test dataset to see if clear differences in the signals could be seen. Our second goal is to search for evidence of system drift before a fault occurred that could potentially be used to improve detection rates should the same fault occur again, or to predict the remaining useful life such that predictive maintenance can be scheduled in advance. To accomplish this, we use the cluster centers computed as described in Section 3.2.3 representing the average latent state value and residual error on each channel for a given run type. The difference between a cluster center of known good runs and of known faults represents the direction of travel in latent space associated with a specific failure mode. We look to see if the probability of faults occurring increases consistently along this dimension and if drift in that direction can be detected prior to a fault occurring.

Chapter 4

Experiment Design and Data Sets

In this chapter, we outline our data collection methods and describe the various fault detection tasks our system is tested against in more detail. To test our system's ability to generalize, we collect a range of datasets from different types of manufacturing equipment. Across all datasets, features such as the sampling rate, number of sensor channels, and amount of training data vary greatly, as do the types of sensors in each experiment. For every dataset, we test the accuracy with which the fault system can separate nominal points from faults. For datasets with multiple fault types, we also test the classification accuracy of the system. For reference, a summary of the characteristics of each dataset is shown in Table 4.1.

Table 4.1: Fault Detection Datasets

Dataset	Sampling Freq. (Hz)	No. of Channels	Input Signals
Synthetic	Arbitrary	-	-
Bearing	1200	2	Vibration
Mill (Chiron)	1000	3	Vibration
Mill (NASA)	250	6	Current Acoustic Vibration
Mill (Nazha)	200	3	Vibration
Etcher (ADI)	2	30	Mixed

4.1 Synthetic Generator

To guide the early development of our model architecture we developed a synthetic data generator. Our generator models a target system as a Markov chain, where each node represents a discrete state for the machine. Systems can be initialized with an arbitrary number of sensor channels, machine states, and transition probabilities between states. At each time step, the current state determines the signal produced on every sensor channel. Signals are produced as linear combinations of sine waves with amplitudes and phases determined by the state, with additive Gaussian noise. Our early architectures are tested on their ability to detect faults represented as states which are not present in the training data. The interpretability of the models was also tested by measuring our ability to classify the machine state using the compressed representation in the latent state and the ability of the system to classify the channels associated with each fault.

4.2 Bearing

In this dataset collected by Case Western Reserve University [21], a fan supported by circular ball bearings was spun at various speeds using a twohorse power motor, as shown in Figure 4-1. Small defects were introduced to the inner raceway, outer raceway, or balls in the bearing using electro-discharge machining. Vibration data was collected at 12,000 Hz from accelerometers at both the drive end and fan end of the motor housing. The data is divided into runs approximately 120,000 samples long. There are four runs with an undamaged bearing and four runs with each of the three fault types. The goal of our task is to successfully detect which run used a damaged bearing and classify where in the ball bearing the fault is located.

‘

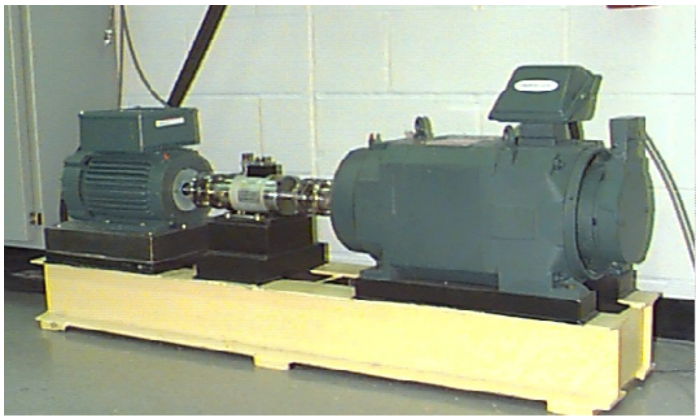
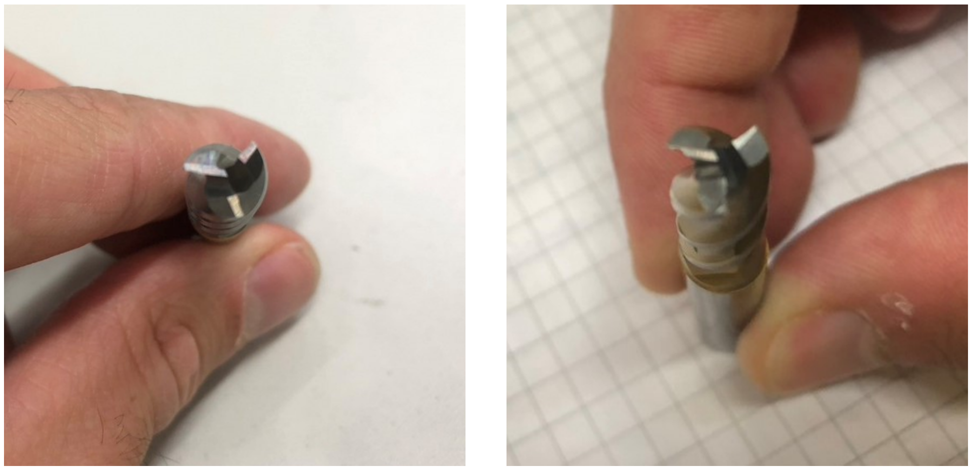


Figure 4-1: Case Western bearing experimental setup [21].

4.3 Mill

We consider data from three different milling machines, including two collecting in collaboration with HARTING, and one from a public repository by NASA. In two cases the fault being detected is damage to the cutting tool as shown in Figure 4-2. Over time chips form in the cutting edge, and if left unchecked, these can eventually lead to the tool snapping. In the final case, the cooling system was switched off for the final few cutting runs. The details of each dataset are discussed further below.



(a) Healthy Cutting Tool (b) Damaged Tool

Figure 4-2: New vs. worn cutting tool. Chips can be seen in the tip of the blade.

4.3.1 Chiron Mill

In this fault detection dataset, a Chiron milling machine was used to drill through steel plates at various speeds with either a new or worn tool. Three different cutting speeds were tested with each trial being run twice for a total of 12 runs, as shown in Table 4.2. Three axis accelerometer data was recorded at 1000 Hz from a sensor placed above the cutting tool.

Table 4.2: Mill Chiron Experiments

		Cutting Speed					
		700 mm/min		950 mm/min		1200 mm/min	
Tool	New	Exp 1.1	Exp 1.2	Exp 2.1	Exp 2.2	Exp 3.1	Exp 3.2
Condition	Used	Exp 4.1	Exp 4.2	Exp 5.1	Exp. 5.2	Exp 6.1	Exp. 6.2

The goal of this fault detection task is to see if the model can separate out the drilling runs with a worn tool, after training on cutting runs with the new tool. Given the small number of runs available to us, we run a k-fold evaluation, training the model on every possible combination of four new tool runs and testing on the remaining eight runs. To compare anomaly scores across trials, the anomaly scores are scaled to have a standard deviation of one and shifted such that zero represented the highest anomaly score given to a run with a new tool.

4.3.2 Nazha Mill

In these experiments, a fault was induced in a milling machine by switching off the cooling system. Our model is trained using three axis vibration data from 50 cutting runs and tested on another 50 runs including 10 faults. Data was sampled at 2000 Hz.

Table 4.3: NASA Mill Experiments

Case	Depth of Cut (mm)	Feed Rate (mm/m)	Material
1	1.5	0.5	Cast Iron
2	0.75	0.5	Cast Iron
3	0.75	0.25	Cast Iron
4	1.5	0.25	Cast Iron
5	1.5	0.5	Steel
6	0.75	0.5	Steel
7	0.75	0.25	Steel
8	1.5	0.25	Steel

4.3.3 NASA Mill

In this publicly available data set provided by the NASA Ames Prognostics Data Repository [1], a milling machine was repeatedly used to drill through metal plates under varying conditions, as shown in Table 4.3.

Measurements of tool wear (VB) were taken between runs by using a microscope to measure the distance in millimeters between the cutting edge and the end of the abrasive wear on the tool, as shown in Figure 4-3. These readings provide our first continuous measure of degradation to the health of the monitored equipment, in contrast to the binary fault labels in the other data sets. Our fault detection system is trained on 70% of the available runs with a VB below .3mm and tested on the remaining runs. To judge performance as well as measuring the accuracy of the binary prediction, we also explore the regression loss, using the system to predict tool wear from both the latent state and the anomaly score as a supervised modeling problem. In real world applications, slow degradation in performance is far more common than sudden failures. In these tests, we explore adapting our model for this purpose and how it may be used to predict upcoming faults.

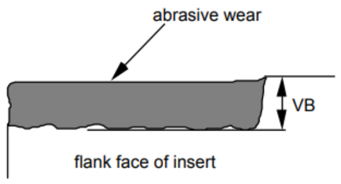


Figure 4-3: Measure of tool wear on the cutting edge of the mill insert [1].

4.4 ADI Etcher

This dataset contains internal sensor readings from a Lam plasma etcher that was used in a production line by ADI. This fault detection dataset was originally gathered and analyzed in the masters thesis of three previous authors, O. Mahklouk [23], H. He [15], and T. Chen [7]. The dataset contains 4000 total wafer etches split between two different wafer recipes, labeled recipe 920 and recipe 945. During production, a fault occurred which caused approximately 700 of the etched wafers to fail an electronic test performed afterward. Each run contains readings from 30 separate channels, including sensors recording voltages, pressures, temperatures, and more from throughout the machine, a sample of which are shown in Figure 4-4. Each sensor channel was sampled at 2Hz, and runs typically contain either 300 or 600 samples each, depending on the recipe.

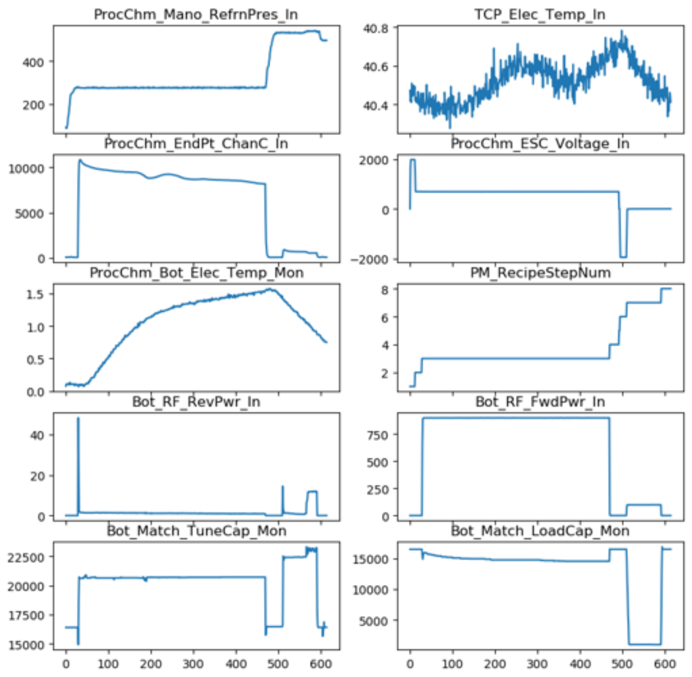


Figure 4-4: Sample of channels from the ADI etcher dataset.

On this data set, we test the ability of our system to detect which wafers would fail the e-test based on the sensor readings. During both training and testing, our system is not given access to the recipe being run, and we use this to further test the ability of our system to classify different run types. Our system is trained on 2400 nominal runs, using a further hundred runs for threshold setting, and tested on the remaining 1500 runs which contain an almost even split between faulty and nominal.

As the largest dataset available to us and the only one containing an organically occurring fault while the machine was actively being used for production in industry, this is also the dataset on which we seek to detect evidence of system drift prior to a fault.

