Automating ATLAS control room anomaly detection with deep learning

Avery Hanna,[1,2] Mario Campanelli (Supervisor),[3] and Antoine Marzin (Supervisor)[1]
[1)] *CERN, European Organization for Nuclear Research 1211 Geneva 23, Switzerland,*
[2)] *Physics Department, Tufts University, 574 Boston Ave, Boston, MA, USA*
[3)] *Physics Department, University College London, Gower St, London WC1E 6BT, United Kingdom*

(Dated: 29 August 2024)

To ensure high-quality data acquisition at ATLAS, the detector status is monitored by a team of shifters in the control room where they watch plots of the incoming data and compare them with the expected standards. We propose the use of an online anomaly detection model to facilitate the detection of issues during data-taking and decrease the workload of the control room staff. Our model is a predictive long short-term memory autoencoder that takes in time-series data on a range of Level-1 rates and instantaneous luminosity and then, via unsupervised learning, learns to predict how those rates will change such that the level of error between the prediction and real data can be used to classify the data as clean or anomalous. We show that our model effectively detects anomalies in all features and that such an approach shows promise for online use in the control room. This model can easily be adapted to run in real-time, alerting shifters to potential anomalies as the data comes in.

I. INTRODUCTION

ATLAS, a general purpose detector at the LHC at CERN, is a critical tool in expanding our understanding of the universe at the most fundamental levels. For the work at ATLAS to be of any use in the search for dark matter, in understanding the Higgs boson better, or in any of the other applications we need to get good data out and keep things running smoothly. Data quality monitoring (DQM) is generally divided into two areas: online and offline. Online DQM involves monitoring the data in real time and allows for intervention to correct problems that may arise. Offline DQM is a more rigorous sweep done after acquisition to review the quality of data that has been stored away and thus its usefulness for actual analyses. In this report, we focus on the online step.

To ensure high-quality data acquisition at ATLAS, the detector status is monitored by a team of shifters in the control room. This can result in inconsistencies in anomaly classification as different people may have different criteria in mind when watching for anomalies. Each person is also working an 8-hour shift and monitoring many plots which naturally results in a certain level of human error. Finally, this current system involves a high personnel demand and adds to the workload of the many PhD students, postdocs, and others who work these shifts. We aim to address these problems, improving the quality of anomaly detection (AD) and decreasing the workload of the control room staff by developing a machine learning model to watch the incoming time-series data on the status of the detectors and flag anomalies.

Research in AD has been of utmost importance to the experiments at the LHC which require innovative approaches to filter good and interesting data out of the masses of output from their detector systems. Investigations on AD approaches for DQM are currently underway at other LHC experiments in addition to our work at ATLAS. Many of these approaches rely on autoencoders[1,2], especially variational autoencoders[3].

II. DATA SELECTION AND PROCESSING

With almost 1 billion collisions per second, if we were to read out all the data produced at ATLAS it would amount to 60 TB of data per second which is too much to handle let alone store[4]. To address this challenge, the data passes through the trigger and data acquisition (TDAQ) system which filters down the data just to what is most relevant to the researchers using this data. The first pass in this filtering is the Level-1 (L1) trigger which is based on custom hardware to reduce the acquisition rate from 40 MHz to 100 kHz[5]. In our model, we focus on this level of the data to keep as much information as possible.

TABLE I. Definitions of input features used in model.

