INL/EXT-19-55368

Light Water Reactor Sustainability Program

Report for 2.2.1 Task 5: Develop and Document a State-Based Alarm System for a Nuclear Power Plant Control Room Using Machine Learning

Jens-Patrick Langstrand, Hoa Nguyen, Robert McDonald

August 2019

U.S. Department of Energy

Office of Nuclear Energy

INL/EXT-19-55368
Revision 0

Light Water Reactor Sustainability Program

Report for 2.2.1 Task 5: Develop and Document a State-Based Alarm System for a Nuclear Power Plant Control Room Using Machine Learning

Jens-Patrick Langstrand, Hoa Nguyen, Robert McDonald

August 2019

Idaho National Laboratory
Idaho Falls, Idaho 83415

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

IFE/INL-196543

SOW 14512

Report for 2.2.1 Task 5. Develop and document a state-based alarm system for a nuclear power plant control room using machine learning

Institute for Energy Technology

	KJELLER	HALDEN
Address	NO-2027 Kjeller, Norway	NO-1751 Halden, Norway
Telephone	+47 63 80 60 00	+47 69 21 22 00
Telefax	+47 63 81 63 56	+47 69 21 22 01

Report number	Date
IFE/INL-196543-2.2.1	2019-08-09

Report title and subtitle	Number of pages
Report 2.2.1 for Task 5: Develop and document a state-based alarm system for a nuclear power plant control room using machine learning	19

Project/Contract no. and name	
Contract no 196543	

Client/Sponsor Organisation and reference	
Idaho National Lab, USA	

Abstract

	Name	Date	Signature
Author(s)	Jens-Patrick Langstrand Hoa Nguyen Robert McDonald	2019-08-09	
Approved by	Andreas Bye	2019-08-09	

1. Background

Institutt for Energiteknikk (IFE) operates the OECD Halden Reactor Project (HRP). The organization has extensive experience from more than 20 years of research in human system interface (HSI) design and operation of nuclear power plant research simulators in the Halden Man-Machine Laboratory (HAMMLAB).

HAMMLAB serves two main purposes. These comprise the study of human behavior in interaction with complex process systems and the development, testing, and evaluation of prototype control centers and their individual systems. The aim of HAMMLAB is to extend the knowledge of human performance in complex process environments in order to adapt new technology to the needs of the human operator. By studying operator performance in HAMMLAB and integrating the knowledge gained into new designs, operational safety, reliability, efficiency, and productivity can be improved.

IFE also provides new and innovative technology to customers in the form of operational task-based displays, large screen displays, innovative eye-tracking programs, and innovative performance testing methods. IFE also provides expert support in both nuclear power plant operations and setting up and running operations-based experiments and workshops.

2. Introduction

Idaho National Laboratory (INL) has contracted IFE to support human factors research and the development of leading-edge technology to support control room operators as the U.S. fleet undergoes modernization and digitalization in legacy plants and control rooms. As the nuclear industry starts to shift to more digital controls and systems, these upgrades provide more information to the main control room in the form of digital signals and values, leading to an increase in the number of alarm points that are provided by the vendors in their digital control systems. New digital control rooms also shift from control boards that operators stand at, to soft controls on a computer monitor at a sit-down workstation, and alarm panels to a single display. This change to a single display has created the problem called alarm waterfalling, a situation in which the addition of new alarm points causes an overload of information to the operator during either a plant disturbance or other abnormal operating conditions. IFE’s goal is to find a workable solution to assist operators in both handling and understanding incoming alarms during emergency situations under these waterfall conditions.

3. Current Design

In legacy control rooms, operators might have 20 to 30 alarms to deal with on 12 separate control boards during a reactor trip. The new digital alarm system in the upgraded control room will display between 150 and 200 alarm points on a single screen. This overload of information prevents the operators from identifying abnormal alarms quickly, without having to scroll through the alarm list looking for abnormalities. One way to tackle this problem would be to create a state-based alarm system that would recognize plant states, identify alarms that are expected in those states, and suppress expected alarms from the alarm screen so that

unexpected alarms would be displayed, prioritized, and easily identified by the operators. This can be done today, but is labor intensive in that an operator or training instructor would be required to run scenarios and log those alarms to be suppressed. The hope is that machine learning (ML) can provide an alternative solution with less manual effort required.

4. Machine Learning for Unexpected Alarm Detection

The creation of traditional state-based alarm systems requires operators or trainers to manually specify the state conditions for the reactor at each state. The work described in this report differs in that instead of trying to manually define these states, researchers at IFE collected data and used ML to learn these states automatically. To achieve this, these researchers have tried two different approaches to model the problem using ML: first, Supervised learning, and second, Anomaly detection.

Supervised learning is employed when clear expected output is commensurate with a set of input. The use of supervised-learning is intuitive in this case because process signals can be treated as input and alarm-states as output. It is convenient as well because no data annotating step is required.

Anomaly detection is useful when accessible data contains predominantly either positive or negative samples. In this case, the ML model can attempt to learn positive or negative behavior and thereby detect anomalies when data differing from the learned behavior are provided. This can also be called Semi-Supervised learning because only one class or type of data is used during the training of the model.

5. Scope

The core deliverable for this project is a prototype alarm management/filtration system (AMS) that uses ML to filter expected from unexpected alarms during the operation of the generic Pressurized Water Reactor (gPWR). This approach to alarm filtering is novel, and much of the work involved exploration and testing of ML concepts and techniques to attempt to solve alarm overflow in digital control rooms.

As part of the work package, a workshop was held at INL to facilitate knowledge transfer. The workflow of collecting data, processing them, and training and deploying a ML model were addressed. Lessons learned and recommendations of how to use the ML system were also presented as examples of future steps.

Summary of deliverables:

1. A python script that processes output from IFE Teams Viewer tool into a format suitable for ML.
2. A trained ML model that can be used to detect anomalies (abnormal alarms).
3. A tool that interfaces with the simulator and processes data in real time to detect and flag anomalies (abnormal alarms) for the operator.
4. Knowledge transfer through a workshop at INL.

The whole process from collecting data to training and deploying a ML system will be covered in this report, as will lessons learned.

6. Data collection

In order to train ML models, IFE researchers needed first to collect meaningful data that capture the nominal behavior of the gPWR. They chose scenarios including full power and no incidents, full power with reactor trips, full power with malfunctions, to full power with reactor trip and malfunctions. The reason for choosing these scenarios was to allow us to focus on testing and iterating our ML models on a simpler problem, rather than initially spending an inordinate time on scenarios and complex combinations. Initially, IFE researchers believed they would need to create a data collection tool for this project; however, it turned out that an IFE-developed tool called Teams could be repurposed. In interest of time, this option was chosen.

6.1. Collection tool: Teams Viewer

To collect data from the gPWR, IFE researchers used Teams Viewer, a tool developed by IFE that allows the collection of synchronized sound, video, simulator signals, and simulator events, as well as having the ability to replay collected data. For our purposes, only the process, alarm signals, and process events were important. In total, IFE researchers ran the simulator 11 times with scenarios of various lengths and complexities. The real-time connection tool described in Section 6.16 could also be extended to function as a data collection tool as an option to the IFE Teams tool.

6.2. Run Descriptions

Run 1: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 1.5 hours with no failures initiated to collect nominal data.

Run 2: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 2 hours with no failures initiated to collect nominal data.

Run 3: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 4 hours with no failures initiated to collect nominal data.

Run 4 (Anomaly Test): The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 5 minutes and initiated a manual Reactor Trip from 100% and then stabilized the plant by using plant procedures E-0 and ES-0.1.

Run 5: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 30 minutes and initiated a manual Reactor Trip from 100% and then stabilized the plant by using plant procedures E-0 and ES-0.1

Run 6: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 10 minutes and initiated a manual Reactor Trip from 100% and then stabilized the plant by using plant procedures E-0 and ES-0.1

Run 7: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 10 minutes and initiated a manual Reactor Trip from 100% and then stabilized the plant by using plant procedures E-0 and ES-0.1. Once the plant was stable, IFE researchers inserted two anomalies.