Chapter 5

Experimental Results

In this chapter, we outline our system tests and experimental results. Results are organized by the type of machine being monitored. For every dataset, we test the overall detection accuracy for our model and our two baselines. For datasets with multiple fault types, we further measure the unsupervised classification accuracy. Finally, for datasets with multiple types of runs, such as varying etch recipes or milling speeds, we further break down our results by run type. The final fault detection accuracy of our system across all datasets is shown in Table 5.1.

Table 5.1: Accuracy Scores Across All Datasets

Dataset	VAE	DAGMM	RDAGMM
Bearing	1.00	1.00	1.00
Mill (Chiron)	0.93	**0.99**	**0.99**
Mill (NASA)	0.73	0.77	**0.82**
Mill (Nazha)	**0.95**	**0.95**	0.90
Etcher (ADI)	0.77	0.80	**0.81**

5.1 Bearing

The first set of results are based on the dataset collected by Case Western Reserve University [21], as summarized in Section 4.2. In this dataset, a fan supported by circular ball bearings was spun at various speeds using a two horsepower motor. Small faults were introduced to either the inner raceway, outer raceway, or balls in the bearing using electro-discharge machining. The goals of our system in this task is to successfully detect when the bearing has been damaged, and to segment runs with faults in different locations to separate fault classes.

5.1.1 Detection Accuracy

In the fault detection test, our model and our two baselines are able to perfectly classify all runs with a damaged bearing. As shown in Figure 5-1, even when runs are broken down in smaller fragments of only 0.1s, two orders of magnitude (noting the log scale on this figure) separate the anomaly scores for windows having a healthy versus a damaged bearing.

Table 5.2: Bearing Fault Detection Accuracy

	VAE	DAGMM	RDAGMM
Accuracy	1.00	1.00	1.00
False Positive Rate	0.00	0.00	0.00
True Positive Rate	1.00	1.00	1.00
ROC Area Under Curve	1.00	1.00	1.00

5.1.2 Fault Classification

Our unsupervised classification algorithm is similarly able to perfectly classify the runs by the type of fault. The refitted GMM finds the correct number of clusters in the test set and provides an equivalent labeling with an NMI of 1, as shown in Figure 5-3. Figure 5-2 shows the anomaly scores for each moment in time, with color denoting the type of fault. Visually there is a notable difference between runs with a healthy bearing, damage to the raceway, or damage to the balls.

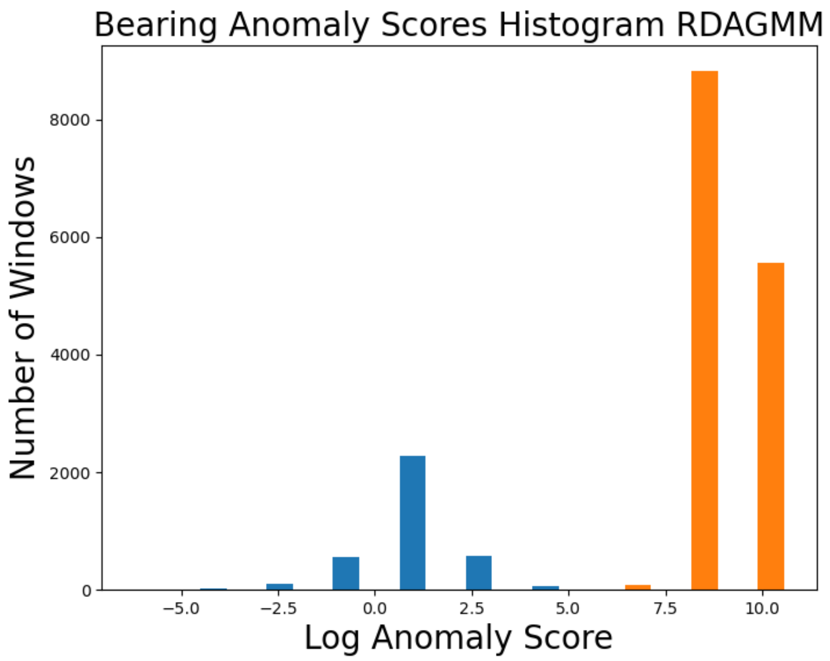


Figure 5-1: Histogram of log anomaly scores for each 0.1s window of sensor data, as calculated by the RDAGMM. Blue and orange indicate that the sensor data was recorded with a healthy or damaged bearing, respectively.

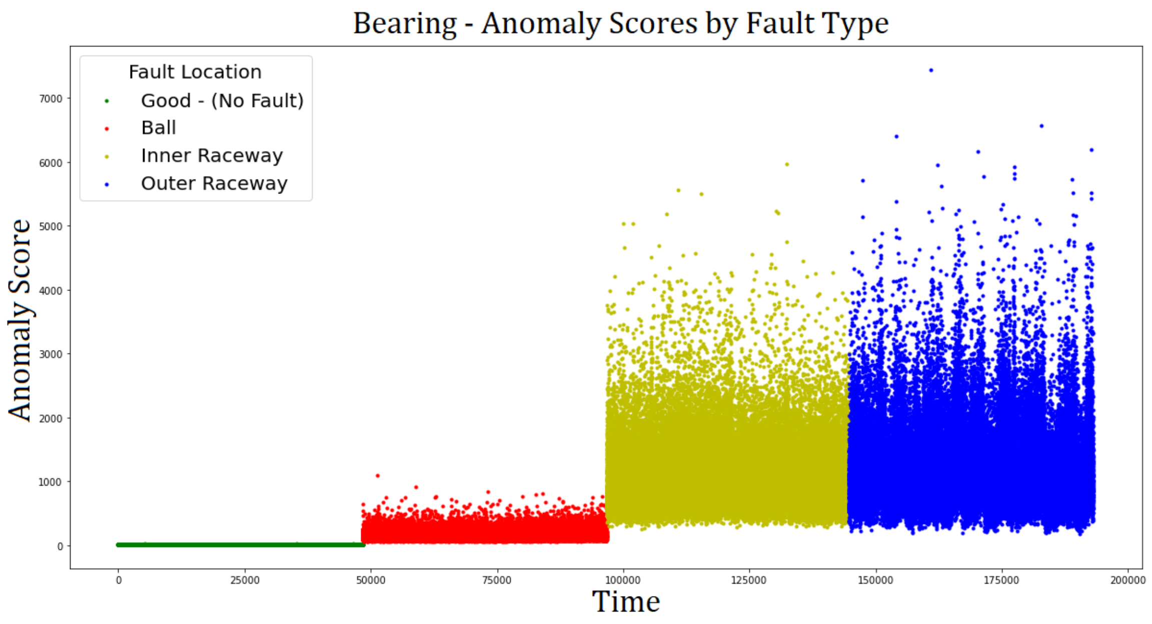


Figure 5-2: Anomaly score calculated for every window of time in our test dataset by the RDAGMM. Color indicates the true labels for the type of fault induced on the bearing. Green indicates the runs with an undamaged bearing.

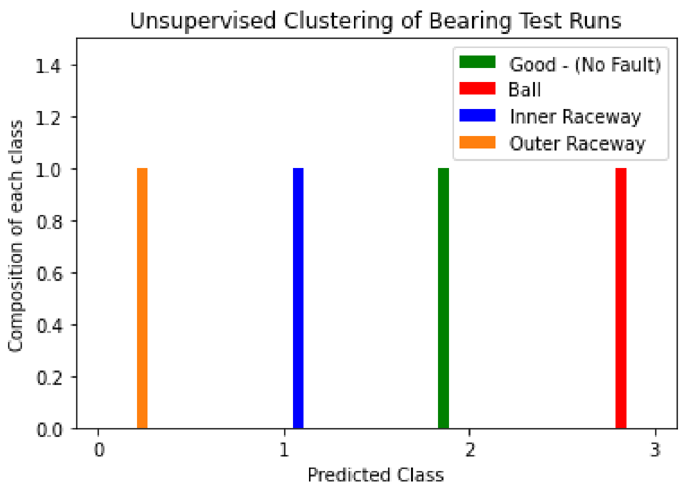


Figure 5-3: Unsupervised class labels assigned to each run in the test dataset after refitting a GMM with unknown number of classes. Runs are perfectly separated by fault type.

5.2 Mill Chiron

In this task, we test the ability of our system to separate milling runs with either a new or damaged tool, based on the Chiron mill data summarized in Section 4.3.1. Due to the small number of runs available, a k-fold evaluation is run to test the performance of our system. In each iteration, the model is trained on six cutting runs with a new tool, and tested on the remaining ten runs, two with a new tool and eight with a damaged tool. This evaluation loop is run for all 15 possible train/test split combinations.

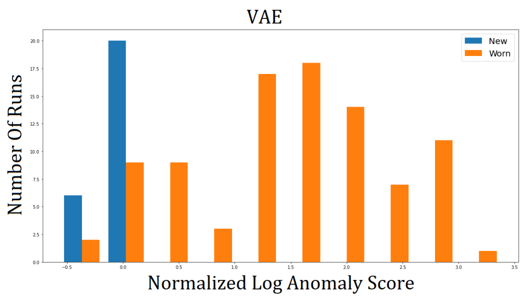
5.2.1 Detection Accuracy

For all models, the majority of worn tool cutting runs are separable from those with the new tool, using the anomaly score. As shown in Table 5.3, the DAGMM and RDAGMM both outperform the vanilla autoencoder, with only a single false negative and no false positives.

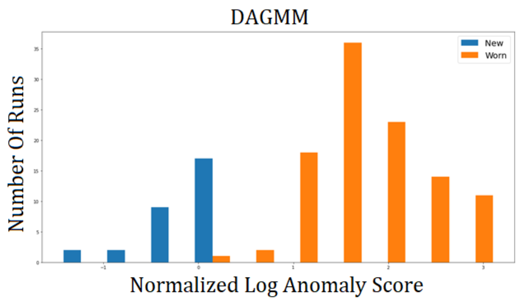
Table 5.3: Mill Chiron Experiments

	VAE	DAGMM	RDAGMM
Accuracy	0.93	**0.99**	**0.99**
False Positive Rate	0.00	0.00	0.00
True Positive Rate	0.82	**0.99**	**0.99**

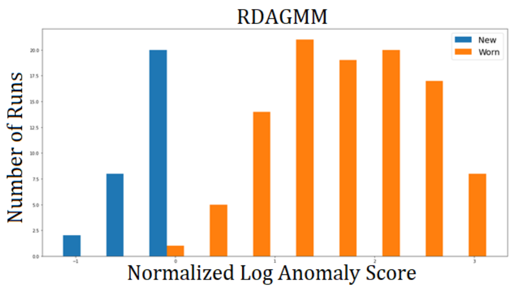
The distribution of anomaly scores for our model and both baselines is shown in Figure 5-4. For the latter two models, the only misclassification occurs when the model is trained exclusively on the lower speed runs. With no high speed runs present in the training set, these runs are assigned a significantly higher anomaly score during test evaluation.



(a) VAE



(b) DAGMM



(c) RDAGMM

Figure 5-4: Histograms of k-fold cross validation results on the Chiron Mill dataset with three model architectures in order of increasing complexity. Plotted on the X axis is normalized log anomaly score, and the Y axis represents the total number of trials which fall in this range. Runs with a new tool are shown in blue and runs with a damaged tool are shown in orange.

5.3 Mill Nazha

In these experiments using the Nazha mill dataset as summarized in Section 4.3.2, a fault was induced in the milling machine by switching off the cooling system. Similar to the previous task, our model is trained using three-axis vibration data from fifty cutting runs with the milling machine operating normally. It is then tested on forty additional runs, including ten faults where the cooling system was switched off.

5.3.1 Detection Accuracy

At the two-sigma threshold, our model and the two baselines successfully detect every fault in the evaluation dataset, with all models scoring above 90% as summarized in Table 5.4. Our model has the highest false positive rate, with three additional misclassified runs, leading to the lowest overall accuracy overall.

