11/27/2024

Predictive Maintenance on Direct Air Capture (DAC) Process Equipment via LSTM Autoencoder Model

MGT 6748

Dan Hardiman
ZERO CARBON SYSTEMS

Introduction

The Intergovernmental Panel on Climate Change suggests that between 1.3-2.9 gigatons of carbon dioxide will need to be removed from the atmosphere annually by 2050 to limit global warming to 1.5 °C above pre-industrial temperatures [1]. There are several solutions in development for addressing the problem of removing CO_2 and permanently storing it. Among these is direct air capture (DAC), which is a method for chemically or physically separating carbon dioxide from other gas components in air. This is an emerging technology that is being pursued by a swath of new companies, including Zero Carbon Systems (ZCS). ZCS's technology uses a mechanical design solution to move a substantial amount of air across a substrate that physically adsorbs CO_2 and then applies a process to release and store it [2]. The process of developing this technology involves running many experiments to collect data on different contactor materials and under a range of different operating conditions. The data from these experiments is then used to build models for characterizing the operation of a large-scale facility and optimizing the design of such a facility to achieve high CO2 production capacity with low energy and cost expenditure. One piece of equipment that ZCS uses to run these experiments is called the core tester. This machine flows CO2 at a set concentration through a single core (the larger scale plant uses panels which are composed of many cores side by side) which is filled with a set amount of the adsorbent polymer material polyethyleneimine (PEI). The CO2 concentrations at the inlet and outlet of the core are measured using LICOR sensors, and the difference between these concentrations over time is used to calculate the amount of CO2 adsorbed to/desorbed from the core during a given test (Note: the process flow diagram for the core tester cannot be shown here due to confidentiality restrictions). The experimental data collected from the core tester is crucial for the overall model development work, so any unplanned downtime that the unit undergoes can be quite costly to ZCS's timelines. As such, it is very important to be able to predict failures before they happen and/or detect them in real time to minimize the downtime of the unit and maintain the quality of the data that the unit produces. Furthermore, since the amount of historical data available from the core tester is limited, it is important that the analytical modeling techniques to be used for this task do not rely on having large, labeled datasets with numerous past known failures to be used for training. Therefore, the solutions to be considered for this problem will utilize unsupervised machine learning techniques.

Literature Review

Predictive maintenance (PdM) is an application of machine learning and AI techniques that has substantially grown in popularity in recent years. It can take many different forms across different industries, and the appropriate ML models to use for this task vary according to the types of data produced by the machines used in each industry. Paolanti et al. (2018) performed a comprehensive survey of PdM methods in use across different industries and their machine learning approaches [3]. The basic scheme of PdM is the following: 1) take measurements of physical quantities in real time; 2) estimate measurable (or non-measurable) parameters at time t + dt; 3) identify the system status considered anomalous or faulty; 4) plan preventative and corrective activities before the system reaches the critical condition. This scheme requires the ability to forecast the values of the relevant physical quantities for the system at hand, compare them to the real time measurements, and identify when the forecasts don't match the current state of the system such that the system has reached or is approaching an anomalous state requiring intervention. Conventional approaches for this task of forecasting include cross-sectional forecasting (the estimation of parameters of which there exist no measurements) and time series forecasting (the estimation of parameters that change over time). Furthermore, for a given problem to be suitable for a PdM solution, there typically need to be three data sources available: 1) fault history – a dataset containing a sufficient number of examples of the normal operation scheme and the failure scheme(s) for training the algorithm; 2) maintenance/repair history – information about replaced components, preventative maintenance tasks performed previously, etc; 3) machine conditions – data containing time-varying functions that acquire patterns associated with aging or any anomaly that could cause performance reduction. The types of machine learning algorithms used for PdM fall into two general categories: classification and regression. Classification models can either be binary (i.e. estimating the probability that the equipment will fail over a given period in the future) or multiclass (e.g. categorizing failures based on different known root causes or allocating time intervals for the failures of individual components of a machine).

Amruthnath et al. (2018) analyzed vibration data collected from a submarine using several different unsupervised machine learning models and compared their effectiveness at predicting/detecting a known failure point in the dataset [4]. They used PCA to reduce the dataset to two dimensions and tested several different types of clustering models (K-means, C-means, hierarchical clustering, GMM-EM). All of these models agreed that the optimal number of clusters for the dataset was three, with these three clusters corresponding to "healthy", "warning", and "faulty" equipment states. However, even though these models are unsupervised (i.e. don't require a labeled dataset for training), interpreting the results and identifying which data points correspond to the faulty equipment state is a non-trivial task, and there is no guarantee that the same clusters found in the training data would be seen in future data as well. Furthermore, the features in the raw dataset are all various metrics pertaining to vibration measurements from a single exhaust fan. In the core tester and other types of equipment that ZCS operates, there are many different parts of the system that could fail independently of one another, so applying a dimensionality reduction and clustering algorithm to all these data streams together as the authors do here wouldn't make the most sense for the ZCS use case.

The general task at hand with the PdM problem is the unsupervised identification of anomalies in time series data. Aminikhanghahi et al. (2017) performed a survey of techniques that have been used for change point detection in time series data, and their analysis classified each technique as either supervised or unsupervised [5]. Of the unsupervised methods listed, the most well-known is cumulative sum (CUSUM), which accumulates deviations relative to a specified target of incoming measurements and identifies a change point when the cumulative sum exceeds a set threshold. This approach would be well-suited for cases where an equipment failure would manifest as a drift over time from a sensor's expected value. However, it wouldn't be as efffective if the sensor's values were not evenly distributed on either side of the expected value when the equipment was in a healthy state, or if an equipment failure could show up as a single anomalous value in the time series that wouldn't make a large difference to the trend of cumulative deviations. An ideal algorithm would be able to detect any and all of the possible types of change points that could happen in a given time series.

Larzalere (2019) created and trained an AI deep learning neural network for anomaly detection on vibration data from four bearings that were run to failure under constant load [6]. The implementation was done using Python, Keras, and Tensorflow, and the specific type of model used was a long short-term memory (LSTM) autoencoder model. LSTM networks are a subtype of recurrent neural networks (RNN), which are defined by their ability to persist information (i.e. cell state) for use later in the network thereby solving the long-term dependency problem that plagues conventional RNNs [8]. As such, they are especially well-suited for analyzing time-dependent data that changes over time, such as sequential measurement data from sensors for detecting anomalies. The LSTM model takes the training data, creates a compressed representation of its core features, and then learns to reconstruct the original data from the compressed version. The distribution of the reconstruction errors on the training data is then used to determine an appropriate error threshold beyond which a data point would be flagged as being an anomaly. The LSTM model in this paper was trained on a subset of the data during a time period when the bearings were functioning normally, and then tested on another subset of the data during which time the bearings failed. The reconstruction errors for the training data were plotted as a histogram and the threshold value was selected visually based on the cut-off point of the upper tail. The threshold value was set high enough to avoid too many false positives, but not so high that true anomalies would not be detected.

Methodology

Dataset

The dataset used in this study consists of data from various sensors on the core tester from March to November of 2024. The core tester database contains a variety of different columns, including measurements of pressure, temperature, CO2 concentration, and moisture concentration, valve positions, system states, and experimental setpoints/parameters. The data for each of these columns can be queried with different frequencies of data sampling from the traces up to 5 Hz (i.e. one data point every 0.2 seconds); however, that amount of data would be excessive for these purposes, so an interval of one data point every 900 seconds/15 minutes over this seven-month time period was used instead.