Run 8: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 5 minutes and initiated a manual Reactor Trip from 100% and then stabilized the plant by using plant procedures E-0 and ES-0.1. IFE researchers inserted two anomalies, one at 32:51 simulator run time and one at 37:10 simulator run time.

Run 9: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 10 minutes and initiated a manual Reactor Trip from 100% and then stabilized the plant by using plant procedures E-0 and ES-0.1. Once the plant was stabilized, approximately 7 or 8 minutes after a trip of the Condensate pump. Then 10 minutes after stabilization a trip of a Circulation water pump.

Run 10: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 1 minute and initiated a manual Reactor Trip from 100% and then stabilized the plant by using plant procedures E-0 and ES-0.1. Once the plant was stabilized, inserted a malfunction for FCV-2200A, but no alarm was generated. After another 3 minutes, inserted a malfunction (tripping of the A Main Feedwater Pump). While re-establishing let down, received alarm XN07E03 twice.

Run 11: The simulation began at 100% reactor power with all systems in a normal configuration. The simulator ran for approximately 10 minutes and initiated a manual Reactor Trip from 100% and then stabilized the plant by using plant procedures E-0 and ES-0.1. Once the plant was stabilized, inserted malfunction EPS021 (loss of bus 1D2) at 13 minutes and 30 seconds. Inserted malfunction TUB04F Main Turbine vibration) at the 15 minute mark. Inserted malfunction PR506A (Pressurizer PORV 445A leak actuated by the tripping of the reactor).

6.3. Data Description

The data collected by the Teams Viewer are received in multiple files; only the files used will be covered here:

- valueInfo.txt: describes the content of the binary file containing all the simulator process signals. For every signal it contains its id, valid ranges, unit and lastly a short description.
- values.dat: contains all the process signals collected during a simulator run in binary format.
- processEvents.txt: contains all the process events with timestamps and identifications (IDs)—for instance, lamps turning on, valves opening or closing, and alarms on and off—that occurred during the simulator run.

6.4. Deliverable 1: Data Preprocessing Script

Before using the collected data to train the ML models, the data were preprocessed into a format that could be used by the ML models. The binary files containing the

collected process signals were parsed into csv and further packed into pickles, which allows for quicker loading of the data. See Figure 1 for an image with graphs of some of the process signals collected during this step.

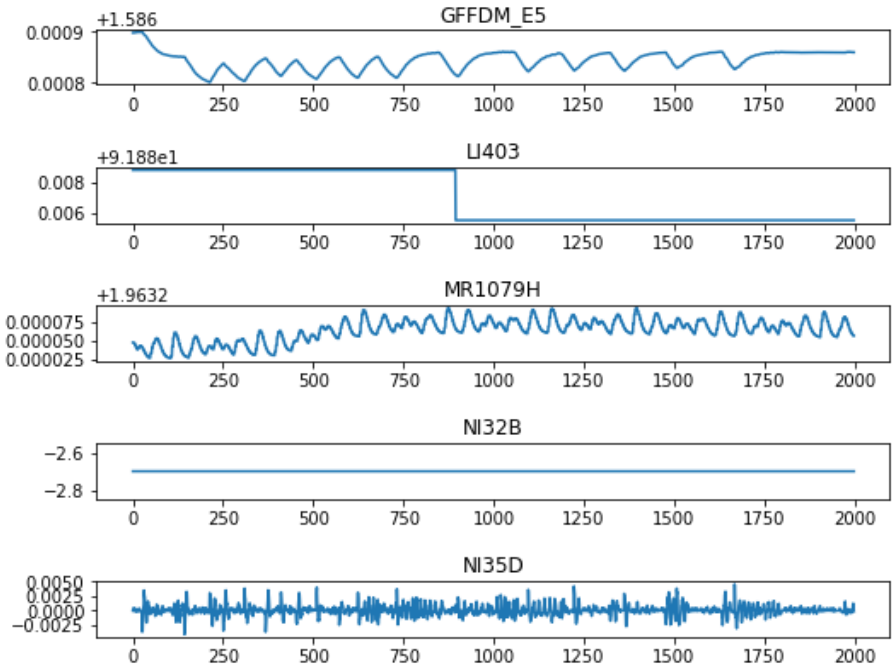

Figure 1. A selection of process signals used to train the ML systems.

The alarm signals were extracted from the process events file, using the timestamps, signal IDs, and alarm states to convert the events into continuous signals. The timestamps of both data sources were used to synchronize process signals with alarm signals.

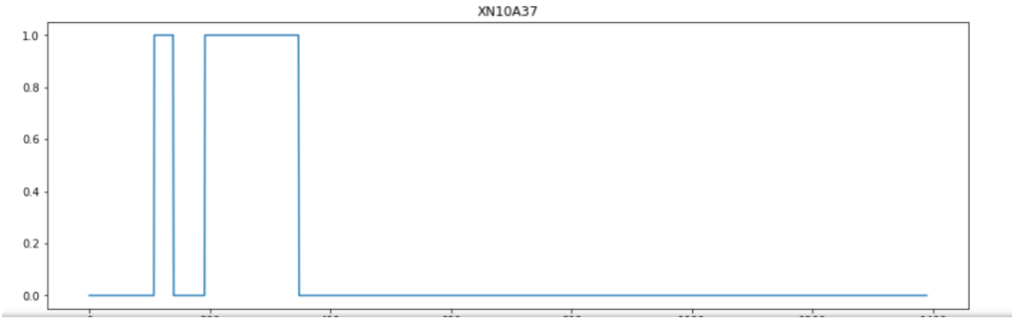

Figure 2. An example of an alarm signal that was created using the process events. A value of 0 represents the alarm off; 1 represents the alarm on.

Both data sources were loaded as data frames in a Python notebook script. In total, there are 655 process signals and 751 alarm signals. Last, the signals required normalization in order to make the ML training more effective and reduce the amount of time required to train a model. Without this step, the training would require more time and might even fail to converge on a good solution. An attempt to use the specified ranges from the file describing the ranges of each signal to normalize the values failed, however, because some signals had incorrect ranges specified. As a result, Min-Max scaling was performed on the available data in order to have data

within a valid, normalized range. Ideally, the predetermined ranges would have been used to normalize the input data because the ranges cover all possible values the signals can have. In this way, even if the full range is not represented in the training data, the model would have access to the full range when working on unseen data.

Because the written script relied on data collected by the IFE-developed Teams Viewer tool, it would be difficult for INL to directly utilize the script without having access to that IFE tool. However, one option could be to extend the real-time tool to also have data-collection capabilities, allowing INL to collect data.

6.5. Machine Learning Modelling

With the data preprocessed and normalized, it was possible to begin ML modelling using the collected data. The data were divided based on run conditions: the first three runs were used to train a baseline model that should be able to filter all alarms that occur during normal 100% power with all systems in normal configuration. The fourth run introduced a reactor trip and was used as a test of the baseline model; the model was believed capable of filtering the expected alarms from 100% power and normal system configuration while flagging all alarms caused by the reactor trip.

Runs five, six, and most of seven were used together with the first three runs to train the next iteration of models. These runs started with normal operations at 100% power and had a manual reactor trip occur during the run. The reactor was then, in each run, stabilized using operator procedures. These data were used to train the ML models to recognize the alarms caused by reactor trips as expected. The last part of the seventh run contained two anomalies and was used as a test to see if the models were able to flag those anomalies. The remaining four runs contained anomalies during or after reactor trips and were used as anomaly tests.

6.6. Tools and Frameworks

To handle the data preprocessing and preparation, as well as ML, Python was used, with frameworks such as Pandas, Numpy and Keras. Keras is a widely used ML framework that provides a higher-level, easier-to-use API for modelling. Keras also supports multiple popular ML libraries, such as Theano, CNTK, and Tensorflow. Tensorflow was selected for these tests. Tensorflow is a popular open-source ML library, developed by Google.