Variable Abbreviation	Meaning
MSPC_FW	Muon rate in forward detector on side C
MSPC_EC	Muon rate in endcap on side C
MSPC_BA	Muon rate in barrel on side C
MSPA_FW	Muon rate in forward detector on side A
MSPA_EC	Muon rate in endcap on side A
MSPA_BA	Muon rate in barrel on side A
L1_MU12BOM	Central muon rate with pT value of 12 GeV
L1_MU14FCH	Muon rate with pT value of 14 GeV
L1_eEM26T	Tight electron rate with energy greater than 26 GeV
L1_eEM26M	Medium electron rate with energy greater than 26 GeV
L1_jXE120	Missing transverse energy greater than 120 GeV
L1_jJ500	Small-R jets with 500 Et threshold rate
L1_jJ180	Small-R jets with 180 Et threshold rate
L1_jJ160	Small-R jets with 160 Et threshold rate

Here we focus on using L1 trigger item rates and L1 muon sector logic inputs — considered primary items since they are not pre-scaled and are used as main triggers in physics analyses. The trigger rates represent the frequency with which we detect different particles with different energy thresholds. We include rates looking at electrons, muons, missing transverse energy, and jets. The muon sector logic inputs give us insight into the muon rates by section of the detector. We also incorporate pileup — the average number of collisions per bunch

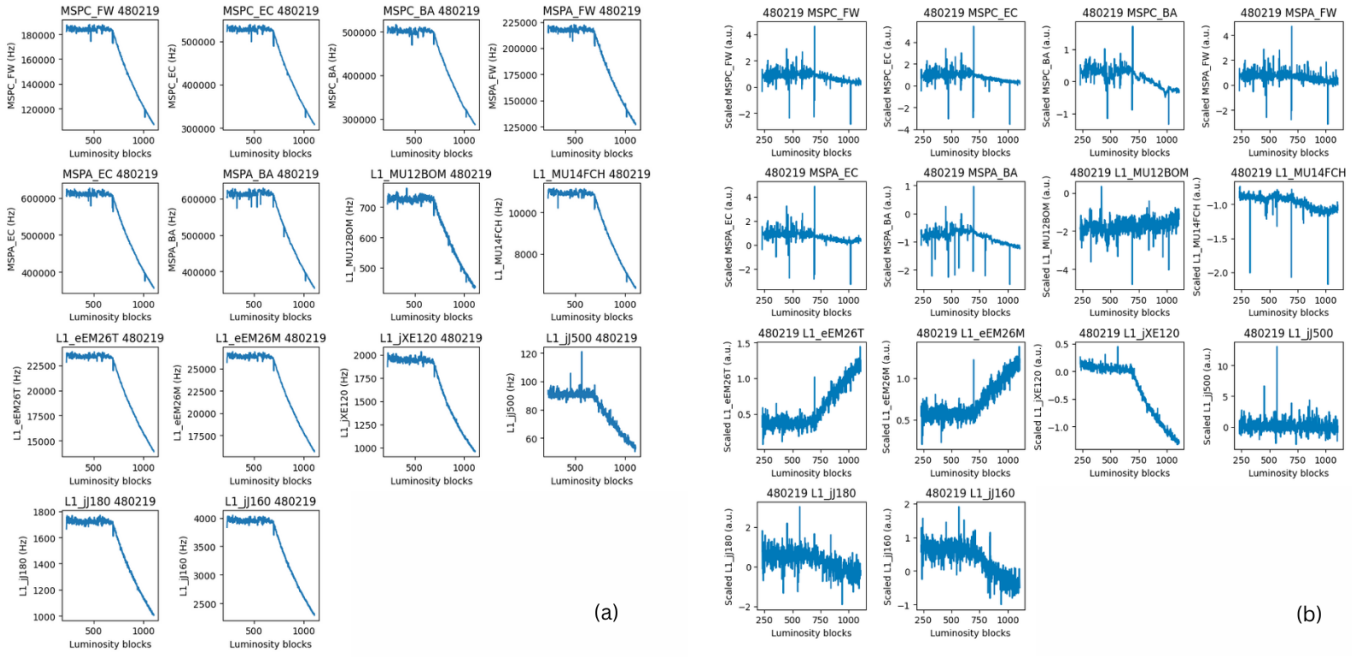

FIG. 1. Test data by feature. a) Raw data for region of interest as acquired directly from database. b) Data after normalizing by pileup and scaling with robust scaling procedure. Shows data as it will be when input into model.

crossing which is kept constant for around the first half of the run and then decays toward the end as there are fewer and fewer protons in each bunch[6]. Rather than including it as its own feature, we normalize the features discussed above by it. As seen in Figure 1, this results in a flattening of the plots of each feature in time. We also note that we see more variation and spikes in our data since we are looking at a smaller y-axis range. This is helpful for our model because via our scaling and loss functions we can treat these fluctuations as outliers and limit their influence on our model, instead learning the real trends in the data. In particular, we apply robust scaling which centers the data around 0 by shifting by the median and scales according to the interquartile range[7].