Many of these sensors contain information that could convey signals indicating system malfunctions. For example, earlier this year it was found that an N2 valve on the core tester was not opening and closing properly and needed to be replaced. As shown below in Figure 1, this manifested as a slight drop in pressure (brown trace) and a corresponding increase in the inlet moisture concentration (red trace) when the system state switched from adsorption to desorption (shortly after 20:00 in the time series plot). While this is certainly an observable signal that would stand out as an anomaly to someone familiar with this data, training a model to understand the typical shapes of these curves for each of the different sensors to the extent that it would be able to detect a shift similar to that shown in Figure 1 would be a challenge, especially since these curves can vary depending on the type of test being run. That being said, this would be a great area of exploration for future research and experimentation since this event is one of the few known instances of a hardware failure on the core tester up to this point that resulted in a clear signal in the data.

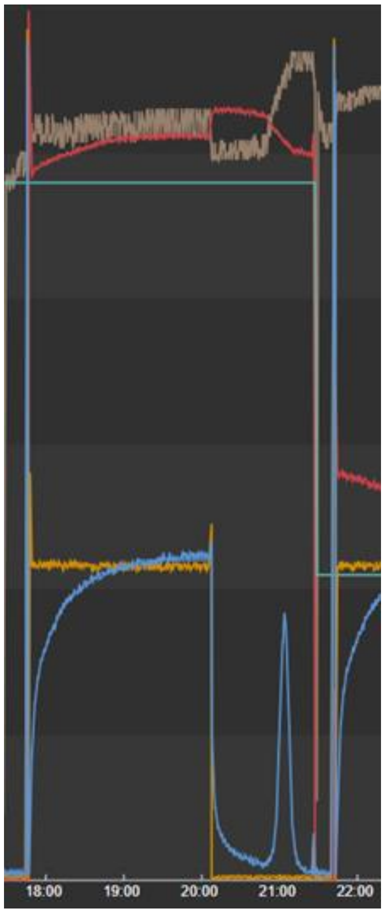


Figure 1. Time series data from example test showing pressure drop (brown trace) and moisture increase (red trace) resulting from N2 valve failure.

Each test that is run on the core tester has setpoints associated with it for the various experimental parameters. These include the dry bulb temperature, dew point temperature, N2 flow rate, CO2 flow rate, inlet CO2 concentration, and the temperatures for the two deaeration tanks. Comparing these setpoint values to the corresponding measured values for each sensor could be a reliable source of information for detecting various types of equipment failures that would impact test performance. The deltas between measured values and setpoints would be expected to always be close to 0 (at least during the times when a test is being run), so therefore any systematic drifts over time or any data points of large magnitude could reasonably be interpreted as a signal of an equipment malfunction or other type of anomaly occurring during the test. While not every failing data point on these delta charts would necessarily require corrective actions to be taken on the equipment (calibrate sensor, replace parts, etc.), it would at least indicate that the data collected from the test when the failure happened is potentially compromised and shouldn't be used. See Figures 2-7 below for charts showing the measured data and setpoint deltas for each of the sensors mentioned above. While most of the raw data charts show quite a lot of variability, the setpoint delta charts generally tend to stay near zero on average with a few instances of large magnitude variations from baseline (hypothetically indicating equipment issues that happened at those times) but no obvious systematic drifts.

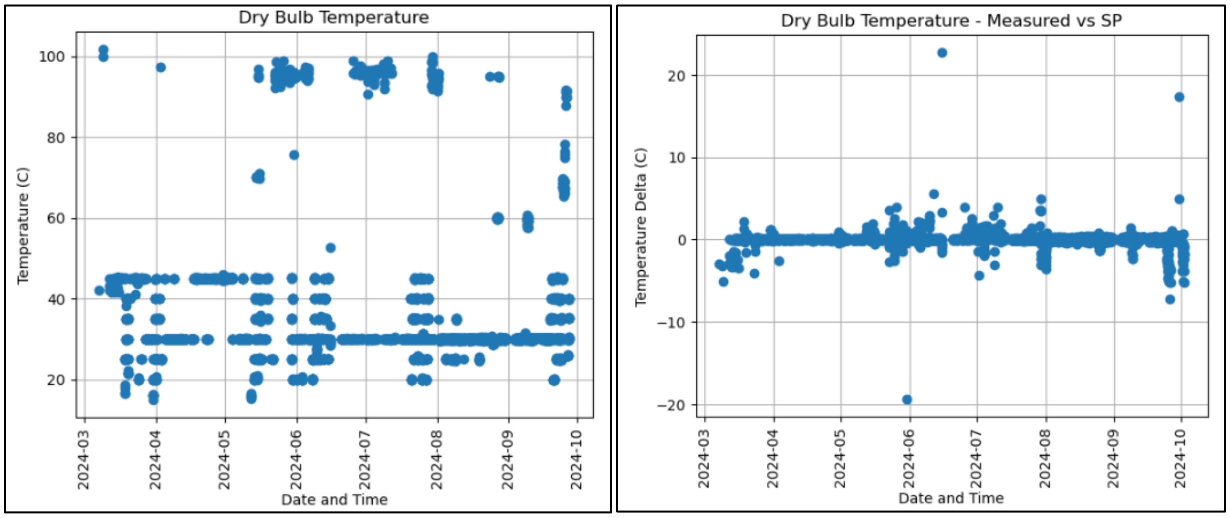


Figure 2. Dry bulb temperature – raw data and measured vs setpoint delta.

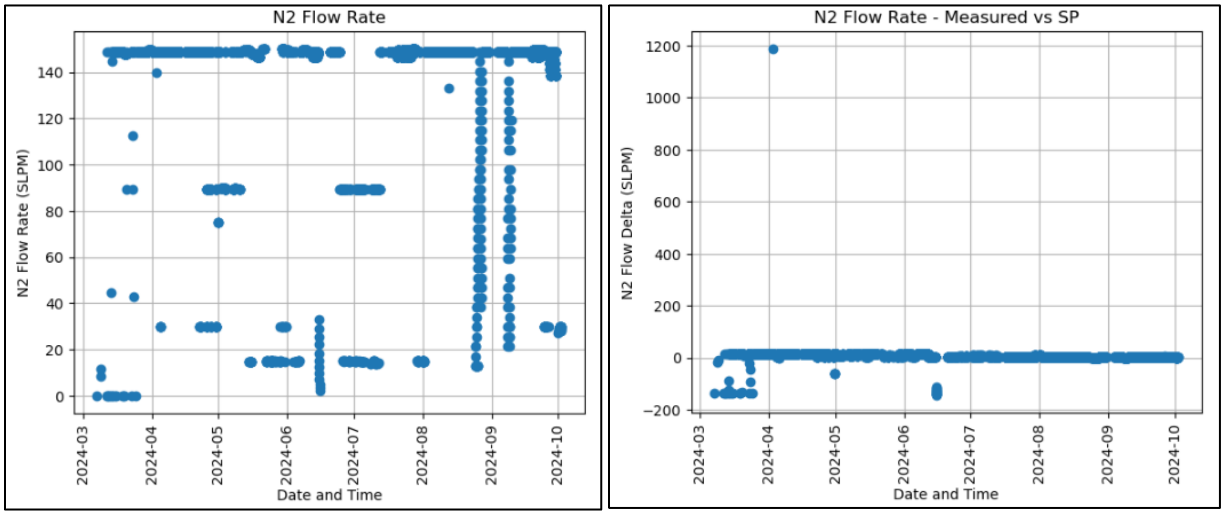


Figure 3. N2 flow rate – raw data and measured vs setpoint delta.