Table 5.4: Mill Nazha Fault Detection Accuracy

	VAE	DAGMM	RDAGMM
Accuracy	**0.95**	**0.95**	0.90
False Positive Rate	**0.07**	**0.07**	0.13
True Positive Rate	1.00	1.00	1.00
ROC Area Under Curve	0.99	0.99	0.99

Independent of thresholding, all three models perform equivalently as classifiers, as shown by the ROC curves in Figure 5-5, with an area under each curve of 0.99. The full distributions of anomaly scores for our model and both baselines are shown in Figure 5-6.

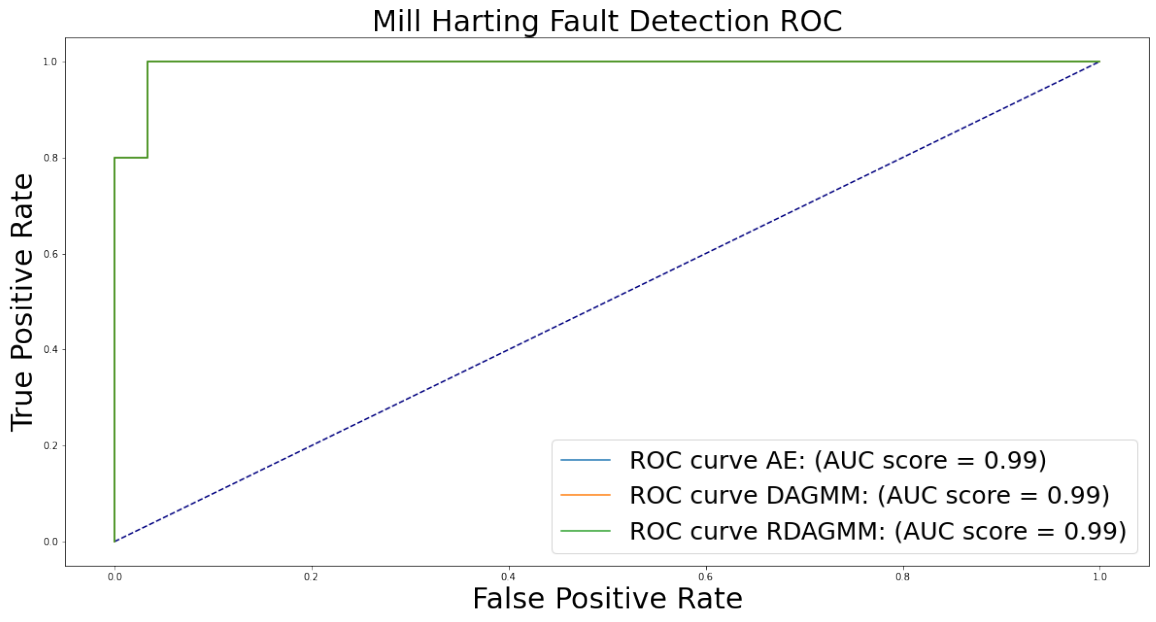


Figure 5-5: Mill Nazha ROC curves. Curves for all three models overlap perfectly.

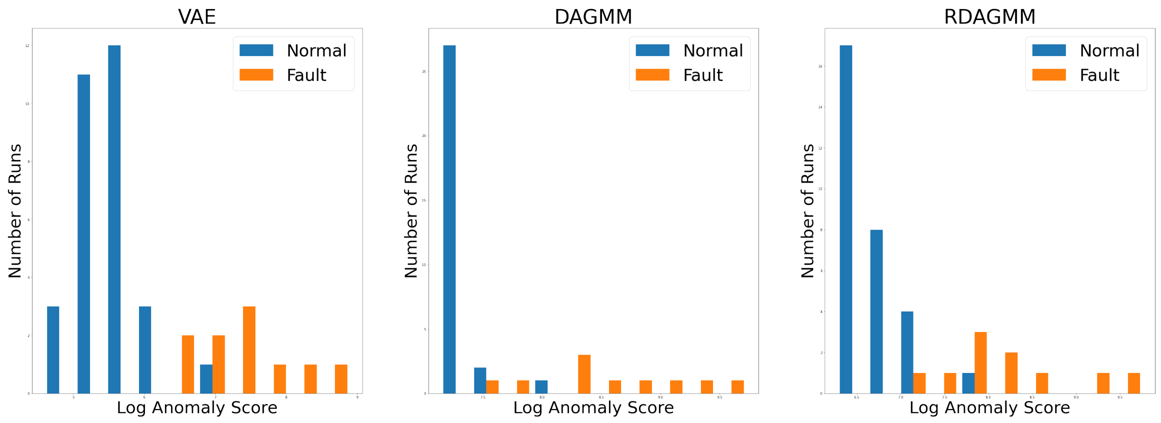


Figure 5-6: Comparison of anomaly scores generated by all three models for test runs in the Nazha Mill dataset. Faulty runs with the cooling system shut off are shown in orange. Nominal runs are shown in blue.

5.4 Mill NASA

In this task, we train our system on milling runs from the NASA mill dataset summarized in Section 4.3.3. The model is trained on 80% of runs with measured tool wear below 0.3mm. The remaining runs are used as the test dataset and we evaluate the ability of the model to distinguish runs with a more damaged tool. The system is provided acoustic, vibration, and motor current readings. The material being cut, motor speed, and depth of cut were all changed between runs. With these three variables taking two values each, there are a total of eight run types. Anomaly scores were normalized independently for each run type. We further test the fault detection accuracy if the material and machine settings are not known and the ability of our model to classify these different run types.

5.4.1 Detection Accuracy

This section details the ability of our model to detect faults if the depth of cut, material, and feed rate are all known at the time of evaluation. Table 5.5 shows the fault detection rates for our model and benchmarks when setting thresholds for each run type independently. Our model outperforms both benchmarks with a final accuracy of 82%. Our model also outperformed both baselines independent of the threshold, as shown by the ROC curves in Figure 5-7.

Table 5.5: NASA Mill Detection Accuracy - Normalized Run Type

	VAE	DAGMM	RDAGMM
Accuracy	0.73	0.77	**0.82**
False Positive Rate	0.10	**0.00**	**0.00**
True Positive Rate	0.73	0.77	**0.85**
ROC Area Under Curve	0.88	0.92	**0.95**

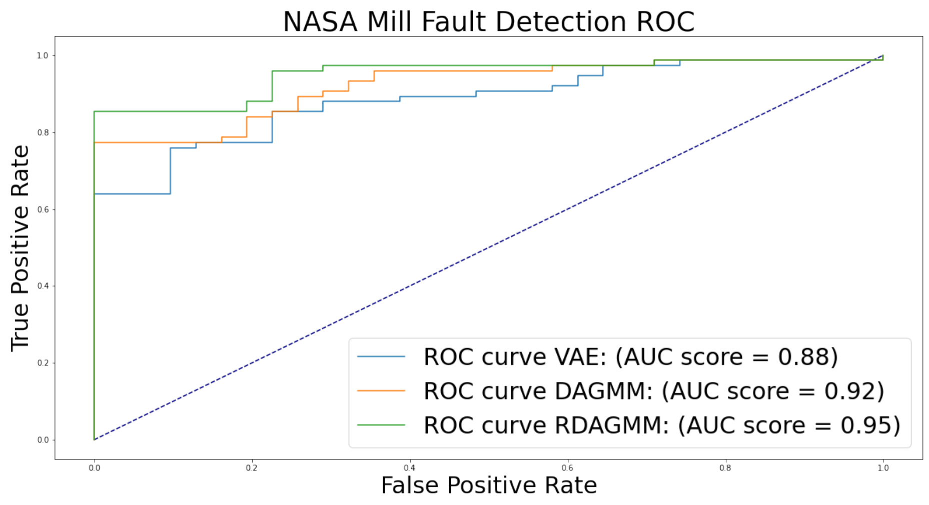


Figure 5-7: NASA Mill ROC - Normalized by run type.

5.4.2 Detection Accuracy With Hidden Information

Without normalizing anomaly scores by run type, our model and both baselines perform substantially worse. Our model still outperforms the two benchmarks as shown in Table 5.6, with a final detection accuracy of 0.67%, a 14% decline from the previous results. Independent of thresholding, our model also outperforms the two benchmarks as shown by the ROC curves in Figure 5-8, but the AUC is similarly reduced across all three models.

Table 5.6: NASA Mill Detection Accuracy - Hidden Variables

	VAE	DAGMM	RDAGMM
Accuracy	0.59	0.62	**0.67**
False Positive Rate	0.19	0.19	0.19
True Positive Rate	0.59	0.62	**0.65**
ROC Area Under Curve	0.78	0.77	**0.80**

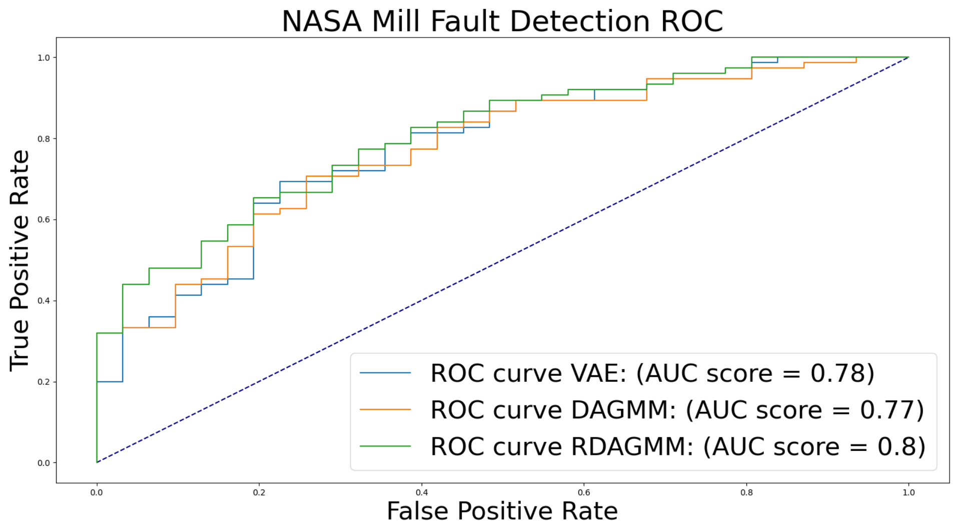


Figure 5-8: NASA Mill ROC curves with hidden variables.

This decline in performance is due to the fact, as shown by the unnormalized anomaly scores in Figure 5-9, that while a roughly linear relationship can be seen between the tool wear and the log anomaly score for runs of the same type, the range of anomaly scores assigned to runs of each type varies dramatically. For example, focusing on the top two plots of Figure 5-9, on runs where iron is being cut to a depth of 1.5mm with a feed rate of 0.5, the optimal threshold to maximize detection accuracy is around an NLL of 3.25. By contrast, when the depth of cut is reduced to 0.75mm, this threshold is higher than the vast majority of faults. Setting a single threshold for all run types becomes impossible.