6.7. Supervised Learning Approach

In ML, supervised learning is a task that matches function-mapping input to desired output. In our case, the idea is to infer the current alarm state from the process signals.

$$Y = f(X) = \begin{cases} 0 \; if \; an \; alarm \; is \; Off \\ 1 \; if \; an \; alarm \; is \; On \end{cases}$$

X is the input array of process signals. *Y* is the output array of alarm signals. An alarm has two states, off or on, and is encoded 0 or 1, respectively.

As the predicted alarm state is compared with the actual state, any differences are marked as unexpected alarms.

6.8. Data Preparation

Due to limited scenarios on which to gather data, many alarms stayed consistently off in the dataset or were seldom triggered. It was estimated that only one-fourth of the dataset contained samples that had at least one alarm on. This led to the problem of an imbalanced dataset in that most samples were labelled as one class. In these cases, the algorithm will try to always produce the output as the majority class and still get a very high accuracy. To counter the imbalanced-data problem, IFE researchers first used oversampling by tripling samples with at least one alarm on; second, they avoided using accuracy as a metric for training.

6.9. Modelling

Deep Neural Network (DNN) was selected as the only algorithm to try. One big advantage of this model over other algorithms is its ability to produce output as an array. The network architecture is a factor to consider. Because the problem is relatively simple, a fully connected DNN was attempted. This involved constructing networks in two ways: first, by increasing the numbers of perceptrons in higher-level hidden layers and then reducing the numbers to produce the correct output size (i.e., a diamond-shaped model); the second method represented the opposite approach, it first reduced the numbers of perceptrons and then increased them again (producing an hourglass-shaped model). IFE researchers found that the second architecture began with lower performance, but it converged more quickly and achieved better performance overall. Both architectures used drop-out regularization to reduce overfitting.

Layer (type)	Output Shape	Param #
dense_11 (Dense)	(None, 703)	461168
dropout_10 (Dropout)	(None, 703)	0
dense_12 (Dense)	(None, 1406)	989824
dropout_11 (Dropout)	(None, 1406)	0
dense_13 (Dense)	(None, 2812)	3956484
dropout_12 (Dropout)	(None, 2812)	0
dense_14 (Dense)	(None, 1406)	3955078
dropout_13 (Dropout)	(None, 1406)	0
dense_15 (Dense)	(None, 703)	989121
dropout_14 (Dropout)	(None, 703)	0
dense_16 (Dense)	(None, 751)	528704

Total params: 10,880,379
Trainable params: 10,880,379
Non-trainable params: 0

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	335872
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32896
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8256
dropout_4 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 32)	2080
dropout_5 (Dropout)	(None, 32)	0
dense_6 (Dense)	(None, 16)	528
dropout_6 (Dropout)	(None, 16)	0
dense_7 (Dense)	(None, 128)	2176
dropout_7 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 256)	33024
dropout_8 (Dropout)	(None, 256)	0
dense_9 (Dense)	(None, 512)	131584
dropout_9 (Dropout)	(None, 512)	0
dense_10 (Dense)	(None, 751)	385263

Total params: 1,063,007
Trainable params: 1,063,007
Non-trainable params: 0

6.10. Results

The models were evaluated on a separated set of data that was not included in the training process. IFE researchers wanted the models to detect as many positive samples as they could. As the training dataset was imbalanced, accuracy was not used, but confusion matrices and recalls were chosen as our evaluation metrics. Figure 3 and Figure 4 show the evaluation for the diamond-shaped and hourglass-shaped models, respectively. When an alarm is on, the hourglass-shaped model is more likely to predict it as on.

```
Recall/Sensitivity:  0.720967429236138
|                |   actual OFF |   actual ON |
|----------------+--------------+-------------|
| predicted OFF  |       693379 |        5757 |
| predicted ON   |          941 |       14875 |
```

Figure 3: Confusion matrix and recall for diamond-shaped model

```
Recall/Sensitivity:  0.9994668476153548
|                |   actual OFF |   actual ON |
|----------------+--------------+-------------|
| predicted OFF  |       286381 |          11 |
| predicted ON   |       407939 |       20621 |
```

Figure 4. Confusion matrix and recall for hourglass-shaped model

Finally, the predicted and true alarm signals were visualized. Figure 5 illustrates examples of the visualization. The blue lines show true alarms signals received from the simulator. The orange lines represent the predicted output received from our hourglass-shaped model. Note that the failed predicted alarms may be unexpected alarms that should not be predicted as on (for filtering).

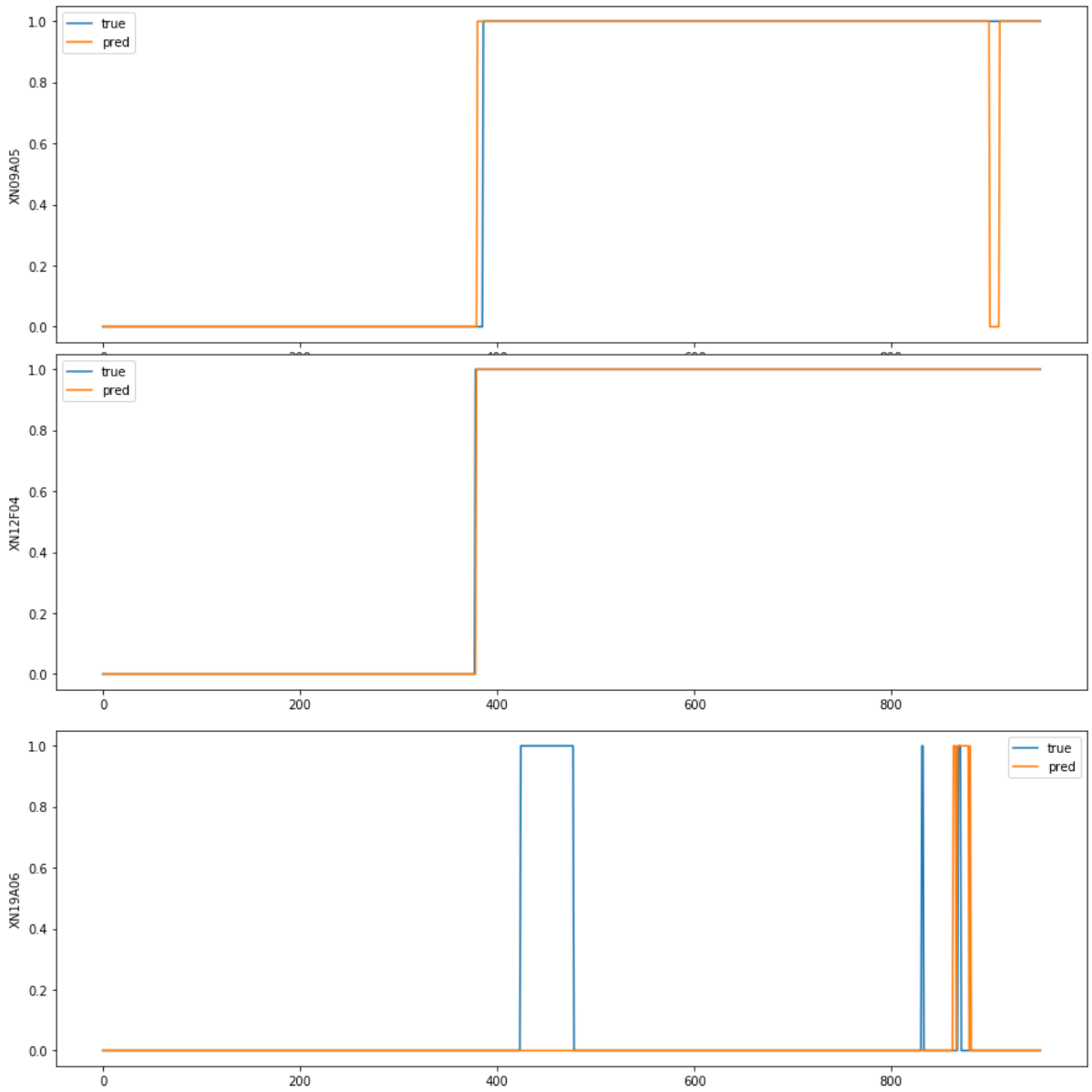

Figure 5. Visualization for three alarm signals: XN09A05, XN12F04 and XN19A06. The blue lines are true values received from the simulator. The orange lines are predicted values output from the hourglass-shaped model. The model has learned perfect matching behavior of signal XN12F04, almost-perfect behavior of signal XN09A05, and has learned to some degree the behavior of signal XN19A06.