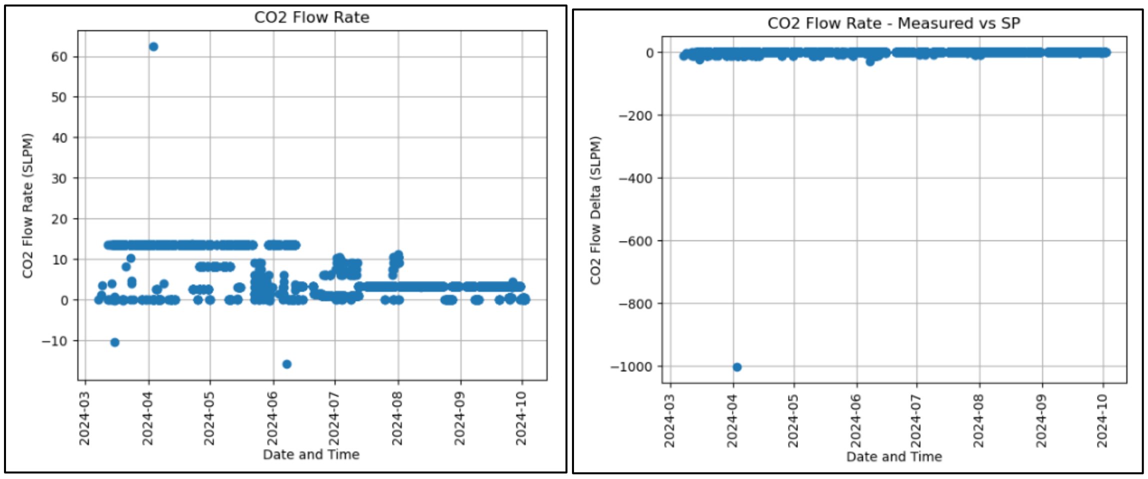


Figure 4. CO2 flow rate - raw data and measured vs setpoint delta.

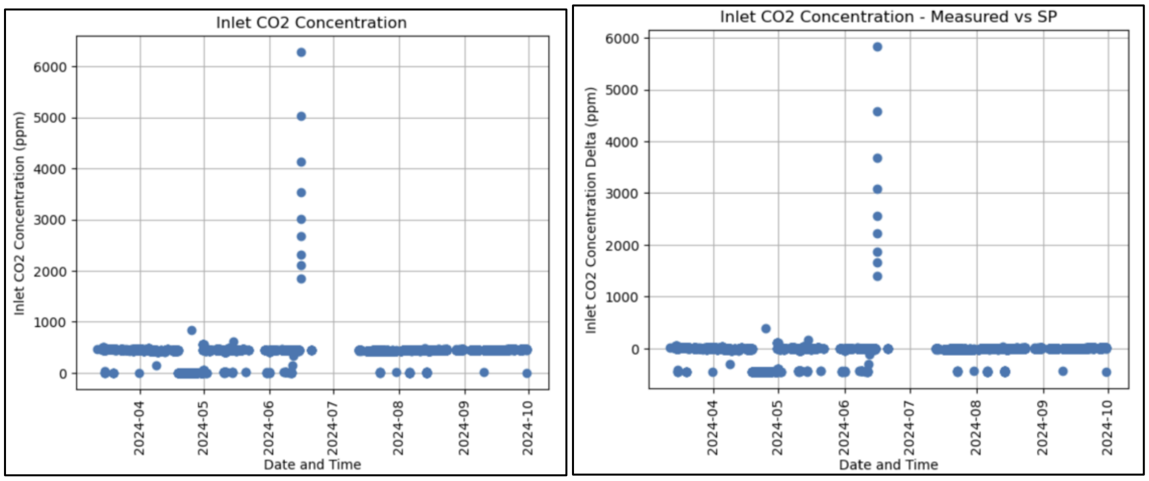


Figure 5. Inlet CO2 concentration - raw data and measured vs setpoint delta.

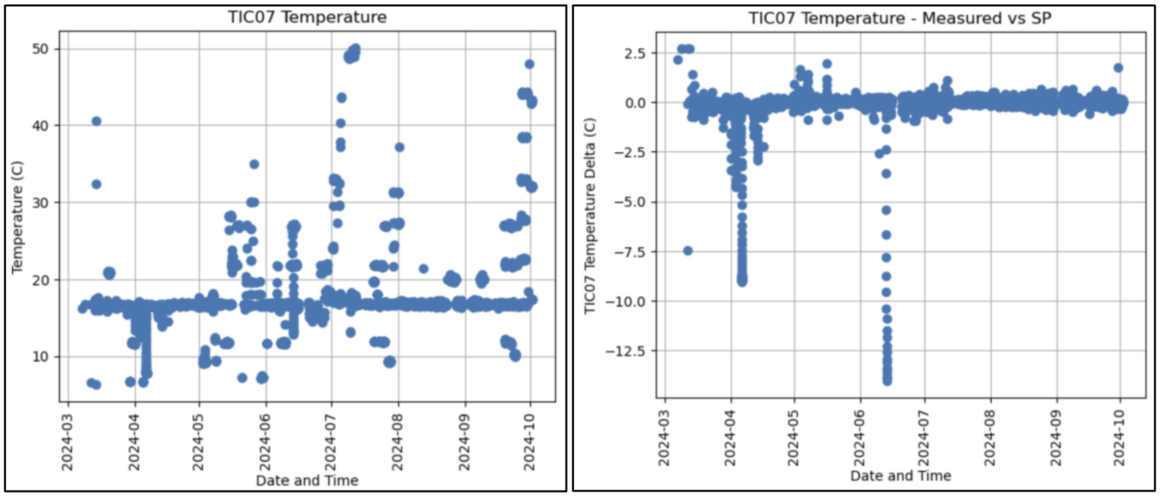


Figure 6. TIC07 (deaeration tank 1) temperature – raw data and measured vs setpoint delta.

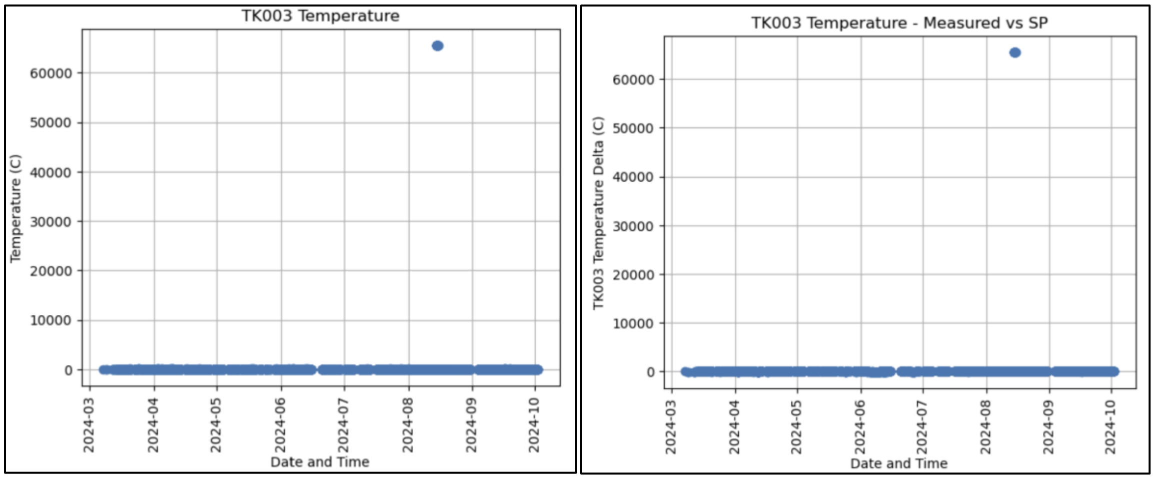
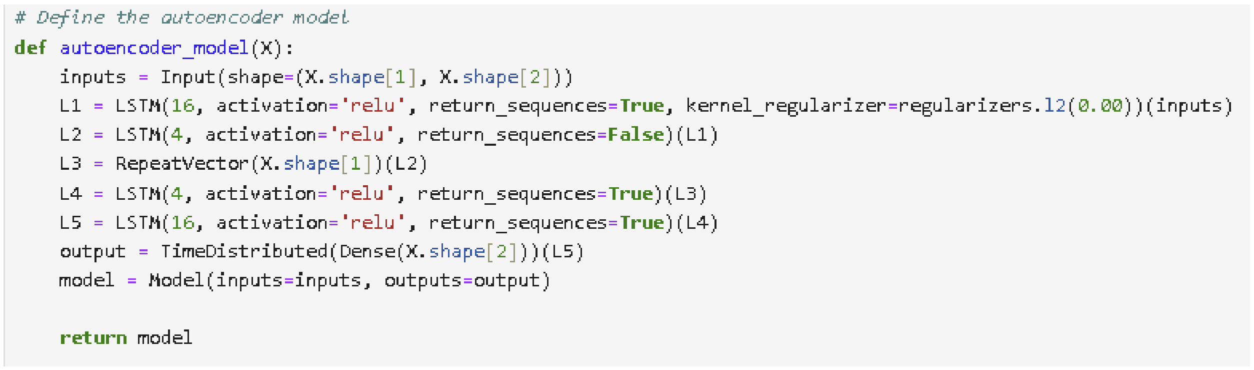