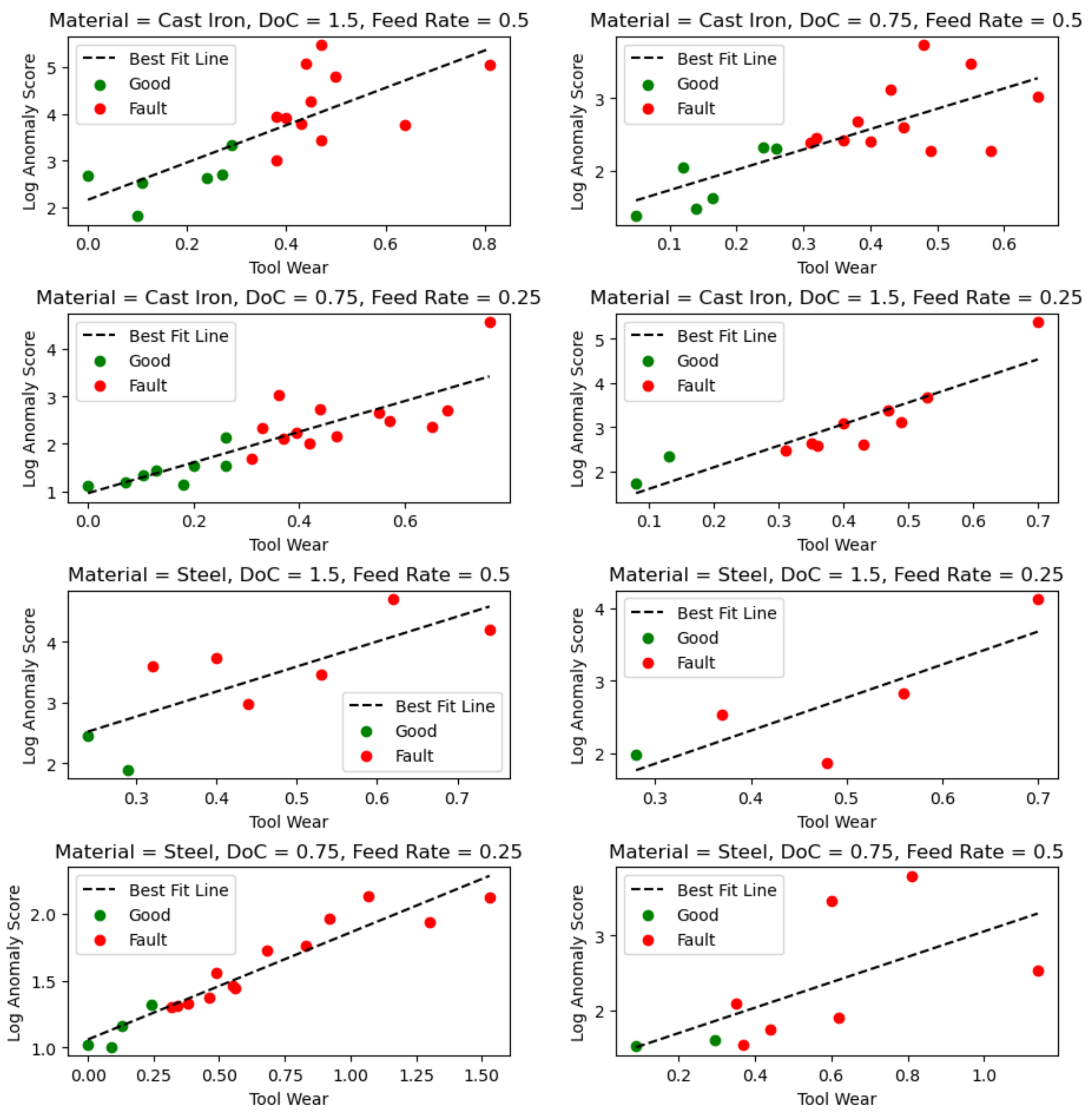


Figure 5-9: NASA Mill Anomaly Scores. Each plot represents runs with a unique combination of machine settings and material being cut. The log anomaly scores are plotted against the measured tool wear for each run.

5.4.3 Clustering

The reduction in accuracy seen above is mirrored by the results of our unsupervised clustering algorithm. In this instance, our system failed to meaningfully label different run types. As shown in Figure 5-10, our system groups runs into 5 clusters with an NMI to the true labels of 0.35.

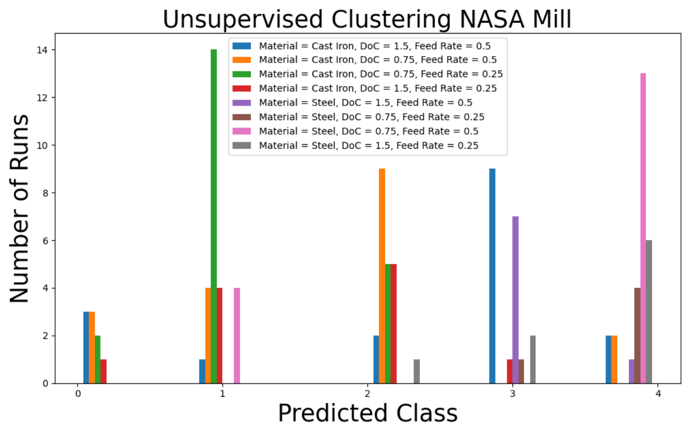


Figure 5-10: NASA Mill unsupervised clustering results. Color denotes the true labels for each run, and the x axis shows the predicted cluster label.

5.5 Etcher ADI

In the data-set provided by ADI as summarized in Section 4.4, our model has access to all internal sensors and monitors within a plasma etcher during operation; a total of 30 channels including temperatures, pressures, voltages, and gas concentrations. The etcher was being used to execute two different wafer recipes titled “recipe 920” and “recipe 945”. After approximately 200 runs, a fault occurred which resulted in a long sequence of failed etches before the machine was repaired and normal operation resumed. In this trial, we train our models on 1000 successful etching runs and then evaluated the ability of our model and the baseline approaches to classify the failed etches by anomaly score.

5.5.1 Detection Accuracy

The results of our model detection accuracy compared to our two baselines are summarized in Table 5.7. As shown, our model slightly outperforms the VAE and DAGMM, with a two-sigma threshold scoring the highest accuracy and TPR as well as the lowest FPR.

Table 5.7: Etcher MIT Results

	VAE	DAGMM	RDAGMM
Accuracy	0.77	0.80	**0.81**
False Positive Rate	0.18	**0.12**	**0.12**
True Positive Rate	0.72	0.72	**0.74**
ROC Area Under Curve	**0.90**	0.88	**0.90**

Figure 5-11 shows the comparative efficacy of our model as a fault classifier at all thresholds compared to our two baseline models. Our model outperforms the VAE at low false positive rates and underperforms at false positive rates above 40%.

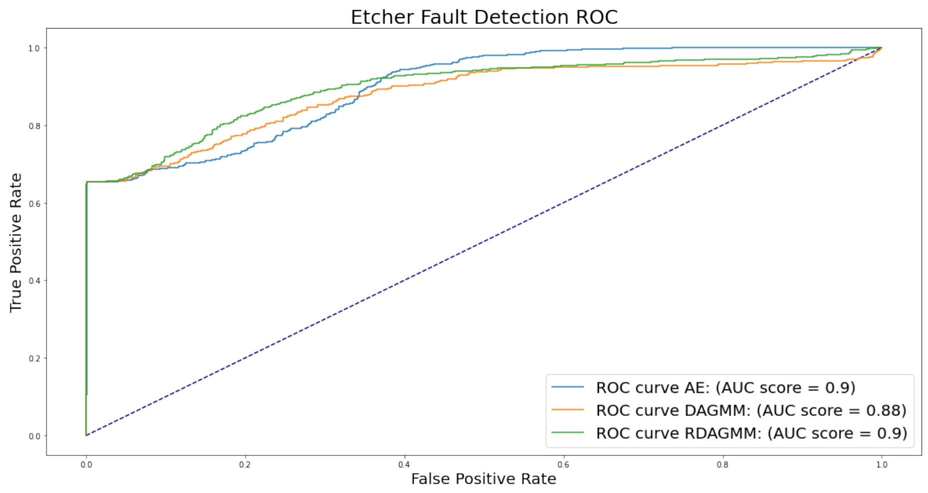


Figure 5-11: ADI etcher ROC curves.

As shown in Figure 5-12, approximately 65% of the failed runs can be immediately separated from the successful runs, with anomaly scores that are orders of magnitude larger than the average, noting the log score in the plot. The remaining failed runs are mixed in with successful etches.

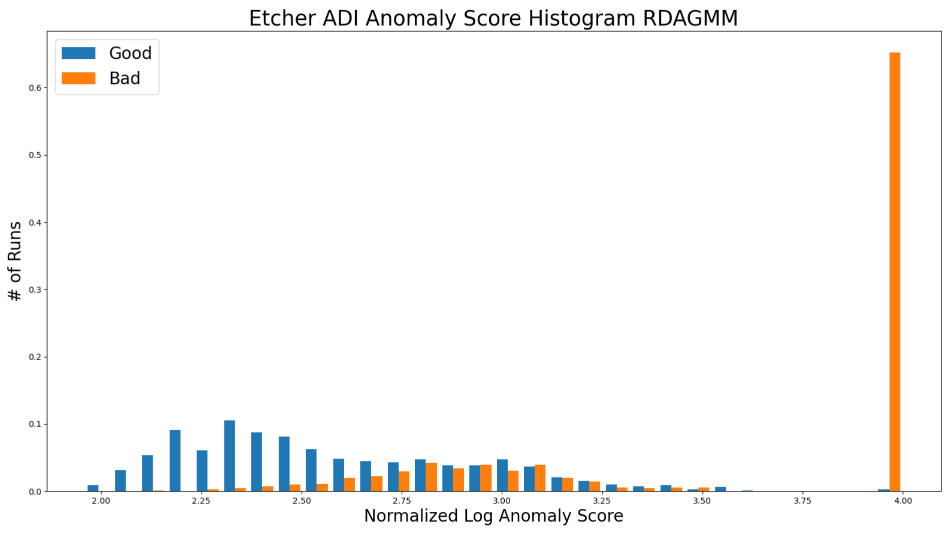
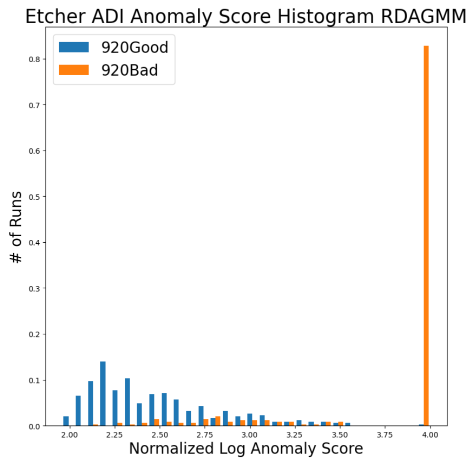
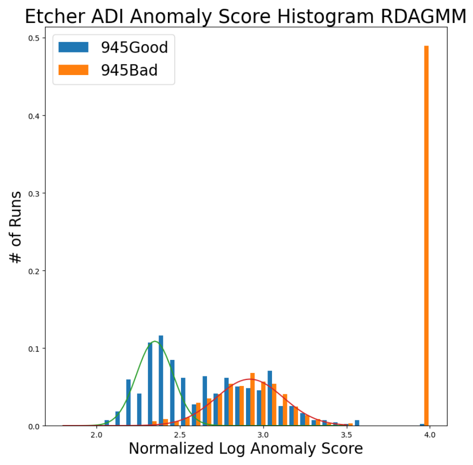


Figure 5-12: Histogram of log anomaly scores for each run in the ADI etch test dataset. Successful etches are shown in blue and failed runs are shown in orange.

Notably, the vast majority of detection errors made by our system are on the runs following the 920 recipe. Figure 5-13 shows the log anomaly scores for every run separated by recipe, and it can be seen that the responses are very different. For the 920 recipe, our model performs substantially better, with a detection accuracy of 92% and 83% of runs being cleanly separable, with an order of magnitude jump in the anomaly score. By contrast, on runs following recipe 945 our model could only separate out 50% of the failed runs in the same way, with the decision boundaries as shown. The remaining runs with a log anomaly score of less than 3.5 roughly follow a sum of two Gaussian distributions; the first, centered around a log anomaly score of 2.3, generating only successful runs, and the second, centered around 2.9, generating an even mix of failed and successful runs. This pattern is reflected in the results of our classification algorithm as discussed next in Section 5.5.2.



(a) 920 recipe.



(b) 945 recipe.

Figure 5-13: Histogram of anomaly scores assigned by our model to runs of each recipe.

5.5.2 Clustering

This section shows the results of our unsupervised classification algorithm. It is unknown whether this etch dataset contains multiple fault types that led to the failed etches. Our classification algorithm groups the runs in our test dataset into five clusters; Figure 5-14 shows the composition of the five clusters found by our system using the dataset provided true labels.