We get the raw input data from archived monitoring data by feature and merge them all together based on the timestamp. In the end we want a single data point for each luminosity block (LB) where an LB is a period in time (around a minute) where the conditions are assumed to be unchanged. To satisfy this condition, we pass over the data and when there are multiple data points for a single LB, we average them into one. If we identify a missing LB or a series of five or less missing LBs, we take the average of the data points on either side of the gap to fill it in. If there is a gap greater than five, we treat each section as a separate run.

In Table 1 we see the details of the features used in our model. Note that our current model uses only 14 features but the range of features monitored in the control room even just at the L1 trigger desk is much larger, not to mention the inputs available beyond this subsection of the data. We chose these features since they are representative of a range of particles but are similar in their behavior, staying relatively constant with constant pileup and decaying with pileup. Some of the features we considered including but ended up dropping for this initial iteration of our model because of the consistently large variations between real and predicted values were the busy rates which indicate when a queue has reached capacity so data is dropped and the empty rates where there are no collisions.

III. MODEL ARCHITECTURE

In the process of defining our architecture, we explored the use of autoencoders and long short-term memory (LSTM) networks using TensorFlow, motivated by prior research in similar scenarios[2]. Here we expound on the development process by discussing several iterations of our model to illustrate our motivations for and confidence in the final product.

A. Autoencoders

Our first pass at a model was a simple autoencoder that took one sample in time and attempted to reproduce it. An autoencoder is made up of an encoding network and a decoding network where in between we get an encoded representation of our data with a reduced dimensionality. The model is trained exclusively on clean data — runs with a normal amount of fluctuation and no significant anomalies — so that when it is given clean data in deployment, its output is quite close to the original input. On the other hand, if it sees an anomalous input in deployment, its procedure for generating the output will

not be as effective and we will observe a higher mean-squared error (MSE) — as defined in equation 1 — between the output and input values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2 \quad (1)$$

This model was quite effective at reproducing the input it was given and when MSE was plotted against LBs, the spikes occurred at LBs where there were also spikes in the input data. This was a promising indicator that the model could be used to identify anomalies in the data. The pitfall of this model, however, was that it only looked at a single LB of data. So even if it could identify anomalies in the data, it would only be effective for single point anomalies and it would not have the capacity to pick up on small variations that occur over time that we may still want to flag as anomalies. Additionally, these point anomalies are some of the easiest to spot as a shifter watching the data by eye so while a model like this could help, it does not particularly improve on the current state of human-based anomaly detection.

B. LSTM networks

To move beyond detecting single point anomalies with an autoencoder, we turned to LSTM networks — a type of recurrent neural network (RNN). As an RNN, LSTM networks incorporate feedback loops so that the output of past samples is included in making decisions on the current sample. However, it improves upon them by separating the paths for long-term and short-term memories to avoid the problem of the exploding/vanishing gradient that comes up with RNNs[8].

In our case, LSTMs are helpful because they allow us to look across several samples in time, giving the model more context about what has been happening with the data as it makes its decisions.

As a first step for incorporating LSTMs, we added an additional LSTM network acting on the output of the earlier autoencoder such that the LSTM network took 5 of the MSE values from the autoencoder and predicted what the next MSE value should be. Then by comparing the prediction with the real value and considering the error, we got an indicator of whether there was an anomaly present or not in the data.