Figure 7. TK003 (deaeration tank 2) temperature – raw data and measured vs setpoint delta.

Model Design

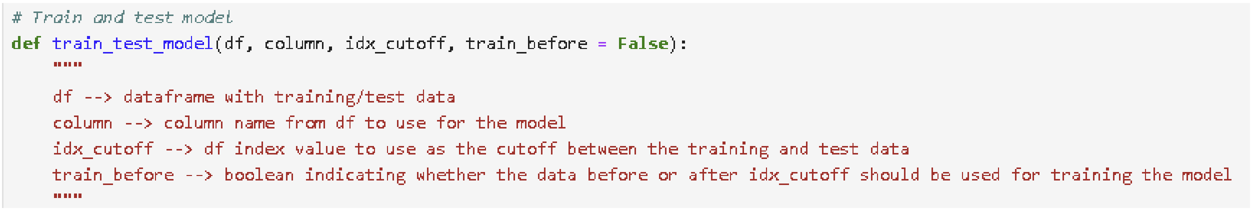
The setpoint deltas for each of the sensors above will be modeled using an LSTM autoencoder with the same scheme as that used by Larzalere [6]. See Figure 8 below for the excerpt of the python code that defines this model using functions imported from the Keras library. The first few neural network layers create a compressed representation of the data input, the vector layer in the middle distributes the compressed vector across the decoder, and the last few layers provide the reconstructed input data.



```python
# Define the autoencoder model
def autoencoder_model(X):
    inputs = Input(shape=(X.shape[1], X.shape[2]))
    L1 = LSTM(16, activation='relu', return_sequences=True, kernel_regularizer=regularizers.l2(0.00))(inputs)
    L2 = LSTM(4, activation='relu', return_sequences=False)(L1)
    L3 = RepeatVector(X.shape[1])(L2)
    L4 = LSTM(4, activation='relu', return_sequences=True)(L3)
    L5 = LSTM(16, activation='relu', return_sequences=True)(L4)
    output = TimeDistributed(Dense(X.shape[2]))(L5)
    model = Model(inputs=inputs, outputs=output)

    return model
```

Figure 8. Python code snippet defining the LSTM autoencoder model.

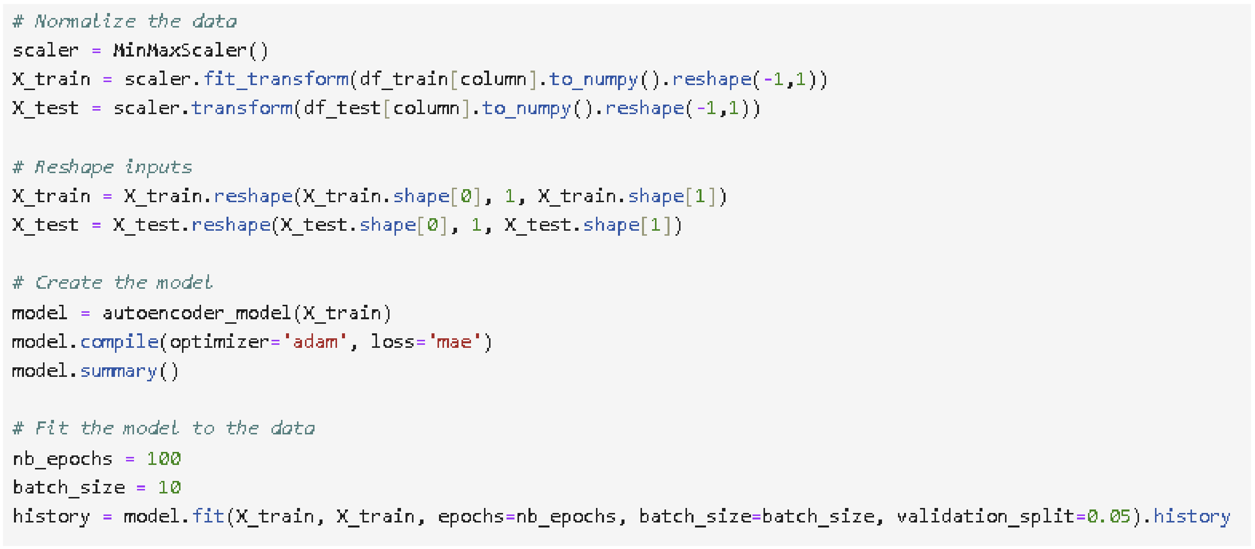
The master function used for training and testing the LSTM autoencoder model is shown below in Figure 9. This function takes as input the dataframe containing the training and test data (i.e. the raw data pulled from the core tester database along with all of the calculated setpoint delta columns), the particular setpoint delta column to use for each unique call of the function, an index value indicating where to split the dataframe into subsets to use for training and testing, and a Boolean indicating whether the subset of the data before or after the cutoff index should be used for training.



```python
# Train and test model
def train_test_model(df, column, idx_cutoff, train_before = False):
    """
    df --> dataframe with training/test data
    column --> column name from df to use for the model
    idx_cutoff --> df index value to use as the cutoff between the training and test data
    train_before --> boolean indicating whether the data before or after idx_cutoff should be used for training the model
    """
```

Figure 9. Python code defining the master function for training and testing the LSTM autoencoder model.

The function then applies a min/max scaling to both the training and test data (see Figure 10), creates the LSTM model using the function in Figure 8, fits the model to the training data, and plots the losses (i.e. reconstruction errors) both as a function of training epoch and as a histogram/density function.