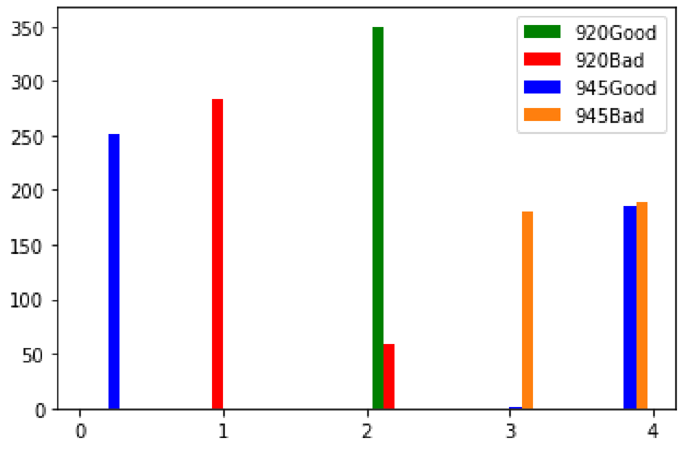


Figure 5-14: Unsupervised clustering of the ADI etcher runs. Color denotes the true labels, and the x axis shows the label assigned by our system.

As pictured in Figure 5-14, the first four classes almost perfectly segment the runs by both recipe and fault label, with only a single false positive in class 3 and 53 false negatives in class 2, out of a total of 1150 runs in these classes. The final class contains a 50/50 split of successful and failed runs, all from the 945 recipe. The NMI score for this labeling is 0.75.

5.5.3 Drift Detection and Fault Diagnosis

In this section, we explore how our model could potentially be used to build diagnostic tools and predict faults in advance of failures. The ADI etcher dataset is used for this purpose, as the only dataset recorded from a machine actively being used in industry that encountered a fault organically. The first goal of these tests is to use our model to pinpoint or connect meaningful differences between failed and successful etches to specific sensor channels and moments in time. The second goal is to find temporal trends prior to a fault that could be used to predict the remaining useful life and preemptively schedule maintenance. Figure 5-15 shows visually how runs are encoded by the RDAGMM in latent space. For both run types, a clear increase in fault density can be seen as the run encodings transition from the bottom left to the bottom right.

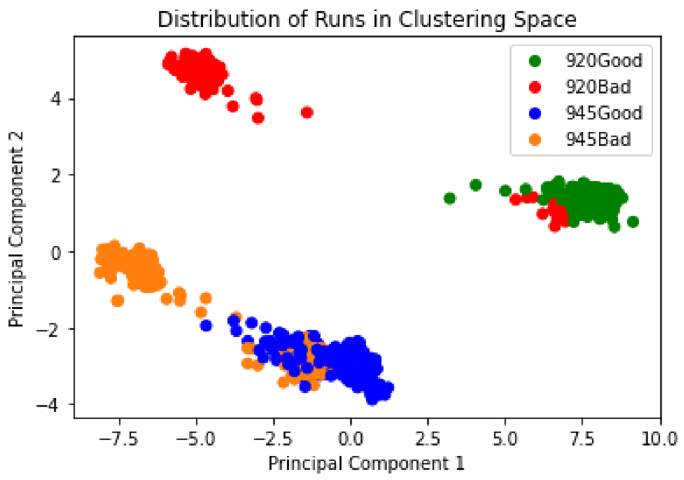


Figure 5-15: Visualization of RDAGMM latent space values for each ADI etching run using first two principal components.

The distribution of run types and assigned class labels are shown chronologically in Figure 5-16. Notably, there are no misclassifications after the machine was repaired around run number 1000. All false positives occur prior to the first failed etch. Additionally, the recipe 945 runs before the fault began are all given a fifth class label containing a number of faults as well. This class corresponds to class label 4 shown previously in Figure 5-14. Asymmetry in classification rates and labelings before and after the fault is consistent with the hypothesis that there was detectable degradation in the machine's health proceeding wafers failing to pass a downstream e-test.

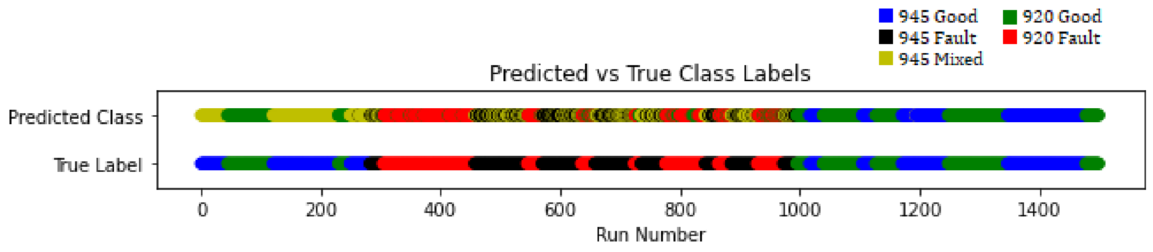


Figure 5-16: True labels for etcher runs and predicted GMM class labels shown chronologically.

5.6 Run Time

In this section, we comment on the computational resources needed for training and inference. All training was performed using a single Nvidia 2070 GPU. Inference was performed using an Intel i7 CPU. Training on our largest dataset, the ADI etcher readings, requires approximately 4 hours. The inference steps require 0.24s per run. In all cases, both training and inference ran substantially faster than the duration of the time series being analyzed, opening the possibility for real time training and inference.

Chapter 6

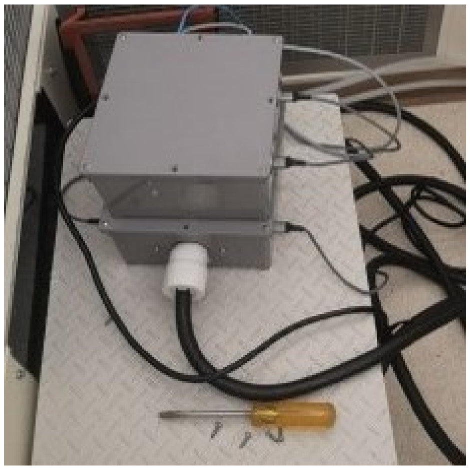
Ongoing Projects

In this chapter, we describe preliminary explorations of improvements to the RDAGMM architecture, and present results using the recording platform we developed to acquire fault detection datasets for further development of the system.

6.1 MIT Etcher Dataset

Key limitations of the datasets used in this work are the artificial nature of the faults, the lack of repeat faults, and the lack of data leading up to the failure. Ideally, to test the viability of our system in a real world setting, particularly the diagnostic and predictive maintenance aspects, would require long term readings from manufacturing equipment in active use. To this end, in partnership with MIT.nano and the Microsystems Technology Laboratories, we have developed and installed a recording platform in the Integrated Circuits Laboratory monitoring a Lam 590 plasma etcher, shown in Figure 6-1b. The plasma etcher is frequently used by a variety of researchers, private groups, and for teaching purposes.

Our recording platform connects between the etcher and the wall power supply and collects three phase current and voltage going into the machine. As shown in Figure 6-1a, the recording platform consists of two boxes, the first containing a twelve volt power supply and a Raspberry Pi 4. The second contains the current and voltage sensors as well as two high speed analog-to-digital converters. The sensors are


(a) Sensor Boxes

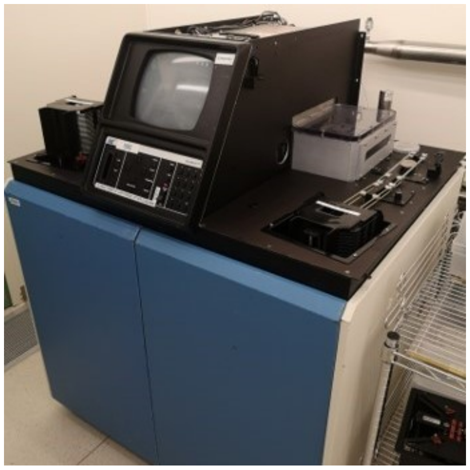

(b) Lam 590 Plasma Etcher

Figure 6-1: Etcher power measurement experimental setup.

capable of sampling at 12,000 Hz to capture high frequency transients. The sensors run continuously, and are monitored by a software trigger that automatically begins recording when it detects that the machine is in active use. When the system detects the etcher is not in use, files are uploaded to a network drive for long term storage. While machine use dropped considerably due to the pandemic, in the six months since we began recording, we have accumulated approximately 150 etching runs, a sample of which is shown in Figure 6-2.

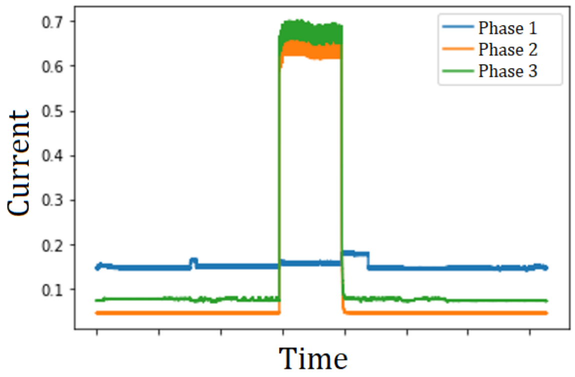


Figure 6-2: Three phase current signal - single etch.

During this time the etcher has suffered from one fault. An electrode gap error on August 24, 2020, caused the machine to jam and triggered routine maintenance, during which it was discovered that there was a vacuum leak in the etch chamber. It

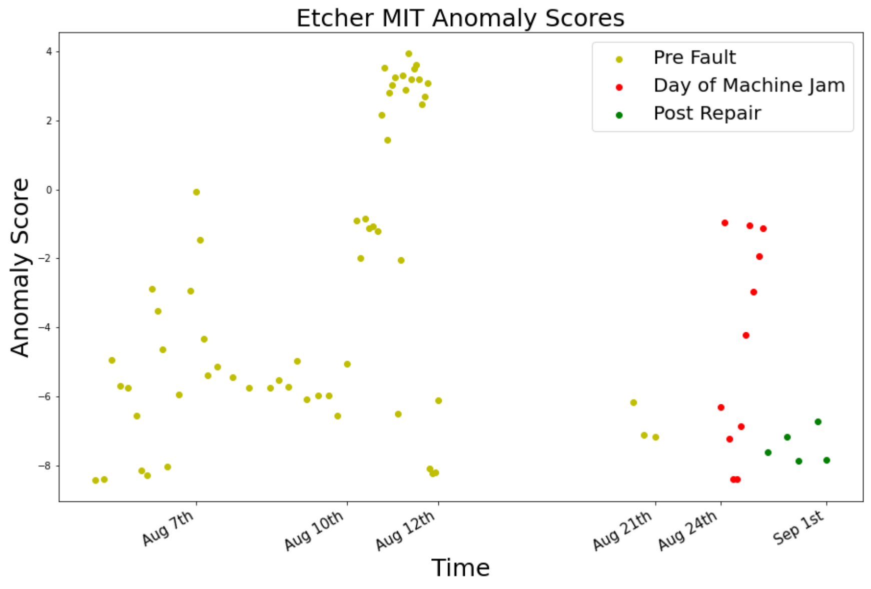
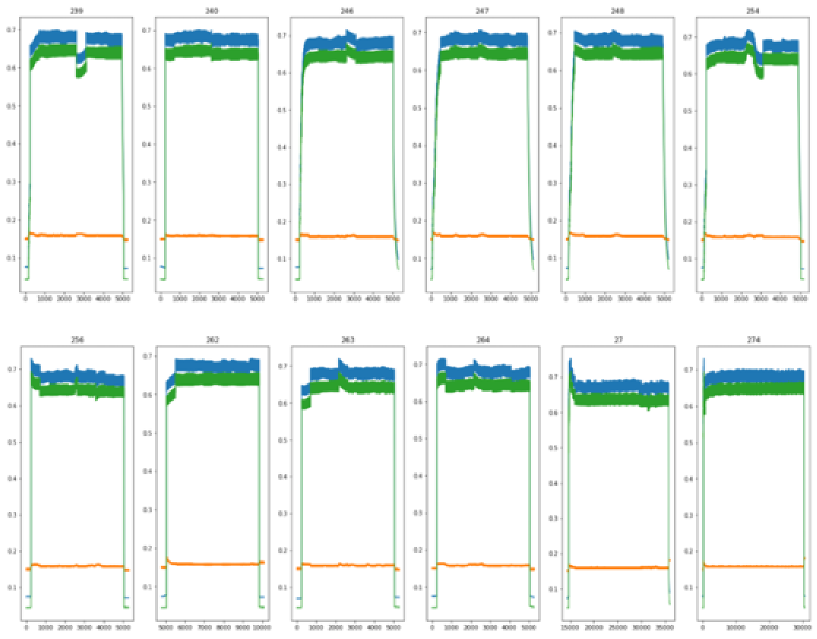


Figure 6-3: Anomaly scores assigned to runs during the month of August. A sharp spike in the residual score can be seen on August 12th after which the machine went unused for 10 days. On August 24th the anomaly score begins spiking again leading to a machine jam as shown in red. Following the repairs, the anomaly score drops again as shown in green.

is unknown when the leak began. Figure 6-3 shows the anomaly scores assigned to the runs around that period of time. Immediately prior to the fault occurring we see the anomaly score spike, and then fall significantly following repairs. Similar spikes can be seen earlier in the month as well, with no fault being reported. To generate these results our system is trained on 50 etching runs from the month prior to the fault.