While this worked to an extent, it did not improve over the original autoencoder structure on its own. The problem with this model was that a full sample of input data got compressed down to a single value before it was passed to the LSTM so it was hard for the model to learn any real significance that reflected what was happening with the input features. To address this, we combined the two separate networks into one with an LSTM autoencoder.

C. Non-predictive LSTM Autoencoder

As a first step in combining the benefits of these two approaches, we build an LSTM autoencoder which functioned the same as the autoencoder described above but instead of dense layers in the encoder and decoder subsections, we used LSTM layers. We compared the reconstructed time series data to the original input and used this MSE to classify the data as clean or anomalous based on how it compared to our threshold value.

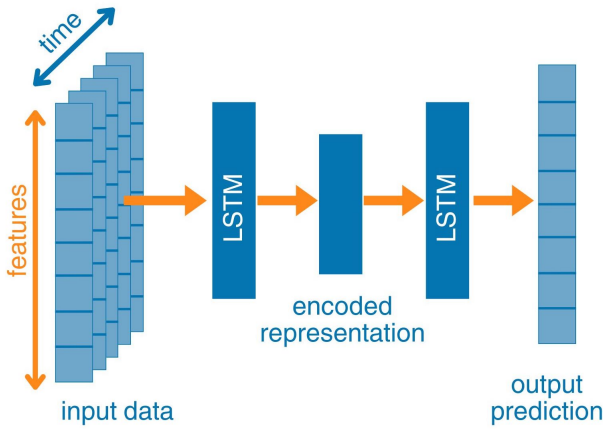

FIG. 2. Model Architecture. Structure of model with data flowing from input on left to output on right.

This showed a lot of promise, with nice ROC curves but with a higher false positive rate compared with our final model described below. By modifying the output of our model we were able to lower the false positive rate and improve the alignment between predicted and real values.

D. Final model: Predictive LSTM Autoencoder

Our predictive LSTM autoencoder combines the benefits of the autoencoder and LSTM network we previously explored. It takes in time-series data of 5 consecutive samples (where each sample is made up of 14 values corresponding to the 14 features used) such that the input spans 5 LBs (i.e. 5 minutes). As output, it generates a prediction of what the next sample should look like, in other words, predicting the values of each feature one minute into the future. The flow of data through this network is summarized in Figure 2. As we did with the earlier autoencoder model, we train on clean data such that the error we see between the prediction and real data serves as an indicator of how anomalous the input data was. We take the MSE between the predicted and real features and average over all of them. We set a classification threshold for MSE such that anything below that threshold is labeled clean and anything above is labeled anomalous.

In this way, we get the benefit of the LSTMs looking across time, giving context to what is happening with the data and how things are changing and the autoencoder shape bringing the data down to a lower dimensionality forces the model to extract the key components from the input data set, dropping the noise in the process. Another nice benefit of this model is

it is quite small, with only four layers and the LSTM layers of size nine. This will be especially important as the final version of this model will step up the number of input features significantly — but by compressing the data with our autoencoder we can limit the resource demand attached to that increase.

IV. MODEL TRAINING

For the sake of this discussion, we will consider the case of training the model across 10 runs (specifically runs 476875, 476428, 476276, 476785, 476718, 479279, 479269, 479239, 480188, and 480197) and testing the model with a single run (480219). This gives the model about 8000 data points for training and about 900 for testing.

As our cost function, we initially started with MSE which is the standard for these types of model applications (see equation 1).

However, this caused a large separation in the training and test loss since there were a few significant outliers in the training data that, when the difference between the real outliers and predicted normal data was squared contributed significantly to the loss. Since we do not want these various outliers that can be present in our training data to have a large impact on our training, we chose to use Huber loss instead. Huber loss acts similarly to MSE for small errors where it is quadratic but for larger errors like those we get with the outliers in our data, it acts linearly. We use TensorFlow's default Huber loss, so the delta as seen in equation 2 is set to one.

$$HL_\delta(y_i, y_i') = \begin{cases} \frac{1}{2}(y_i - y_i')^2 & \text{for } |y_i - y_i'| \leq \delta \\ \delta(|y_i - y_i'| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \quad (2)$$

We still observe a gap in the train and test loss as can be seen in Figure 3 but it is a gap of around 0.2 whereas using MSE results in completely separated loss curves with a gap of around 1.2. We expect such a gap to appear because our test data is an entirely new run and conditions tend to change in the period between runs.