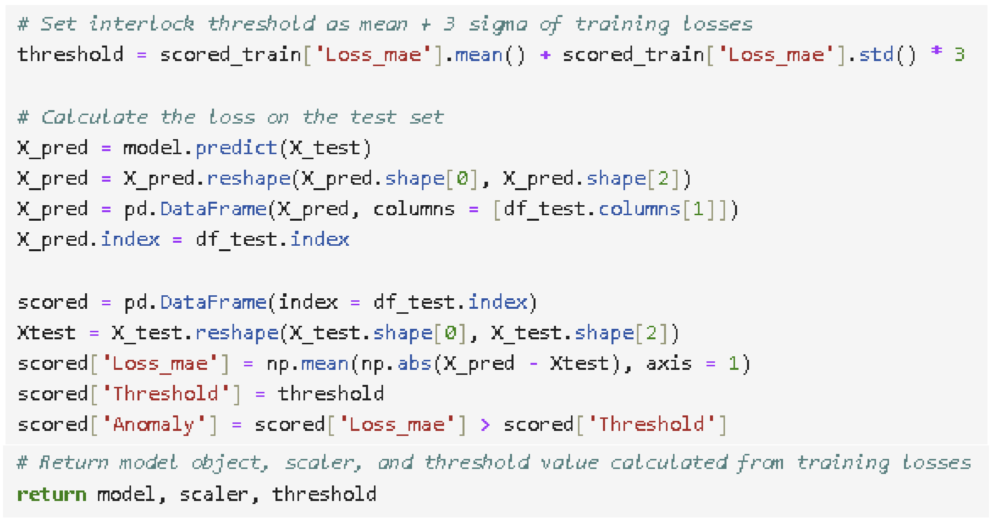


```python
# Normalize the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(df_train[column].to_numpy().reshape(-1,1))
X_test = scaler.transform(df_test[column].to_numpy().reshape(-1,1))

# Reshape inputs
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])

# Create the model
model = autoencoder_model(X_train)
model.compile(optimizer='adam', loss='mae')
model.summary()

# Fit the model to the data
nb_epochs = 100
batch_size = 10
history = model.fit(X_train, X_train, epochs=nb_epochs, batch_size=batch_size, validation_split=0.05).history
```

Figure 10. Section of the master function that applies the scaling to the data and creates and trains the LSTM model.

The anomaly threshold value is set as the mean plus 3 standard deviations of the training losses. In Larzalere's example [6], the appropriate threshold value was easily identifiable from the distribution of training losses since the upper tail of the histogram cut off midway between 0.25 and 0.3. In experimenting with the core tester data however, the distributions of training losses for the various setpoint deltas did not follow such clean-cut patterns (see Results section), so the mean plus 3 sigma seemed like a reasonable choice for the threshold value given the quasi-normal shapes of the distributions.

The losses are then calculated on the test set and any data points whose losses fall above this threshold are identified as anomalies. As shown in Figure 11, the function returns the LSTM model object, the min/max scaler object, and the interlock threshold value which are stored and can be applied to any future data using the same scheme of operations (scale data, predict output values with model, calculate losses/reconstruction errors and compare to threshold value to identify any anomalies).



```python
# Set interlock threshold as mean + 3 sigma of training losses
threshold = scored_train['Loss_mae'].mean() + scored_train['Loss_mae'].std() * 3

# Calculate the loss on the test set
X_pred = model.predict(X_test)
X_pred = X_pred.reshape(X_pred.shape[0], X_pred.shape[2])
X_pred = pd.DataFrame(X_pred, columns = [df_test.columns[1]])
X_pred.index = df_test.index

scored = pd.DataFrame(index = df_test.index)
Xtest = X_test.reshape(X_test.shape[0], X_test.shape[2])
scored['Loss_mae'] = np.mean(np.abs(X_pred - Xtest), axis = 1)
scored['Threshold'] = threshold
scored['Anomaly'] = scored['Loss_mae'] > scored['Threshold']
# Return model object, scaler, and threshold value calculated from training losses
return model, scaler, threshold
```

Figure 11. Section of the master function that calculates the threshold value, calculates the losses on the test set to identify any failing points, and returns the model object, scaler object, and threshold.

Results and Discussion

The LSTM autoencoder model outlined above was created, trained, and tested for each of the core tester setpoint deltas (dry bulb temperature, dew point temperature, N2 flow rate, CO2 flow rate, CO2 inlet concentration, and deaeration tank temperatures) using data collected between March 1st, 2024 to November 7th, 2024 at an interval of 900s between points. The time series for the dry bulb temperature is shown below in Figure 12 with the training data in blue and the test data in orange. The cutoff point between the training and test data was selected such that the training data would contain some variability, but not the largest deviations which would likely correspond to equipment issues. If these very large deviations were included in the training data, then the threshold value would likely end up being too high and these types of events wouldn't be flagged.

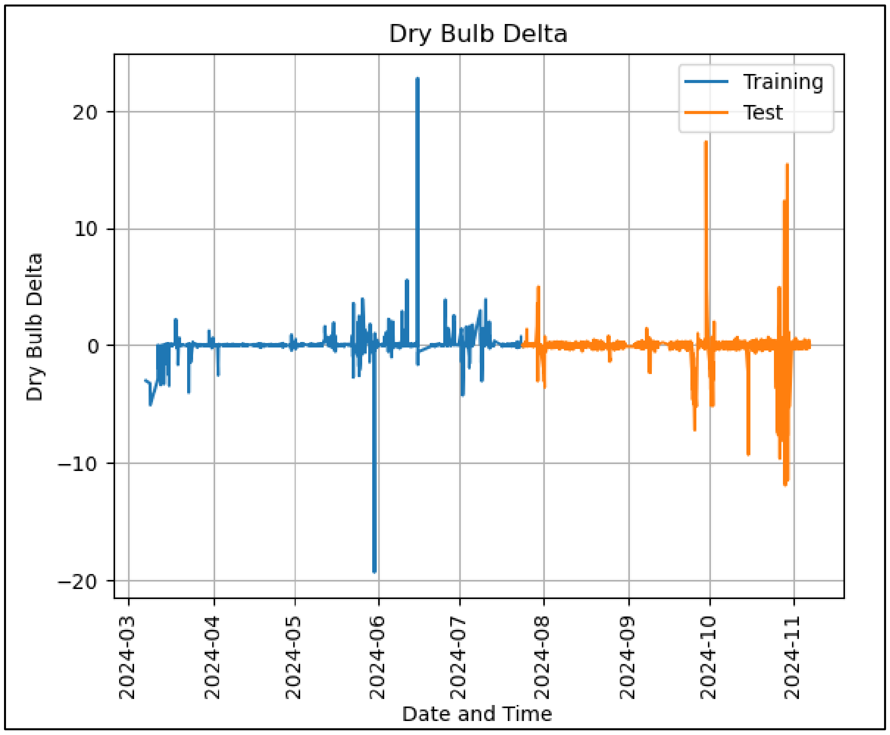
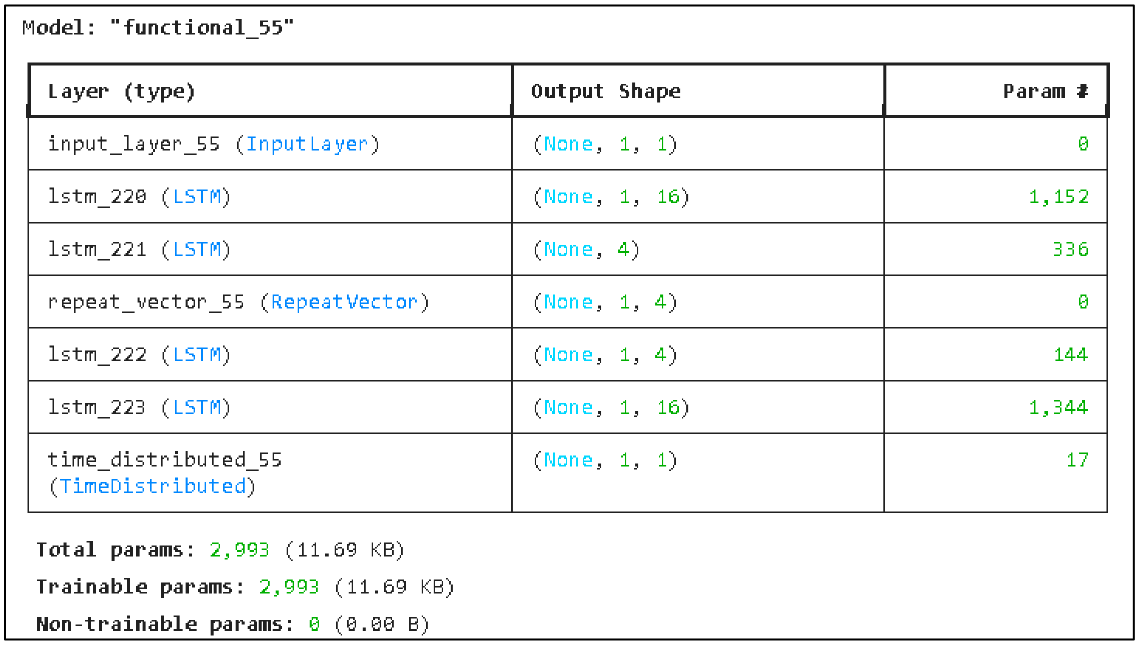


Figure 12. Dry bulb temperature setpoint delta time series with training/test data split used for LSTM model.