For all of these runs, the etch recipe and other machine settings are unknown. Tests are ongoing to determine if spikes in the anomaly score are correlated with faults, or possibly are being caused by novel run settings and other factors. Given the difficulty of this problem, we expect significantly more data will be required to properly train the system. We are, however, cautiously optimistic that ultimately faults will be detectable solely through supply power measurements. Figure 6-4a and Figure 6-4b show a sample of etching runs from before and after maintenance was performed due to the jam. Visually the current draw appears far more stable following the repairs, indicating that changes in the machine health cause detectable changes in the power signals.


(a) Prior to machine jam.

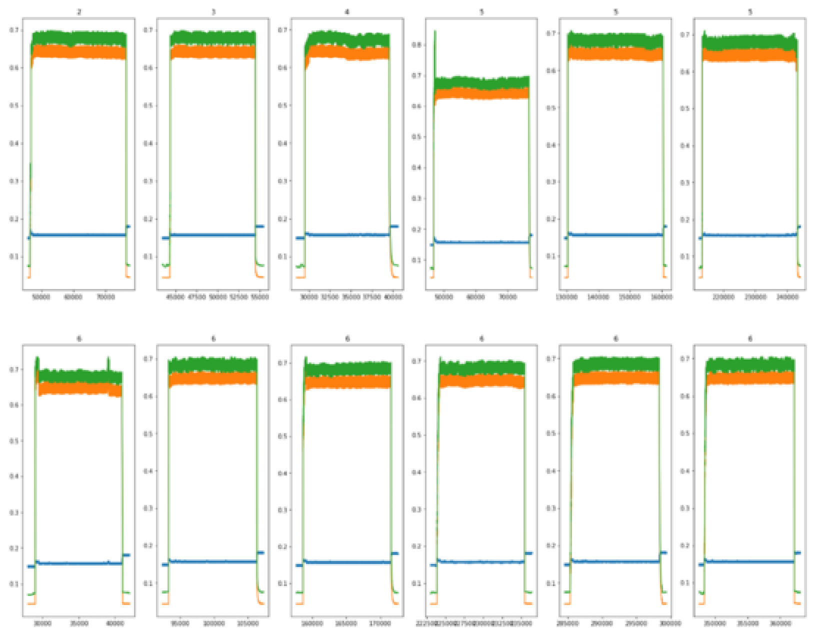
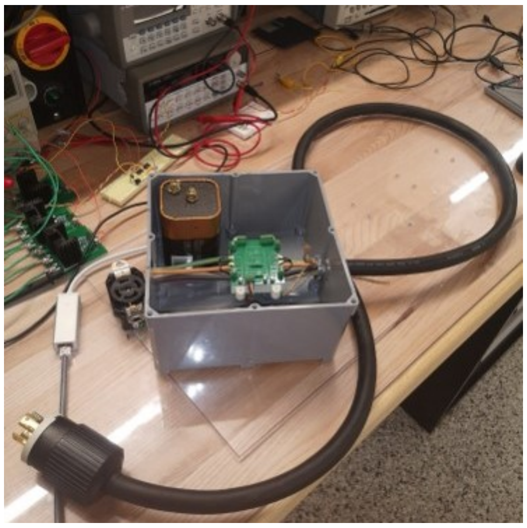

(b) Post repairs.

Figure 6-4: Sample of runs from before and after maintenance was performed on the etcher. Following repairs, current draw appears more stable during the etching phase.

The current recording platform uses a commercial voltage sensor connected directly to the three phase power lines. To further test the efficacy of fault detection systems using noninvasive sensors, a second sensor box shown in Figure 6-5 was built using a prototype of a contactless combined current and voltage sensor developed by A. Casallas [4]. Exploration of power monitoring of industrial equipment using these and other prototypes, in conjunction with our deep learning architecture, is a promising avenue for future work.


(a) Sensor box for use in clean room.

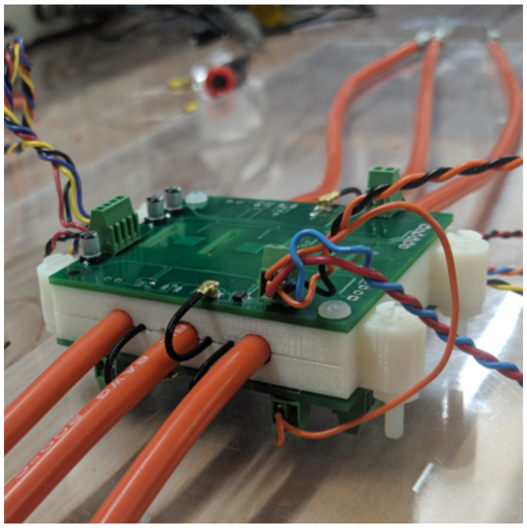

(b) Sensor prototype.

Figure 6-5: Contactless sensor prototype. The sensor clips onto the outside of powerlines and combines multiple measurements of the surrounding electric and magnetic fields provide accurate current and voltage readings robust against external sources of noise.

6.2 Continuous Learning

This section details our proposed solution to the problem of system drift. Over long periods of time, the behavior of a target piece of equipment will naturally change. This can lead to a steady decrease in the performance of a fault detection system. To combat this, we propose a continuous version of our algorithm, the weights of which are constantly updated by training on new batches of data as they are recorded. This allows our model to drift along with the system. At each time step t the network weights are updated by training for one epoch, and the model weights at each time step ϕ_t are stored. Evaluation on a new batch of data at timestep j can then be performed with model weights from any previous time step i. These weights are used to compute the residual vector $Res_{(i,j)}$, latent state representation $LS_{(i,j)}$, and Gaussian mixture model parameters $\Theta_{(i,j)}$ for each batch as shown in Algorithm 2, where i represents the last batch the network weights were trained on and j represents the batch being evaluated. Comparison of the model predictions using weights from different time steps can be used as a proxy for how much the system has drifted in that time.

Algorithm 2 Continuous FDS Training and Prediction

```
1:  procedure CAFD($Sys$)    ▷ Input: A continuous source of sensor data from a monitored system
2:      $FDS_0 \leftarrow Init\_Network\_Weights()$
3:      $t = 0$
4:      while $True$ do
5:          $t = t + 1$
6:          $X_t \leftarrow Sys.pull\_batch\_data()$
7:          $FDS_t \leftarrow FDS_{t-1}.fit(X_t)$
8:          if $t > warm\_start$ then
9:              $Res_{(t-1,t)}, LS_{(t-1,t)}, \Theta_{(t-1,t)} \leftarrow FDS_{t-1}.predict(X_t)$
10:             $Res_{(t-2,t)}, LS_{(t-2,t)}, \Theta_{(t-2,t)} \leftarrow FDS_{t-2}.predict(X_t)$
```

We hypothesize that these time lagged predictions can be used to estimate the rate of system drift as the difference in the model accuracies over time. Second order effects, such as the rate of system drift, could then potentially be used to improve fault detection accuracy and possible estimate the remaining time to failure. As a proof of concept, we run a version of this algorithm on the ADI etcher database detailed in Section 4.4. Our model is trained and evaluated chronologically on batches of 100 runs. Figure 6-6 shows an overlay of the temperature readings from one sensor for 100 etching runs. Temperatures are noticeably higher during failed etches.

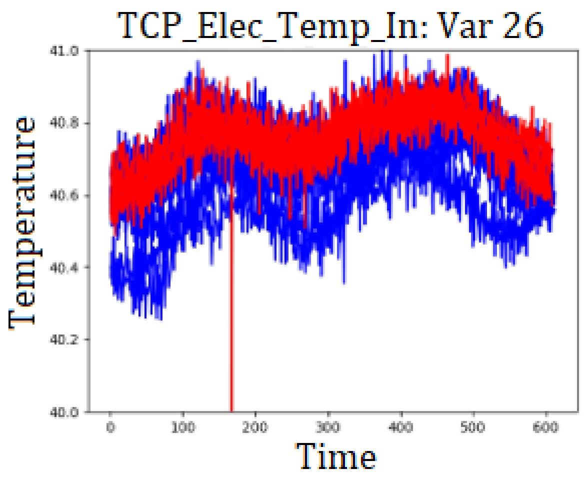


Figure 6-6: Overlay of temperature reading over the course of 100 etching runs. Failed etches are shown in red and successful etches are shown in blue

Figure 6-7 shows the magnitude of the residual vector, or compression loss, on the same 100 runs using model weights from three different points in time. The first training batch at time $t-1$ contains no failed etches, batch t contains a mix of faulty and successful runs, and batch $t+1$ contains only failed etches. Comparing the results from these three sets of model weights, we can see that as more faults are introduced into the training data, the predicted temperature rises.

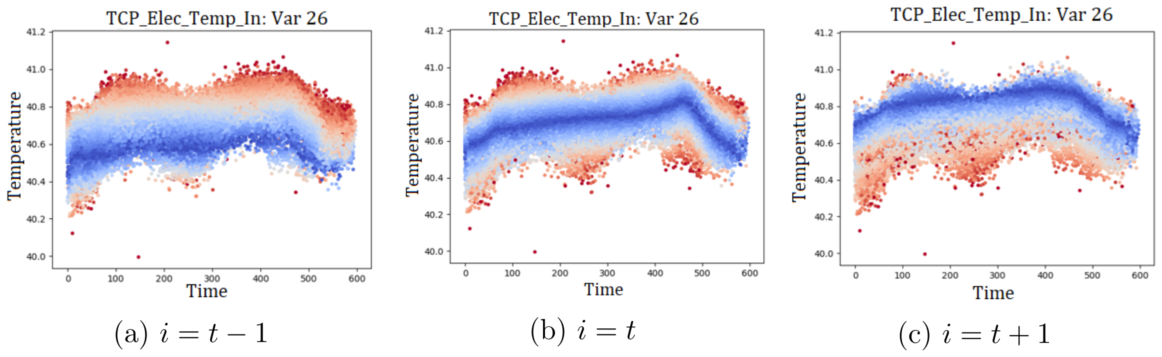


Figure 6-7: Residual error for the 100 runs in Figure 6-6 containing a mix of faulty and successful etches. A heat map is used to indicate the magnitude of the residual vector. Blue indicates low compression loss and red indicates a high compression loss. At timestep $i = t-1$ no faults are present in the training data. As the system is trained on batches containing faults at timesteps t and $t+1$, the predicted temperatures rise steadily.