6.11. Anomaly detection Approach

ML can also be used for anomaly detection. While supervised learning requires a balanced dataset of positive and negative samples, anomaly detection can be achieved using a biased dataset where only positive or negative samples are used. By treating the alarm filtering as an anomaly problem, IFE researchers saved time because only data from the normal state of the plant need be collected. In addition, there is no need to label the data as nominal or anomalous because all data would be nominal. Anomaly detection is a suitable technique to use when the bulk of data are nominal, and only a few examples of anomalies are present. Additionally, if the

difference between anomalies is large, the algorithm will still detect them as anomalies. Instead of manually defining states, data are collected that describe the normal behavior of the reactor and attempt to model this behavior in a way that allows the ML model to learn normal behavior. Therefore, it would be able to distinguish between nominal and abnormal data in the future. In some ways, this approach can be thought of as a true state-based system because the algorithm attempts to learn the whole behavior of a plant operating under nominal conditions.

See Figure 6 for an example of anomaly detection in signal processing. The first graph shows a signal whose nominal behavior is to oscillate at a regular interval, but then at the end of the signal, the pattern becomes abnormal. The second graph shows an anomaly detection system that has learned the nominal behavior of the signal in the first graph. It outputs 0 when the behavior of the signal is nominal and spikes when it is abnormal, resulting in spikes corresponding to the parts of the signal that show abnormal behavior.

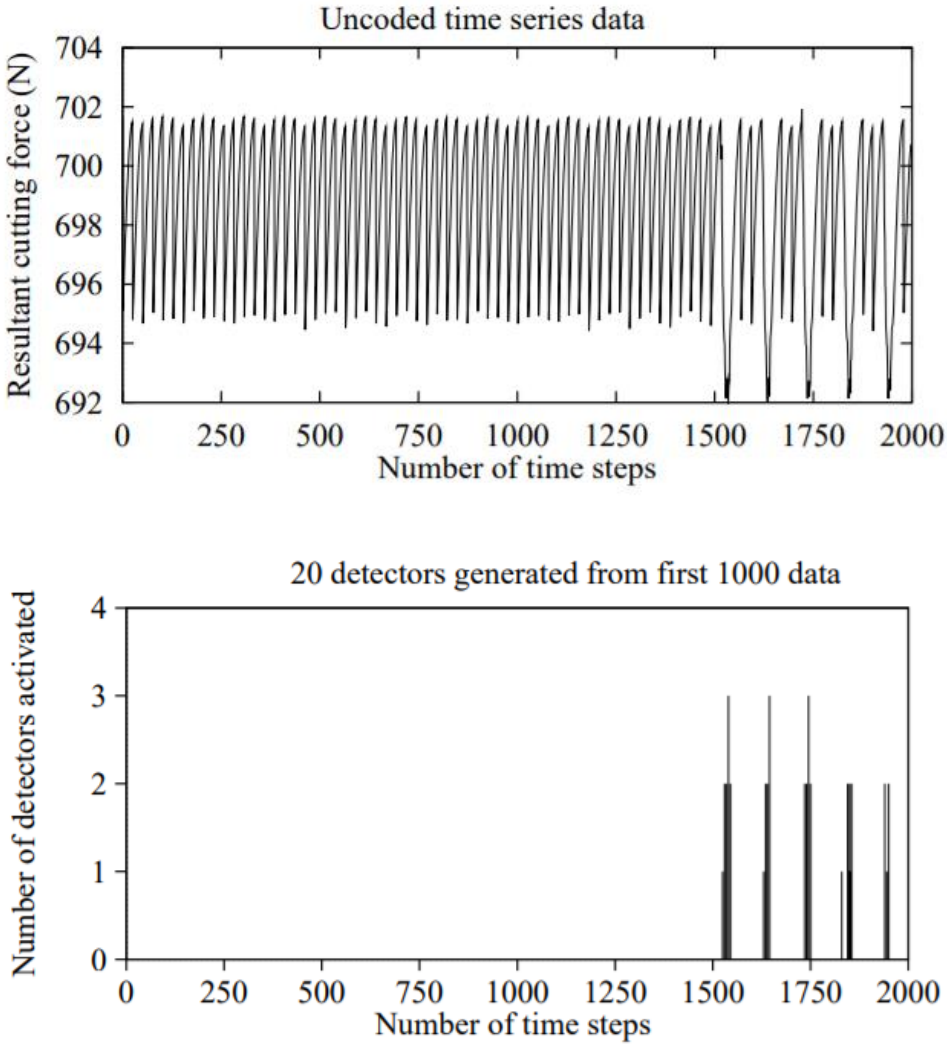

Figure 6. These graphs were taken from "Novelty Detection in Time Series Data using Ideas from Immunology," by Dipankar Dasgupta and Stephanie Forest.

In this case, anomalies are unexpected alarms that are of interest to the operator. By running the simulator from shutdown for refueling to full-power mode without introducing any failures or anomaly conditions, the collected data describe nominal behavior of the plant. An added benefit of using this approach is that it can detect, not only anomalous alarms, but also process signals. This information could perhaps be useful for the operators as well. Anomaly detection does not come without challenges; for instance, alarms can be nominal in certain plant conditions while also being anomalous under other plant conditions (i.e., contextual anomalies). Another problem is that noise can appear as anomalies and result in false positives. Because the framework was developed and prepared for data collection, training ML models, and interfacing with the simulator, the scenarios were deliberately kept simple for this first attempt. With the data collected from the 11 simulator runs, modelling and training began for the anomaly detector.

6.12. Data Preparation

Convolutional neural network (CNN) and recurrent neural network (RNN) were the types of neural network building blocks used for the anomaly detection approach. CNNs have been used to great effect when working with image data, but they have also been employed successfully with timeseries data. CNNs use primarily convolutions and pooling layers to produce results. RNNs were specifically made to handle timeseries data and have a memory component that allows previous input samples to influence future predictions. For this problem, both types were run separately as well as in combination into a type called recurrent convolutional neural network (RCNN).

To prepare data for training of the anomaly detector, it was first necessary to create generators that sequence data used in training the models. The generators are used in a way that can be thought of as data augmentation instead of using data in the sequence in which it was generated. The generators will pick a random starting point in the data and create a sequence of a specified length from that point. In this way, the number of valid sequences available for training is increased many times. The combined data from the process signals and alarm signals were split into 70% training data, 15% validation data, and 15% test data. These splits were then used by the generators to randomly sample valid sequences during training. The length of the generated sequences is a parameter that can affect training results and should be tuned during training to find a length that gives good results. An input sequence length of 90 seconds seemed to work well.

6.13. Modelling

For a network architecture, an auto-encoder architecture was selected. Auto-encoders can be used to denoise images, reduce the dimensionality of input data, and to color greyscale images, to name a few uses. The auto-encoder consists of three parts: an encoder, a decoder, and code (the encoded representation of the inputs). To train an auto-encoder, the input data is also used as the desired output data. The encoder reduces the dimensions of the input data to a predetermined encoding size, and the decoder then uses the lower representation and recreates the original inputs as closely as possible. The idea is to use the auto-encoder to learn features that occur during nominal operations of the plant and predict the expected

nominal behavior from the current plant state. See Figure 7 for a visual example on a signal auto-encoder. For this approach, the model must learn the connections between process and alarm signals in order to predict whether an alarm is expected.