The summary of the LSTM model is shown below in Figure 13. This model follows the scheme in the function shown in Figure 8, and the model summary is the same for all sensors (same layers, output shapes, # of trainable parameters, etc).



```
Model: "functional_55"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃        Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━┩
│ input_layer_55 (InputLayer)          │ (None, 1, 1)                │              0 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ lstm_220 (LSTM)                      │ (None, 1, 16)               │          1,152 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ lstm_221 (LSTM)                      │ (None, 4)                   │            336 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ repeat_vector_55 (RepeatVector)      │ (None, 1, 4)                │              0 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ lstm_222 (LSTM)                      │ (None, 1, 4)                │            144 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ lstm_223 (LSTM)                      │ (None, 1, 16)               │          1,344 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ time_distributed_55                  │ (None, 1, 1)                │             17 │
│ (TimeDistributed)                    │                             │                │
└──────────────────────────────────────┴─────────────────────────────┴────────────────┘

 Total params: 2,993 (11.69 KB)
 Trainable params: 2,993 (11.69 KB)
 Non-trainable params: 0 (0.00 B)
```

Figure 13. LSTM autoencoder model summary output for dry bulb temperature setpoint delta.

The model loss as a function of training epoch and distribution of losses on the training data are shown below in Figures 14 and 15. The model loss dropped down to near 0 after the first training epoch and didn't change much thereafter, and the histogram of training losses was very tightly distributed near 0 with only a handful of high-flier points in the distribution. The models that were trained on the other sensors aside from dry bulb temperature also showed very similar trends in these plots.

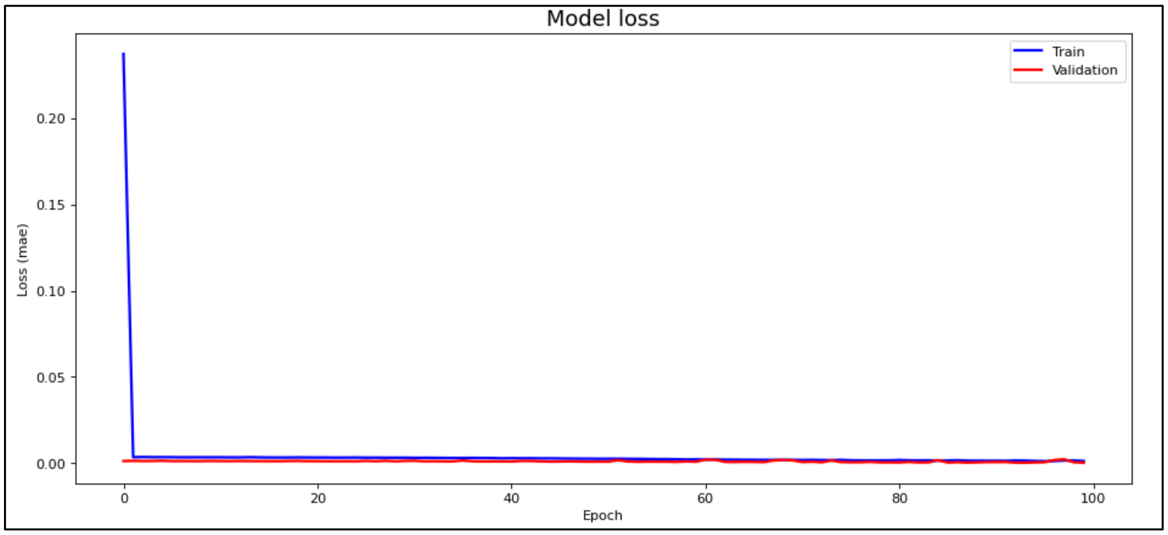


Figure 14. LSTM autoencoder model loss as a function of training epoch for dry bulb temperature setpoint delta.

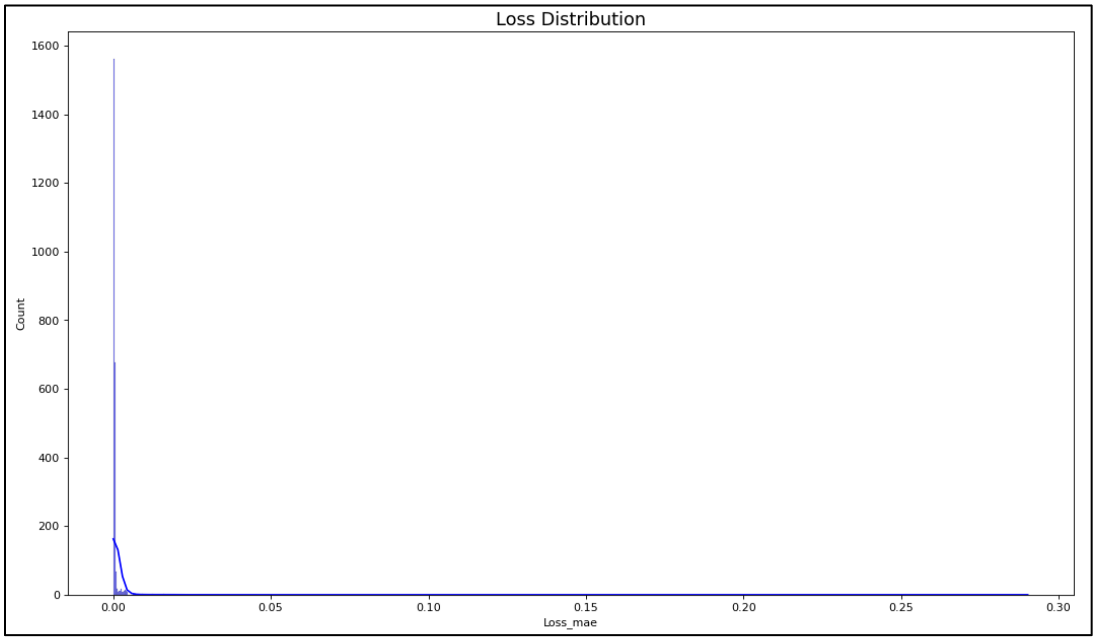


Figure 15. LSTM autoencoder loss distribution on training data for dry bulb temperature setpoint delta.

The trained LSTM model was used to predict the training losses on both the training and test sets, and the resulting losses/reconstruction errors are shown below in Figure 16. The red horizontal line on the plot represents the failure threshold value (i.e. mean + 3 sigma of the training losses), and the results for both the training and test data are shown on the same plot as a single series (relative order of the data points in this plot is the same as in Figure 12). Figure 17 shows a preview of the dataframe containing the time series data that is plotted in Figure 16, along with an "Anomaly" column that indicates which rows are considered failures/anomalies.