6.3 Conditional Decoding

This section details our proposed solution to the problem of discrete run types. In our test on the ADI etcher database, we see vastly different anomaly detection accuracies on recipe 920 (90%) compared to recipe 945 (60%). This was true even after normalizing the anomaly scores for each run type independently. To better incorporate discrete variables Y, defining independent run types, we propose using a conditional decoder to reconstruct the input window as a function of both the latent state and the discrete variables, as shown in Figure 6-8.

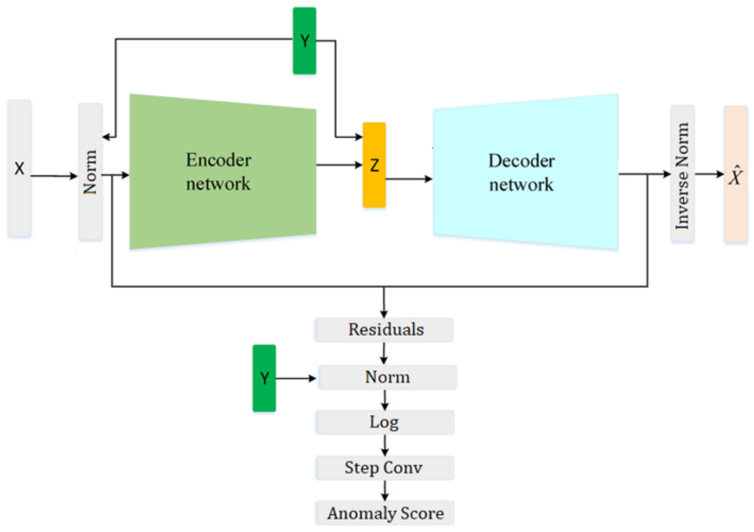


Figure 6-8: Network diagram for conditional variational autoencoder. Discrete variable are appended onto the latent state vector before decoding and used to normalize residuals before computing a final anomaly score

To this end, as well as being used to normalize inputs and residuals by run type, the discrete variables would be appended to the latent state vector before being used as the input to both the estimation network and the decoder. We hypothesis that this should allow the model to better separate variance due to faults, and variance due to machine settings and usage. Conditional variational autoencoders have already been used to great effect in anomaly detection tasks [29] and fault detection tasks [32]. In preliminary testing, we find using a conditional VAE improves the detection accuracy of the VAE by approximately 5% on the ADI etcher dataset. We expect the RDAGMM to see a similar increase in performance from this addition as well.

6.4 Change Point Detection

In the datasets used in this thesis, our raw sensor data is grouped into runs, such as a single etch, and the mean anomaly score for the duration of the run is used to represent the whole. In practice, the labels for the start and the end of different phases of operation may not be available, and in some cases may not exist. To address this, we explored integrating a change point detection algorithm into our model to automatically segment time series data into phases. We implemented and tested a version of the time series segmentation algorithm proposed by Lee et al. [19]. In this method, the raw time series input X is compressed using a sliding window autoencoder onto a reduced feature space $\vec{z}$. The rate of change of the latent state at each time point $\Delta z[t]$ is then computed, given by $\Delta\vec{z}[t] = \vec{z}[t-1] - \vec{z}[t]$. Change points are then predicted as local maxima in the $\Delta\vec{z}[t]$ vector using a peak finding algorithm. Figure 6-9 shows the results of our preliminary tests of this method against a stochastically generated synthetic data set. To generate the dataset, as described in Section 4.1, at each moment in time a true system state is determined by iteratively following a Markov chain. Signals are then generated as the sum of four sinusoids with amplitude and frequency determined by the system state. In our tests, true changes in the system state are all detectable as local maxima in the rate of change in the latent state vector. It remains an active area of inquiry how automated signal segmentation can be integrated into our system to improve fault detection rates.

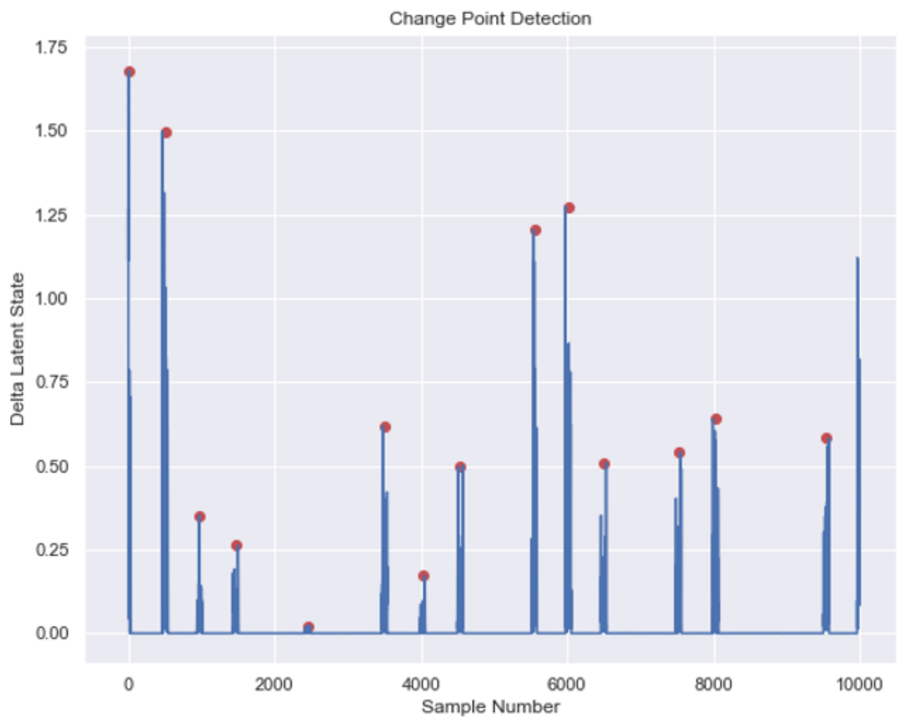


Figure 6-9: Change point detection on synthetic data. The rate of change of the latent state value is plotted on the y axis against time. Change points between system states are denoted with red dots.

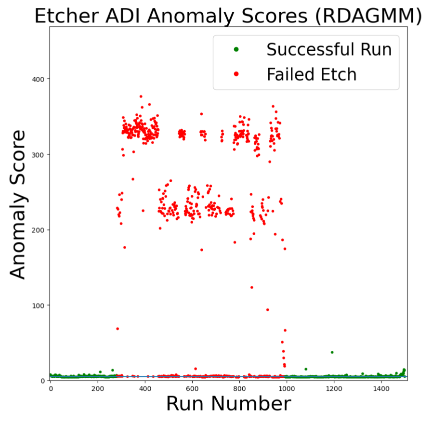
Chapter 7

Conclusion and Future Work

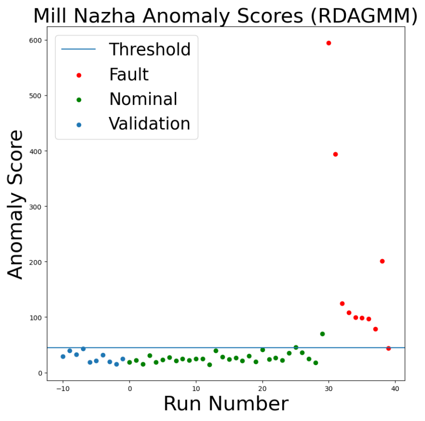
The primary goal of this project is to evaluate whether an unsupervised learning system could function as a general purpose fault detection system across a range of equipment, without expert knowledge, hyperparameter tuning, or other application specific changes. To this end, we propose a modified version of the DAGMM and evaluate its performance across a range of fault detection tasks. In this section, we summarize our results in the context of these goals and present our suggestions for future work.

7.1 Performance

The model appears to generalize well across applications from a variety of machines, with a range of sensor channels (between 2-30 in our results), a range of sampling rates (between 2 and 1200Hz) as well as a variety of sensor types. On the Bearing and Chiron Mill datasets our system scores above a 99% classification accuracy. On the remaining datasets, the majority of faults are always distinguishable by a spike in the anomaly score often spanning multiple orders of magnitude. For example, Figure 7-1 shows the chronological anomaly scores assigned to the Nazha Mill and ADI Etcher datasets. In the Nazha Mill results, we see an enormous spike in the anomaly score on the first run following the cooling system being disabled. Similarly, on the ADI Etcher database, while many faults were missed, at any threshold level our system would have thrown an alarm by the third failed etch, potentially preventing the subsequent 700 failed runs. We see the same effect in the NASA Mill dataset, our test case with a continuous measure of tool wear. Some individual runs with a worn tool are missed, particularly the borderline cases. However, the majority of faults had anomaly scores hundreds, or even thousands, of times higher than the most anomalous run with an undamaged tool, and tended to grow exponentially as the tool became more damaged. In all cases, we believe the fault detection ability of our system is high enough to be of practical use.



(a) Etcher ADI



(b) Mill Nazha

Figure 7-1: Chronological anomaly scores for the ADI Etcher and Nazha Mill evaluation datasets.

Notably, our system has significantly lower scores (by accuracy and ROC AUC) on the ADI Etcher and NASA Mill datasets. We believe this is due to the limited ability of our system to separate changes due to variance in how the machine is used, such as the etch recipe being run or material being cut with a milling machine, from changes due to faulty system behavior. This problem is most clearly seen in the NASA Mill dataset which contains eight possible cutting configurations and a continuous measure of tool wear. Without normalizing the anomaly scores by run type, the accuracy falls substantially from 82% to 67%. Even after normalization, the inability of the model to classify runs with different system settings leads us to believe these factors are still affecting the anomaly score, reducing our system's ability to tease out the effects of tool wear. We see a similar effect with the ADI etcher results, which span two etching recipes. In that instance, the detection accuracies for the 920 recipe are approximately 90% compared to 60% for the 945 recipe. This suggests the model as implemented would perform substantially better in highly repetitive settings, such as manufacturing lines, compared to a setting like a machine shop with more variance in how the equipment is being used. As discussed in Section 6.3 we believe better integrating discrete variables into the model using a conditional decoder will help solve this problem. It remains an exercise of future work how best to address it in the case of hidden variables, where the model is working with imperfect information.

7.2 Data Efficiency

Our results suggest that our model can be effective even in settings with very little training data. On the bearing dataset our model is trained on only 30 seconds of vibration data, and then successfully detects and classifies all faults in the test set. Similarly, on the Chiron mill dataset, our model is trained on only six cutting runs, and then separates faults perfectly in all but one of the k-fold evaluation runs. However, the one error on the Chiron mill evaluation is emblematic of another weakness of our system. The misclassification occurs on an evaluation loop when the model has no high speed runs in its training data. The anomaly score that our model computes is not necessarily a measure of the probability of a fault, only that something has changed compared to the data available in the training set. Any time a variable such as the run speed takes a value that is not present in the training data, our model has a high probability of flagging the instance as a possible fault. When the training data available to our model does not cover the full range of machine settings, this can lead to a high level of false positives. We believe the amount of training data needed to effectively train our system will vary substantially depending on the complexity of the machine in question, the number of sensor channels, and the variance in use.

7.3 Real World Application

While our results are encouraging, further work is needed to demonstrate the efficacy of our model in practical applications. Primarily this is due to the fact that in real world scenarios degradation to machine health is often a gradual process as in the NASA Mill dataset, as opposed to the sudden change seen in the other mill datasets (e.g. by switching out the new tool for one with more significant wear). In these scenarios, detection of system drift over longer timescales can be critical to detecting faults. The induced faults used in the majority of our tests may prove to be easier to detect than those that occur naturally. In addition, the presence of faults being mistakenly added to the training dataset can cause our model to misclassify future faults with a similar signature. It remains an important open question how the detection accuracy of our model will drop as a function of the percentage of faults present in the training dataset. The continual learning version of our detection algorithm may address this challenge, but more real world data spanning longer duration is necessary to further develop and test the efficacy of such an approach.