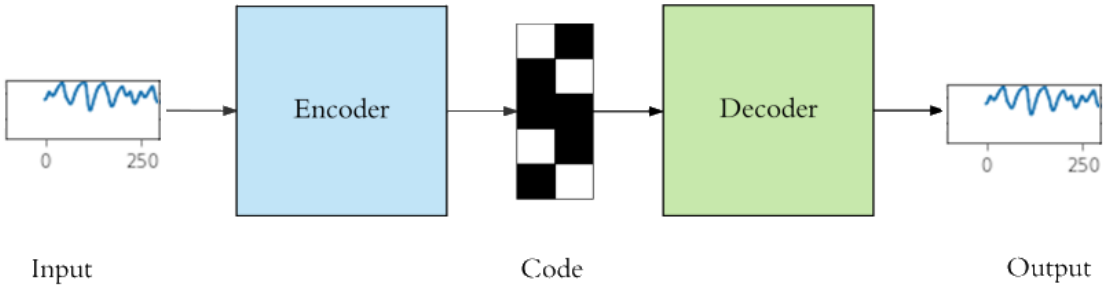

Figure 7. An overview image of the auto-encoder architecture (Image of model architecture borrowed from TowardsDataScience and modified slightly). The input signals are encoded into a lower dimension in the encoder part of the network. Then the code is decoded as closely as possible to the original signal.

To train the auto-encoder the datasets described in Section 6.12 were used. The training datasets are used to continuously train the model for many steps through many epochs. The validation set is used at the end of every epoch to test how the model generalizes to unseen data. After the training is complete, the test set is used to check the actual accuracy of the model using as yet unseen data. Most of the tested models took between 10 and 30 minutes to train; a bigger dataset with a wider variety of scenarios is expected to increase the time required.

A trained model can be used to detect unexpected alarms. To do this, the input signals are provided to the model when it is running inference, and the outputted expected behavior is deducted from the signals. The result is a difference between the actual state and the expected state. If the signals have near-zero difference, they are nominal. However, if the difference is large, an anomaly is present, and the signal will be flagged as unexpected.

6.14. Results

Scenario: 100% power and normal system configuration

The first test determined whether the model would be able to learn all expected behavior of the plant during 100% power and normal system configuration. This model was trained solely on the data collected during this plant condition. The model was able to learn the nominal behavior of many signals; an example can be seen in Figure 8.

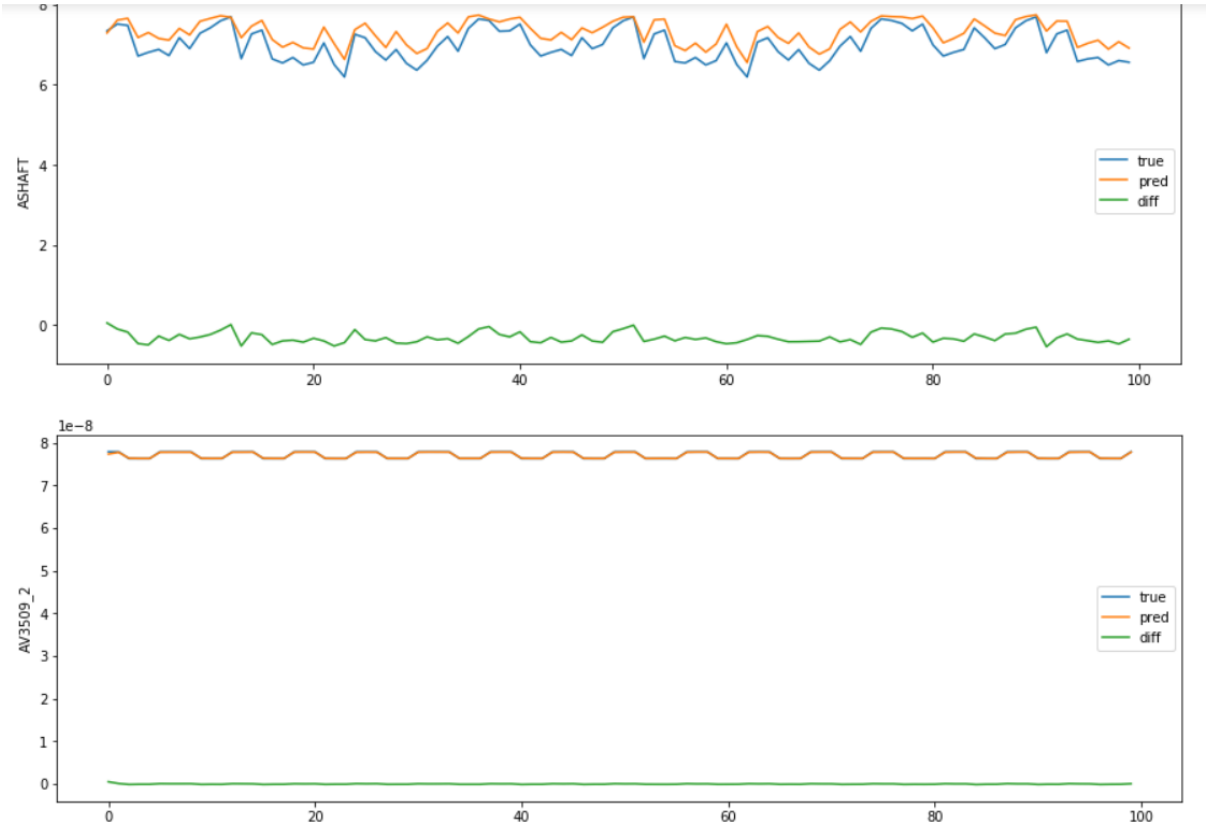

Figure 8. In these two graphs, the true (blue) signal is the signal given as input to the ML model. The decoded signal from the ML model is an orange line. The difference signal (in green) is the result of taking the input signal and subtracting the decoded signal. The result shows a value close to 0, which indicates that there is no anomaly or unexpected behavior for these signals.

To test the model's ability to detect unexpected alarms IFE researchers then used the small dataset collected during a 100% power and normal system configuration with a manual reactor trip. The alarms caused by the reactor trip should then be flagged as anomalies by the model, see Figure 9. The model wrongly predicted two signals and two instances within those signals in the training set and zero signals in the validation set.

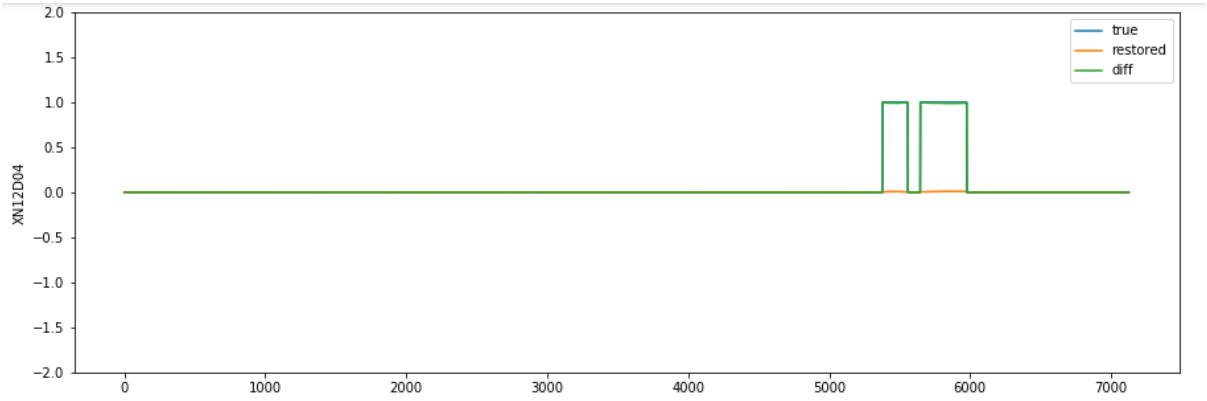

Figure 9. This graph shows an alarm caused by the reactor trip "REACTOR TRIP STEAM GEN-B LOW-LOW LEVEL." In this graph, the green difference line spikes up due to the

alarm not being predicted as expected by the model; therefore, the alarm is correctly treated as an anomaly in this case.