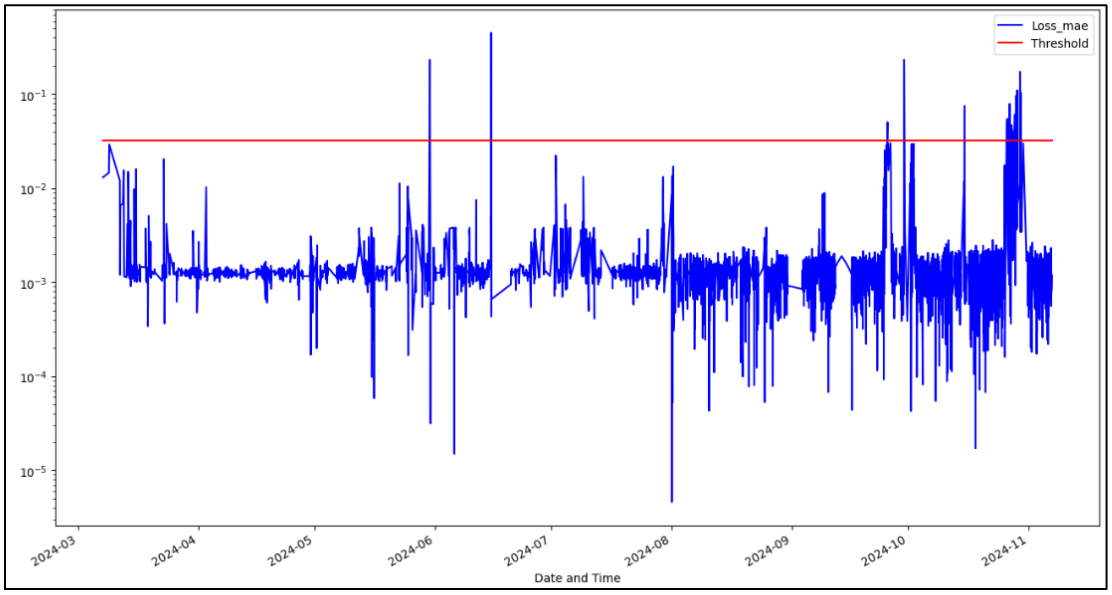
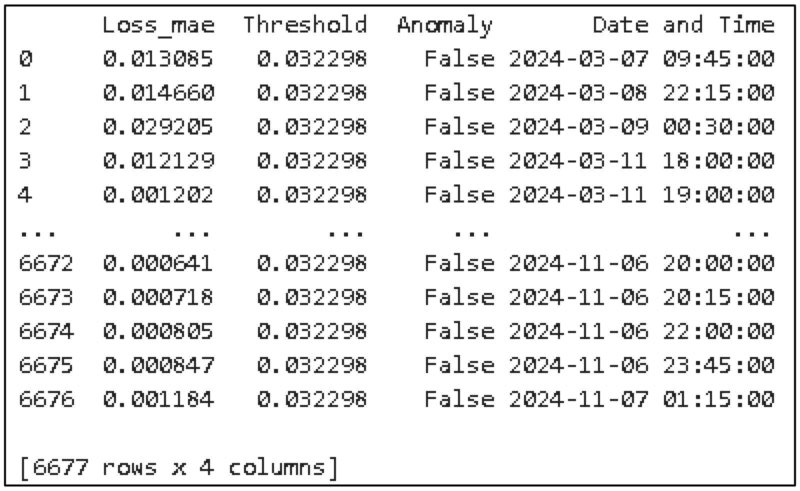


Figure 16. Time series of LSTM autoencoder model losses on training and test data. The failure threshold is shown as the horizontal red line.



```
      Loss_mae  Threshold  Anomaly       Date and Time
0     0.013085   0.032298    False 2024-03-07 09:45:00
1     0.014660   0.032298    False 2024-03-08 22:15:00
2     0.029205   0.032298    False 2024-03-09 00:30:00
3     0.012129   0.032298    False 2024-03-11 18:00:00
4     0.001202   0.032298    False 2024-03-11 19:00:00
...        ...        ...      ...                 ...
6672  0.000641   0.032298    False 2024-11-06 20:00:00
6673  0.000718   0.032298    False 2024-11-06 20:15:00
6674  0.000805   0.032298    False 2024-11-06 22:00:00
6675  0.000847   0.032298    False 2024-11-06 23:45:00
6676  0.001184   0.032298    False 2024-11-07 01:15:00

[6677 rows x 4 columns]
```

Figure 17. Dataframe showing LSTM autoencoder model losses on training and test data. The "Anomaly" column indicates which data points would be considered failures/anomalies.

The median and maximum losses on the full data set (training + test), along with the threshold values for each of the different setpoint deltas, are shown below in Table 1. Note that these values are expressed in terms of the scaled quantities and therefore are unitless.

Column	**Median loss**	**Max loss**	**Threshold**
Dry bulb temperature	0.000980	0.517	0.03573
Dewpoint temperature	0.00167	0.502	0.020601
N2 flow rate	0.000286	7.542	0.137364
CO2 flow rate	0.000364	34.055	0.182682
CO2 inlet concentration	0.00521	37.514	0.075288
TIC07 temperature	0.000429	0.431	0.058594
TK003 temperature	0.000969	422.84	0.005099

Table 1. Median loss, maximum loss, and threshold value for each of the setpoint deltas.

Next, the trained models were tested again on the most recent data available from the core tester in order to simulate the operation of the model in production. A 3-hour window of recent data was pulled from the core tester database and fed into the models after calculating each of the setpoint delta columns. The function used for applying the model to new data operated very similarly to the function outlined in the *Methodology* section (scale the data, calculate the predicted values using the pre-trained model, and plot the losses along with the threshold). See Figures 18 and 19 below for the dry bulb temperature delta trend observed during this window and the losses calculated based on the LSTM model's predictions. Note that the interval between data points in this time series is 60s, compared to the 900s that was used for training the models – this difference does not affect the performance of the model. The plot of the losses in Figure 19 shows that all data points in this series are well within the expected error range since none of the points fall above or even anywhere close to the threshold. This result makes intuitive sense since the time series of the deltas in Figure 18 shows relatively low variability (e.g. no data points outside of +/- 0.5 degrees C from the setpoint) and looks very similar to the time series of the data that was used for training the model originally, hence the small deviations between the actual and predicted values. The results for all the other sensors looked similar to this as well, with time series trends that resembled the training data and no data points identified as anomalies.

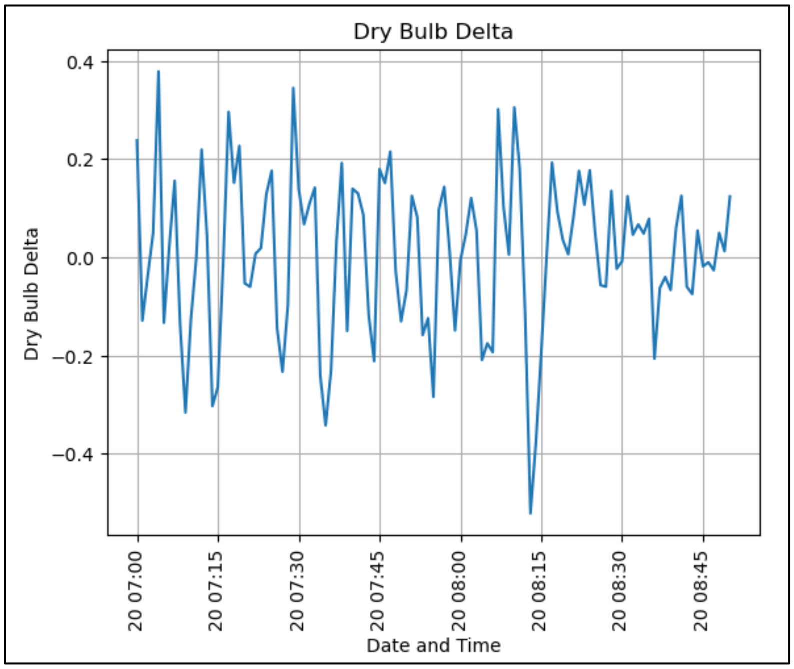


Figure 18. Dry bulb temperature setpoint delta trend during the time period of 11/20/24 06:00-09:00.