7.4 Thresholding

In this work, thresholds on the anomaly score are set at two standard deviations above the mean on a validation dataset. In practice, optimal thresholding should incorporate a cost-benefit analysis of the risk of missed failures versus the number of false positives. Because failures are typically very rare occurrences, even a small false positive rate can lead to the vast majority of warnings being false alarms, effectively negating the usefulness of the fault detection system. On the other hand, in some scenarios, even a single missed fault can be extremely costly, especially in scenarios where it can cause further damage to equipment downstream in the production line. Equally important is the time span within which fault needs to be detected. Using the milling machine as an example, the important event to be avoided may be a tool snapping during operation. Using a FDS to detect tool wear with some margin for error, if runs can be batched into periods longer than a single cutting run, and the probability of a missed detection could be reduced significantly (exponentially w.r.t. batch size if we could assume missed detections were independently distributed). For some applications, it is promising that in our tests our model can detect the majority of faults even at very high threshold levels. Table 7.1 shows the percentage of faults our system detects with a threshold set such that there isn't a single false positive. For applications that require low false negative rates on short time scales, it remains to be seen how much the performance of our system can be improved with further optimization.

Table 7.1: Detection Accuracies at Zero False Positives

Dataset	Detection Accuracy
Bearing	1.0
Mill (Chiron)	0.99
Mill (Nasa)	0.8
Mill (Nazha)	0.86
Etcher (ADI)	0.65

7.5 Interpretability

For the purpose of quickly diagnosing and repairing faults or discounting false positives, the interpretability of a fault detection system is extremely beneficial. To this end, one of the strengths of our system is the richness of the internal representation. Namely, the anomaly score can be broken down by both channel and time to isolate the specific sensors and periods that had the greatest contribution to a positive fault prediction. Secondly, as demonstrated with the ADI Etcher dataset the clustering algorithm can be used to calculate a specific direction between normal and faulty runs, in terms of either changes to the residual or the latent space vector. We hypothesize that by using a small number of known faults, and monitoring drift in the

RDAGMM cluster centers, it may be possible to estimate remaining useful life and schedule maintenance in advance of faults. It remains an exercise of future work to test the efficacy of such a system and explore the best way diagnostic tools could be fashioned from the internal representation of our model. As an example, disentangled variational autoencoders have been shown in practice to find internal representations where each latent value can be mapped to a meaningful generative factor [6]. In our system, this may significantly improve interpretability if it allows values in the latent space to be mapped to physically meaningful quantities in the device, such as tool speed or predicted wear in a milling machine.

7.6 Architecture Improvements

There are two changes to the neural network itself we believe could further improve performance. The first would be to use wavenets to encode the raw sensor data instead of the convolutional layers tested in this work. Wavenets have recently been used to great success in generative model for high sample rate, raw audio files [34]. The window size for a wavenet increases exponentially with the number of parameters. In our model, this could potentially allow us to dramatically increase the window size allowing for better detection of long term trends. The second change we recommend is the incorporation of an attention network. Attention networks have in recent years have achieved state of the art results on numerous time series application including fault detection [18, 20] and should further improve the model's ability to detect temporal trends.

7.7 Conclusion

The motivating question behind this project was to explore how well a fault detection system could perform given the least possible information. Our results suggest unsupervised learning techniques may allow for effective fault detection systems without any knowledge of the target system, the types of sensors being used to monitor it, or labeled examples of faults. Even if the performance of fully unsupervised systems fail to match that of systems created with expert knowledge, which may not always be the case, we believe the benefits of an out-of-the-box solution to any given fault detection task would be substantial. We hope this work serves as a step towards this goal.

Bibliography

[1] A. Agogino and K. Goebel. BEST lab, UC Berkeley. Milling Data Set. NASA Ames Prognostics Data Repository, NASA Ames Research Center, 2007.

[2] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, Jan. 2016.

[3] Vanson Borne, ServiceMax, and GE Digital. After the fall: The costs, causes and consequences of unplanned downtime. 2017.

[4] A. Casallas. Contactless voltage and current estimation using signal processing and machine learning. Master's thesis, Massachusetts Institute of Technology, Aug. 2019.

[5] Joseph E. Cavanaugh and Andrew A. Neath. Generalizing the derivation of the Schwarz information criterion. *Communications in Statistics-Theory and Methods*, 28(1):49–66, 1999.

[6] Ricky TQ Chen, Xuechen Li, Roger B. Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 2610–2620, 2018.

[7] T. Chen. Anomaly detection in semiconductor manufacturing through time series forecasting using neural networks. Master's thesis, Massachusetts Institute of Technology, Aug. 2018.

[8] Dave Crumrine and Doug Post. When true cost of downtime is unknown, bad decisions ensue. International Society of Automation. Jan. 2016.

[9] Mariela De Lucas Alvarez and David M. Lane. A Hidden Markov Model application with Gaussian Mixture emissions for fault detection and diagnosis on a simulated AUV platform. In *OCEANS 2016 MTS/IEEE Monterey, OCE 2016*. Institute of Electrical and Electronics Engineers, Nov. 2016.

[10] Kai Ding, Sheng Ding, Andrey Morozov, Tagir Fabarisov, and Klaus Janschek. On-line error detection and mitigation for time-series data of cyber-physical systems using deep learning based methods. In *Proceedings - 2019 15th European Dependable Computing Conference, EDCC 2019*, pages 7–14. Institute of Electrical and Electronics Engineers, Sep. 2019.

[11] Teguh Handjojo Dwiputranto, Noor Akhmad Setiawan, and Teguh Bharata Aji. Machinery equipment early fault detection using Artificial Neural Network based Autoencoder. In *Proceeding - 2017 3rd International Conference on Science and Technology-Computer, ICST 2017*, pages 66–69. Institute of Electrical and Electronics Engineers, Aug. 2017.

[12] Jicong Fan, Wei Wang, and Haijun Zhang. Autoencoder based high-dimensional data fault detection system. In *Proceedings - 2017 IEEE 15th International Conference on Industrial Informatics, INDIN 2017*, pages 1001–1006. Institute of Electrical and Electronics Engineers, Nov. 2017.

[13] Long Gao, Donghui Li, Ding Li, and Haiyan Yin. An improved LSTM based sensor fault diagnosis strategy for the air-cooled chiller system. In *Chinese Control Conference, CCC*, volume 2019-July, pages 4990–4995. IEEE Computer Society, Jul. 2019.

[14] D. M. Hawkins. *Identification of Outliers.* Springer Netherlands, 1980.

[15] H. He. Applications of reference cycle building and k-shape clustering for anomaly detection in the semiconductor manufacturing process. Master's thesis, Massachusetts Institute of Technology, Aug. 2018.

[16] Yang Huang, Chiun Hsun Chen, and Chi Jui Huang. Motor fault detection and feature extraction using RNN-based variational autoencoder. *IEEE Access*, 7:139086–139096, 2019.

[17] Yongjun Jin, Chenlu Qiu, Lei Sun, Xuan Peng, and Jianning Zhou. Anomaly detection in time series via robust PCA. In *2017 2nd IEEE International Conference on Intelligent Transportation Engineering, ICITE 2017*, pages 352–355. Institute of Electrical and Electronics Engineers, Oct. 2017.

[18] E. Kim, S. Cho, B. Lee, and M. Cho. Fault detection and diagnosis using self-attentive convolutional neural networks for variable-length sensor data in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 32(3):302–309, 2019.

[19] Wei-Han Lee, Jorge Ortiz, Bongjun Ko, and Ruby B. Lee. Time series segmentation through automatic feature learning. *CoRR*, abs/1801.05394, 2018.

[20] Ding Li, Donghui Li, Chengdong Li, Lin Li, and Long Gao. A novel data-temporal attention network based strategy for fault diagnosis of chiller sensors. *Energy and Buildings*, 198:377 – 394, 2019.

[21] K. A. Loparo. Case Western Reserve University Bearing Data Center. *Bearings Vibration Data Sets, Case Western Reserve University: http://csegroups.case.edu/bearingdatacenter/home*, pages 22–28, 2012.

[22] Yiwei Lu, K. Mahesh Kumar, Seyed Shahabeddin Nabavi, and Yang Wang. Future frame prediction using convolutional VRNN for anomaly detection. In *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2019*. Institute of Electrical and Electronics Engineers, Sep. 2019.

[23] O. Makhlouk. Time series data analytics: Clustering-based anomaly detection techniques for quality control in semiconductor manufacturing. Master's thesis, Massachusetts Institute of Technology, Aug. 2018.

[24] Aaron F. McDaid, Derek Greene, and Neil Hurley. Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv preprint arXiv:1110.2515*, 2011.

[25] Baligh Mnassri, El Mostafa El Adel, Bouchra Ananou, and Mustapha Ouladsine. Fault detection and diagnosis based on PCA and a new contribution plot. *IFAC Proceedings Volumes*, 42(8):834–839, Jan. 2009.

[26] Mahnoosh Nadjarpoorsiyahkaly and Chee Peng Lim. A hybrid neural classifier for dimensionality reduction and data visualization and its application to fault detection and classification of induction motors. In *Proceedings - 2011 6th International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2011*, pages 146–150, 2011.

[27] Dawei Pan, Zhe Song, Longqiang Nie, and Benkuan Wang. Satellite telemetry data anomaly detection using Bi-LSTM prediction based model. In *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1–6. Institute of Electrical and Electronics Engineers, May 2020.

[28] Pangun Park, Piergiuseppe Di Marco, Hyejeon Shin, and Junseong Bang. Fault detection and diagnosis using combined autoencoder and long short-term memory network. *Sensors (Switzerland)*, 19(21), Nov. 2019.

[29] A. A. Pol, V. Berger, C. Germain, G. Cerminara, and M. Pierini. Anomaly detection with conditional variational autoencoders. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1651–1657, 2019.

[30] Oleksandr I. Provotar, Yaroslav M. Linder, and Maksym M. Veres. Unsupervised anomaly detection in time series using LSTM-based autoencoders. In *2019 IEEE International Conference on Advanced Trends in Information Theory, ATIT 2019 - Proceedings*, pages 513–517. Institute of Electrical and Electronics Engineers, Dec. 2019.

[31] Lang Qin and Qianqian Zhang. New algorithm for multiple satellite faults detection and exclusion based on time series prediction. In *2017 Forum on Cooperative Positioning and Service, CPGPS 2017*, pages 345–350. Institute of Electrical and Electronics Engineers, Oct. 2017.

[32] You ren Wang, Guo dong Sun, and Qi Jin. Imbalanced sample fault diagnosis of rotating machinery using conditional variational auto-encoder generative adversarial network. *Applied Soft Computing*, 92:106333, 2020.

[33] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4393–4402, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018.

[34] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[35] Arnaud Van Looveren, Giovanni Vacanti, Janis Klaise, and Alexandru Coca. Alibi-Detect: Algorithms for outlier and adversarial instance detection, concept drift and metrics. *GitHub repository*, 2019.

[36] Tao Xinmin, Du Baoxiang, and Xu Yong. Bearings fault diagnosis based on GMM model using Lyapunov exponent spectrum. In *IECON Proceedings (Industrial Electronics Conference)*, pages 2666–2671, 2007.

[37] Jae Wan Yang, Young Doo Lee, and In Soo Koo. Convolutional autoencoder-based sensor fault classification. In *International Conference on Ubiquitous and Future Networks, ICUFN*, volume 2018-July, pages 865–867. IEEE Computer Society, Aug. 2018.

[38] Jianbo Yu. Fault detection using principal components-based Gaussian mixture model for semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing*, 24(3):432–444, Aug. 2011.

[39] Yu Zhang, Chris Bingham, Michael Gallimore, and Darren Cox. Novelty detection based on extensions of GMMs for industrial gas turbines. In *2015 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications, CIVEMSA 2015*. Institute of Electrical and Electronics Engineers, Jul. 2015.

[40] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018.