Scenario: 100% power and normal system configuration + manual reactor trip

Next, training was performed using 100% stable data as well as the reactor-trip data to see whether the model could learn that the alarms related to the reactor trip are expected. From the reproduced signals, the model performs quite well and can correctly anticipate many of the alarms resulting from a reactor trip (see Figure 10). From the alarm signals, the model wrongly predicted six signals and 17 instances within those signals in the training set and five signals and 3243 instances within those signals in the validation set. One signal is responsible for 3232 of the instances “XN11H05” which is “REACTOR TRIP MANUAL (LPPLRTMA)” showing that the model failed to learn this signal as nominal during a reactor trip.

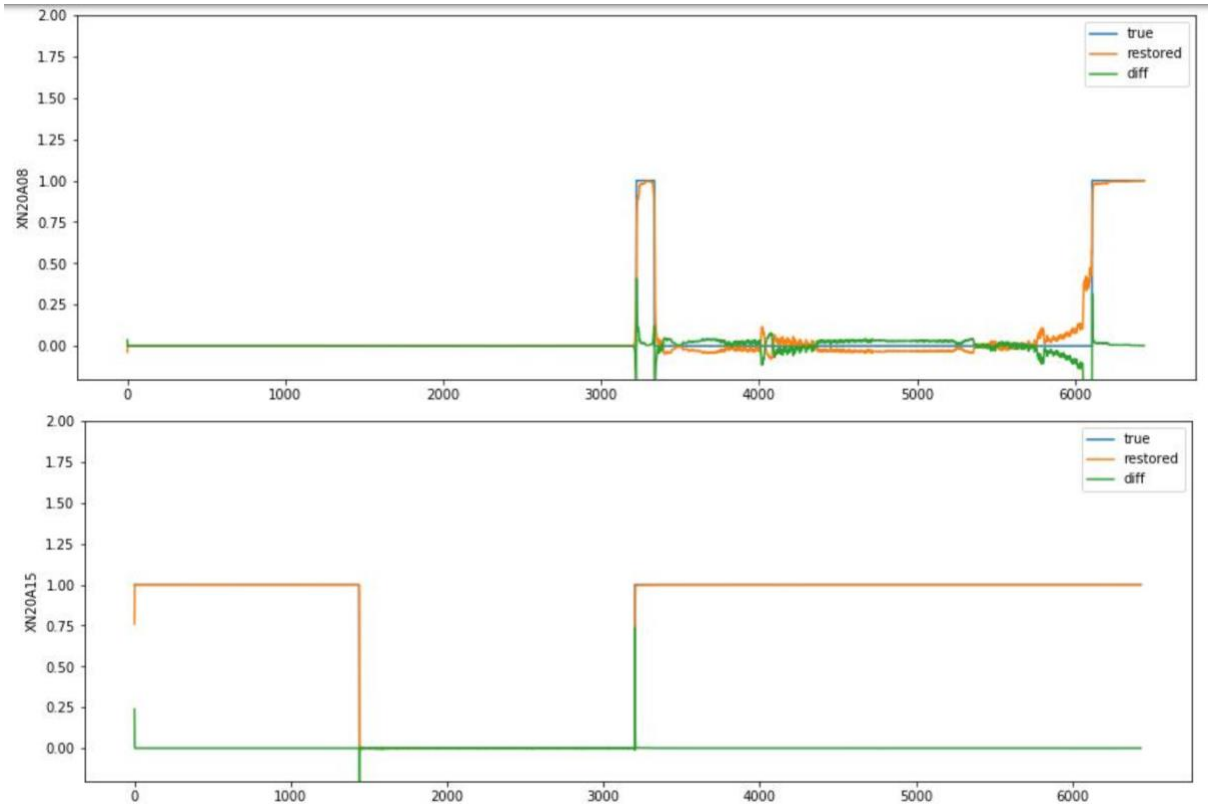

Figure 10. The first graph shows the “TURBINE AUTOMATIC LOADING STOP (JPPLSTL)” alarm and the second graph shows the “TURBINE AUTO STOP OIL LOW PRESS (JRPK109)” alarm. Both alarms are a result of the reactor trip. The model has learned to expect when the alarms turn on and off from correlating the process signals with the alarm signals. Some noise exists in the predicted signal, but by setting a good threshold and using a small-window activation requirement, this noise can be filtered, preventing false positives.

Scenario: 100% power and normal system configuration + manual reactor trip + anomalies

Again, the model can filter many of the expected alarms and flag some anomalies. To better evaluate the results of the models, more work is required on the scenario-

selection side; specifically, more care should be taken to record unexpected alarm IDs so that evaluations might more easily be performed offline while training and testing models.

6.15. Deliverable 2: Trained Machine Learning Models

As part of the tool described in the next section, some of the ML models IFE researchers trained for testing purposes from both the supervised learning and anomaly detection approach have been included. There are some steps required to go from a newly trained model to one that can be deployed and used in an application:

- First, the model must be exported, either to two files, one describing the model architecture and one containing the model weight, or to one file containing both the architecture and the weights.
- Second, the model must be converted from a Keras model (.h5) to a frozen Tensorflow model (.pb).
    - During this step, it is important to find the name of the input and output layers of the model as these will be required when deploying the model for inferencing later.
    - Information about the model's input shape is also necessary.

6.16. Deliverable 3: Simulator and Machine Learning Interface Tool

To connect trained models to the simulator so it can receive live data and perform alarm filtering, a tool in C# was developed that uses ProcSee to communicate with the simulator (see Figure 11). ProcSee uses a publication and subscription model that allows it to subscribe to the process signals of interest and receive messages containing the value of these signals approximately three times per second. Using the publication module, IFE researchers can publish the output of the ML model to update an alarm display made for the testing of our system.

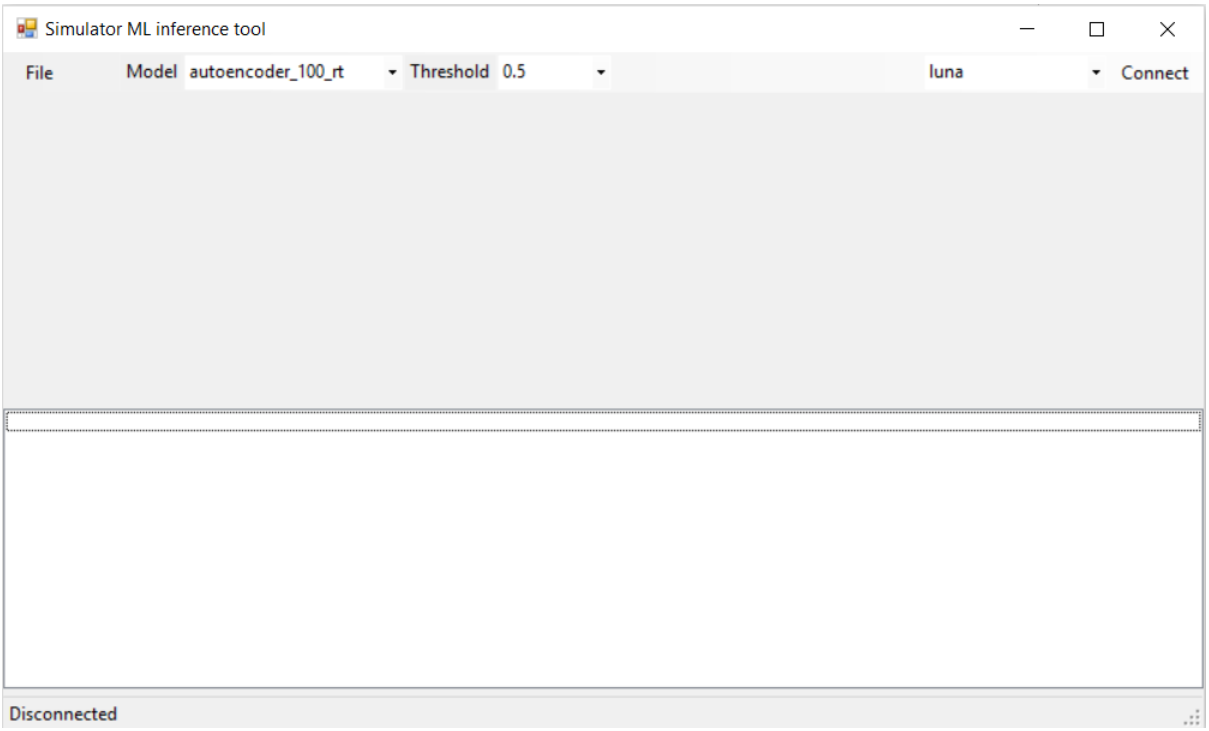

Figure 11. A snippet of the simple tool created to communicate between the simulator and ML models.