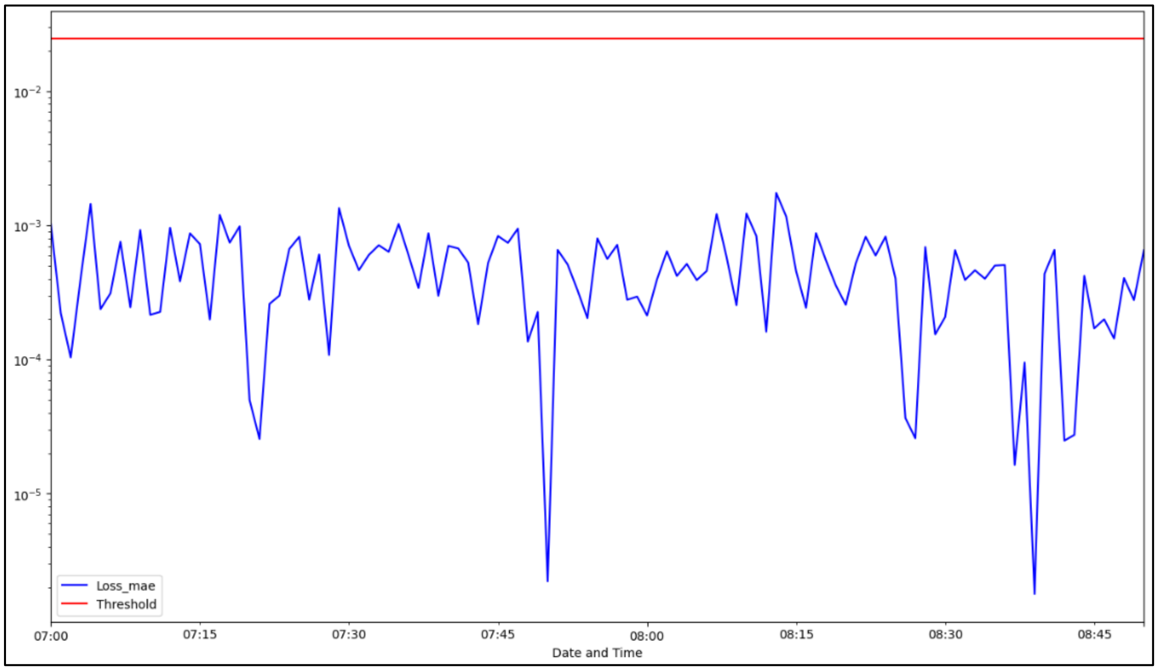


Figure 19. Time series of LSTM autoencoder losses calculated from the data in Figure 18.

The results shown here demonstrate that the LSTM encoder model performs effectively at detecting anomalies/change points on deltas between measured values and setpoints for the various sensors on the core tester. It is difficult to precisely quantify the efficacy of these models since the historical data available from the core tester at this point in time is limited and none of the data points identified by these models as being anomalies are known to have identifiable causes associated with equipment/process failures. However, given the capabilities of the LSTM autoencoder model demonstrated by Larzalere [6], along with the success of the models trained in this experiment at flagging the most suspicious data points in the time series for each of the different sensors that were examined and the sound logical basis for using these particular sensors/setpoint deltas as indicators for detecting equipment failures in the first place, the LSTM autoencoder model seems like a very suitable starting point for ZCS's burgeoning predictive maintenance program.

Conclusion and Future Plans

The goal of this project was to find a machine learning model that would be able to analyze time series data from the core tester and identify any patterns in the data that would indicate potential issues with the equipment and/or process. The most challenging aspect of this problem was the lack of a large, labeled dataset with past known failures to use for training the predictive maintenance model. The core tester is a relatively new piece of equipment with a limited history of past data to draw from, and while there have been a handful of events in recent months involving hardware failures on the unit, these events and the corresponding signals in the data were nowhere near sufficient to represent the full breadth of possible failures/data signals. As such, an appropriate model to use for this problem had to be unsupervised and capable of detecting different types of change points in time series data given only a limited segment of past data representing normal/healthy operation. The model selected was the LSTM autoencoder neural network model, which was trained and tested on time series data representing the deltas between various sensors on the core tester and their respective setpoints. Using a default anomaly threshold value of the mean plus 3 standard deviations of the training losses, these models proved to be capable of detecting moderate to large deviations from baseline in the setpoint deltas for all these different sensors. Furthermore, while none of these sensors have had any observable systematic drifts in the historical data, the use case tested by Larzalere [6] demonstrated that the LSTM model is quite capable of detecting such drifts. The next steps for this project will involve experimenting with LSTM models on other core tester sensors that have shown signals associated with past equipment failures but with more complex patterns (see *Methodology* section), testing and applying similar modeling approaches on other types of equipment that ZCS currently operates/will be starting to operate in the near future, and deploying the existing models to production via python REST API as Larzalere executed with the LSTM model from the vibration data example [7].

References

1. Küng, L., Aeschlimann, S., Charalambous, C., McIlwaine, F., Young, J., Shannon, N., Strassel, K., Maesano, C. N., Kahsar, R., Pike, D., Spek, M. van der, & Garcia, S. (2023, July 19). *A roadmap for achieving scalable, safe, and low-cost direct air carbon capture and storage*. Energy & Environmental Science. https://pubs.rsc.org/en/content/articlelanding/2023/ee/d3ee01008b#!divAbstract
2. "Zero Carbon Systems." Zero Carbon Systems, www.zerocarbonsystems.com/. Accessed 21 Nov. 2024.
3. M. Paolanti, L. Romeo, A. Felicetti, A. Mancini, E. Frontoni and J. Loncarski, "Machine Learning approach for Predictive Maintenance in Industry 4.0," 2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA), Oulu, Finland, 2018, pp. 1-6, doi: 10.1109/MESA.2018.8449150. keywords: {Predictive maintenance;Machine learning;Industries;Current measurement;Time measurement;Forecasting},
4. Amruthnath, Nagdev & Gupta, Tarun. (2018). A Research Study on Unsupervised Machine Learning Algorithms for Fault Detection in Predictive Maintenance. 10.13140/RG.2.2.28822.24648.
5. Aminikhanghahi, S., Cook, D.J. A survey of methods for time series change point detection. *Knowl Inf Syst* **51**, 339–367 (2017). https://doi.org/10.1007/s10115-016-0987-z
6. Larzalere, B. (2019, September 25). *LSTM Autoencoder for Anomaly Detection*. Towards Data Science. Retrieved from https://towardsdatascience.com/lstm-autoencoder-for-anomaly-detection-35f7e1a1e7e9

7. Larzalere, B. (2019, October 14). *Containerized AI for Anomaly Detection*. Medium. Retrieved from https://medium.com/swlh/containerized-ai-for-anomaly-detection-eb3e08225235
8. Olah, C. (2015, August 27). *Understanding LSTM Networks*. Colah's Blog. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/