The tool allows the user to select the ML model to load and the simulator with which to connect. To load a model, a json file with the same filename as the ML model file must also be provided. The json file must contain five values:

1. “friendly_model_name” The name that is used when displaying the model option in the dropdown box.
2. “input_name” The name of the input layer of the frozen ML model.
3. “output_name” The name of the output layer of the frozen ML model.
4. “nmb_input_signals” The number of signals input to the ML model. This number is used when preparing the input data structure used when inferencing with the ML model. Note: this number must match the number of inputs in the input layer of the model used.
5. “input_sequence_length” The length of the input sequence used when running model inference. If the value is greater than 1, it will use a time window of that size when running model inference. Note: this number must match the sequence length used when during the training of the model.

Once loaded, the data received from the simulator signal subscription are used to run inference with the loaded ML model. The output produced by the model is then compared to the original alarm states received from the simulator. If there are discrepancies, they are considered as unexpected and flagged as such. In addition, it is possible to change the threshold value used when processing the output of the ML model during runtime.

After all signals have been processed, a message is constructed and sent to the simulator with a list of the alarms that have changed state and their new state, as predicted by the ML model. Once received, this message updates the alarm display mentioned earlier (see Figure 12). The display was created to facilitate comparison

between the alarm lists. The tool will be made available to INL; however, modifications might be necessary to connect to INL's simulator, but ProcSee is available for use at INL as well. Minor modifications might be necessary to communicate with INL's gPWR simulator.

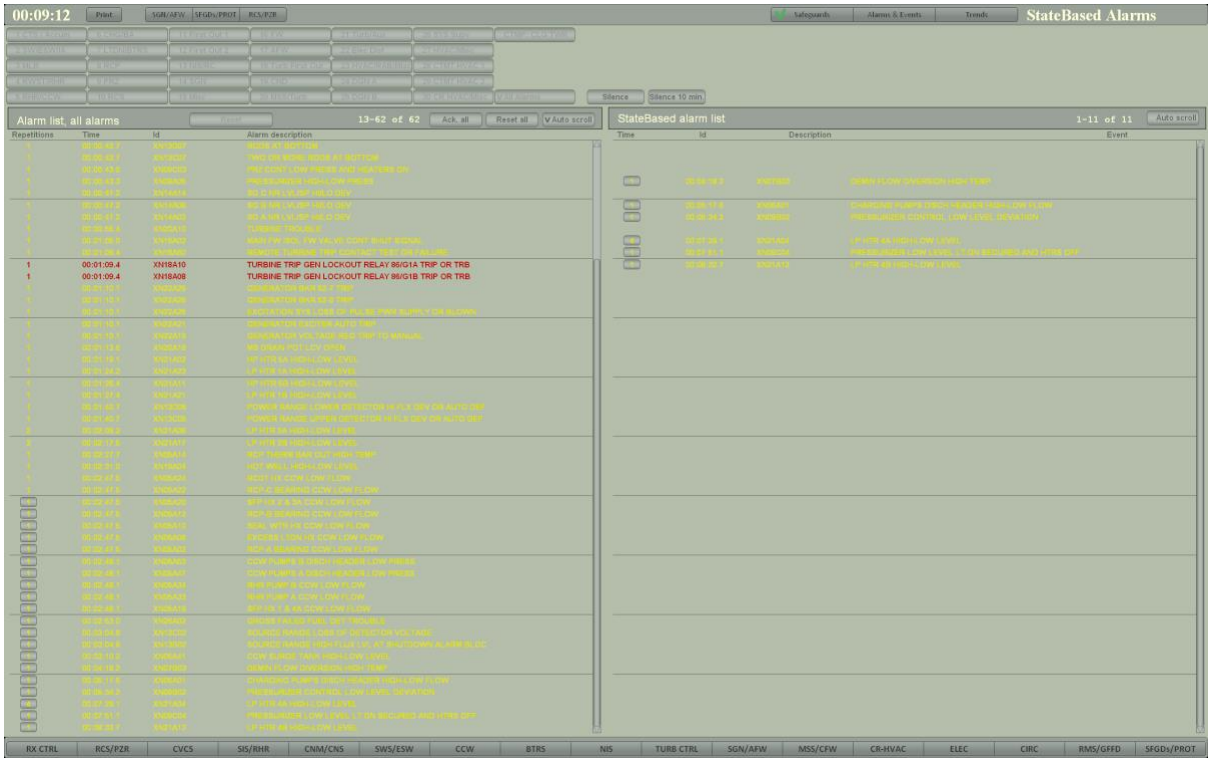

Figure 12. A screenshot of the operator display, showing both alarm lists side by side for easy comparison. The list on the left contains the original alarm list while the right list shows the ML-filtered alarm list.

The display developed for showing the alarm lists is very simple and was used only to test the ML model's filtering capability. This does not mean that the display is the best way to visualize the results of the ML system.

6.17. Real-time Test Results

The ninth scenario inserted an inadvertent reactor trip from 100% power. Two minutes after the trip, a leak from the Reactor Coolant System (RCS) was inserted into the Component Cooling Water (CCW) system at about 100 GPM.

Models were tested from the supervised and anomaly-detection approaches with difference decision thresholds, for example 0.5, 0.75 and 0.9. The classification models immediately produced outputs while the auto-encoder needed 30 seconds to prepare a time window before it could start producing results. An experienced operator verified the accuracy of filtered alarms. Models were still limited in the number of unexpected alarms they could filter. Some alarms were correctly filtered, some were incorrectly filtered, and some that should have been filtered were not. Models were adjusted after the testing, but insufficient time was not available to test again.

Figure 13 shows a screenshot of the operator display, showing an alarm list beside the filtered alarm list. Figure 14 visualizes a correctly filtered unexpected alarm named XN10A37. The model predicted it should be off, but in reality, it was on. The alarm was marked as unexpected. Figure 15, on the other hand, depicts an incorrectly filtered alarm signal.

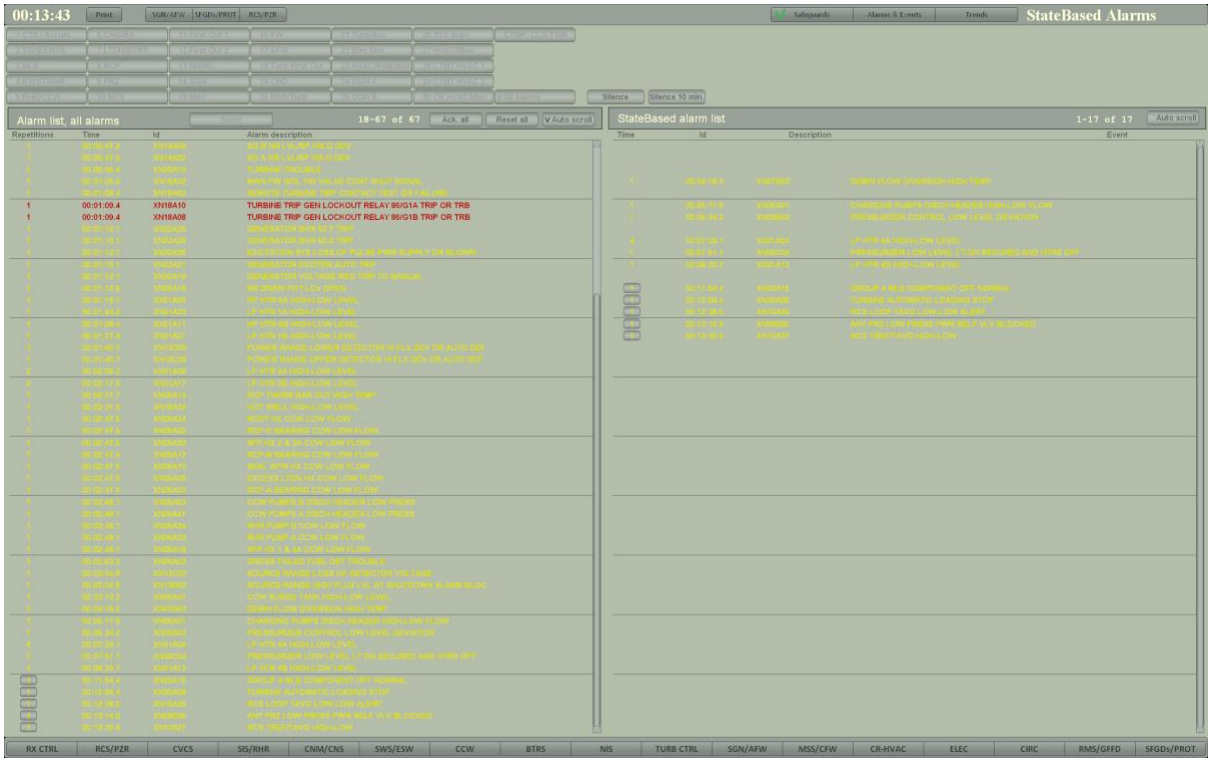

Figure 13. Screenshot of a testing scenario with the alarm list from the simulator on the left and the classification model's output on the right.

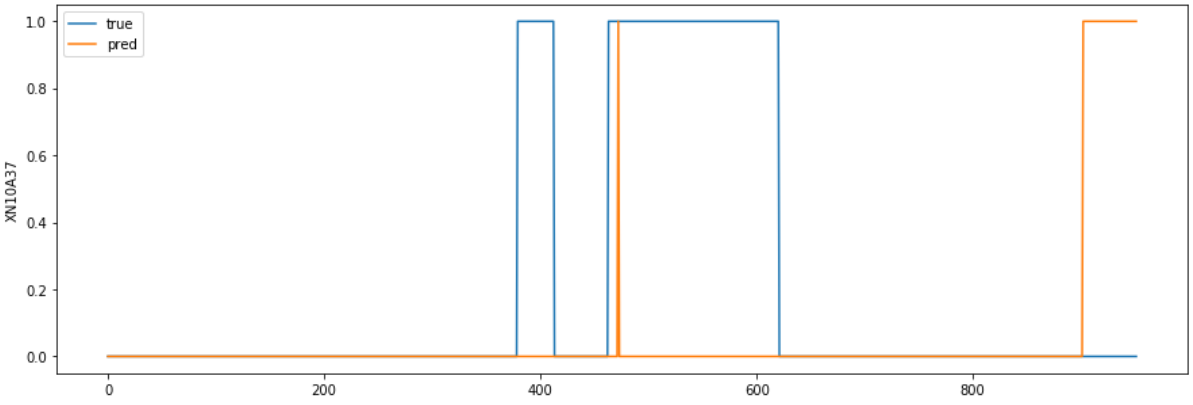

Figure 14. Visualization of a correctly filtered alarm signal XN10A37.

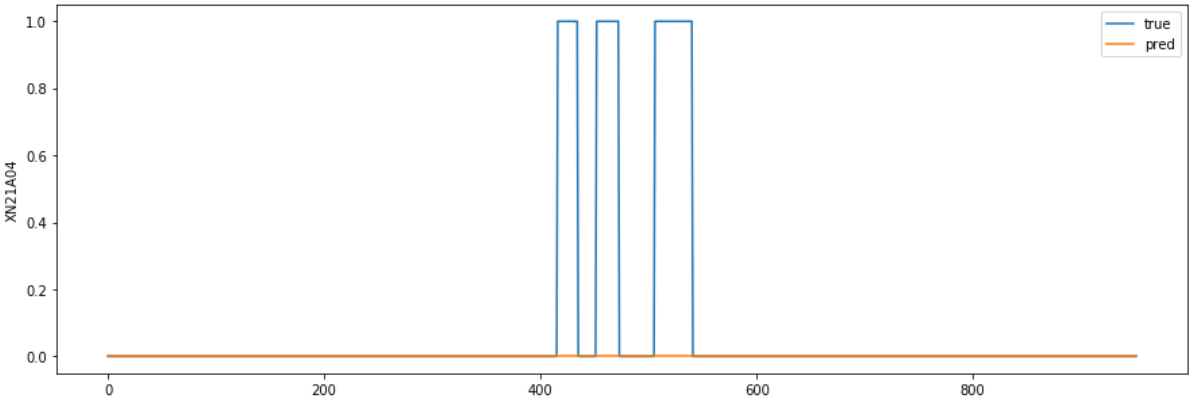

Figure 15. Visualization of an incorrectly filtered alarm signal XN21A04.

6.18. Deliverable 4: Knowledge transfer workshop at INL

A knowledge-transfer workshop was conducted at INL in June 2019. Various INL staff and three IFE researchers attended the workshop. The workshop was divided into two sections, one for work with ML at IFE in general, and one specifically for this project. The second session covered the steps taken in the project up until June, including data collection, the supervised classification approach, the auto-encoder anomaly-detection approach, and the simulator integration with trained ML models.

6.19. Initial Conclusions and Continuation in to the Next Year

Because of the necessity to develop and prepare the framework for data collection, training ML models, and interfacing with the simulator, the scenarios were deliberately kept simple during this first attempt. Although the results are still limited, the ML approach for solving the alarm-waterfall problem seems promising. The project was productive in that it helped researchers gain knowledge, not only on data modelling, but also on plant data in general. With the framework for collecting data, training ML models, and real-time testing in place, a future focus on scenario development, ML modelling, and testing for future work is anticipated.

Performing real-time testing was helpful in generating understanding of the nature of the problem and clarifying exactly where the focus should be in modelling. This knowledge allows better preparation during offline modelling and testing before real-time testing. A recommendation is to record testing scenarios more thoroughly, to enable work on the models without scheduling time on the simulator and operator until necessary.

Because INL has an interest in developing state-based alarms for turbines, the next steps for this study involve IFE focusing the ML-based approach to turbine control system state-based alarms. Most of this year's work can be reused for developing a state-based alarm system for turbine control. In parallel, INL will develop a state-based alarm system for turbine control using human subject matter experts. IFE and INL will then compare the state-based alarm system developed using ML to the state-based alarm system developed by human subject matter experts. This comparison is a straightforward approach to performing the necessary step of validating the effectiveness of using ML to develop a state-based alarm system. Differences in performance will be identified and provided as feedback to the

developers of the ML approach to modify and improve the effectiveness of this technique. A comparison of the effort required to develop the state-based alarm systems can also be made. Data can be collected by INL, or via virtual private network so that Halden can collect data.

6.20. Abbreviations

AMS	alarm management/filtration system
CCW	Component Cooling Water
CNN	convolutional neural network
DNN	deep neural network
gPWR	generic pressurized water reactor
HAMMLAB	Halden Man Machine Laboratory
HRP	Halden Reactor Project
HSI	human system interface
HSSL	Human System Simulation Laboratory
IFE	Institutt for Energiteknikk
INL	Idaho National Laboratory
ML	machine learning
RCNN	recurrent convolutional neural network
RCS	Reactor Coolant System
RNN	recurrent neural network