At one point, we explored the possibility of cutting out outliers from our training data. However, we did not see a significant change in performance and we believe the use of Huber loss and the robust scaling described in the data selection and processing section should be sufficient to limit their impact on our model. Notably, if we plug the training data into our model, the predictions it generates align closely with the overall trends in the data but do not fluctuate with the various outliers (see Figure 4). This is both a result of the tools deployed and the actual autoencoder structure which allows us to eliminate noise in the data reconstruction process.

V. MODEL PERFORMANCE

In modifying out model — from large architecture decisions to hyperparameter tuning — we based our choices on a few key criteria. We examined how predictions aligned with the test sample, classification performance on a modified test set, and AD in real anomalous runs.

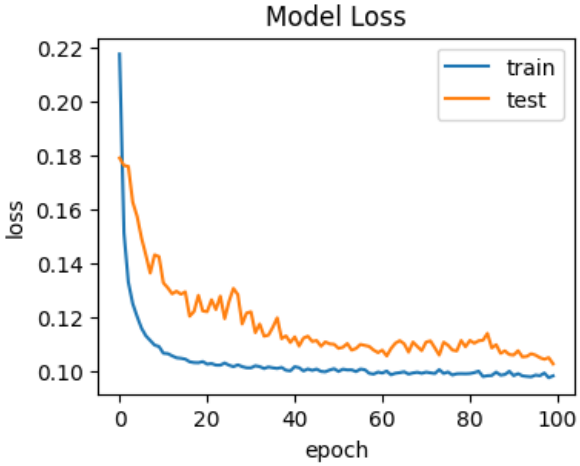

FIG. 3. Loss Curves. The loss calculated for train and test data sets over the course of 100 epochs.

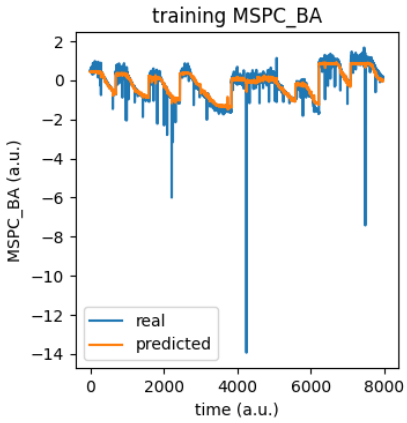

FIG. 4. Real vs predicted training MSPC_BA. The predicted values follow the shape of the training data closely but smooth out the noise and outliers.

A. Validation on the test sample

Our first indicator of model performance is considering how closely the model's predictions for the test data align with the data itself. Considering Figure 5, we observe that similar to the case with the training data, the prediction is mostly smooth across time and tends to match the trends of the data well as we would hope.

However, notably in the cases of MSPC_BA and L1_MU14FCH, the predicted results are shifted in the y-direction from the real data. Like with the difference in loss curves, we expect this is the result of conditions having changed in the test run so that the data does not align well with what the model has seen before. One option we

are considering to resolve this is to retrain the model at regular intervals in real-time so that after say 30 LB of the new run, the model would retrain and by incorporating the data of the present run into training, it would adapt to any changes in conditions. With this approach we would still expect to see the shift between real and predicted at the beginning of the run, but with each retraining, we would hope to see the two curves match better and better.

In these real vs predicted plots, we also tend to notice that our predicted value separates from the real in the low pile-up region of the run (i.e. at the tailend). In Figure 5, this is especially noticeable in the case of L1_jXE120. With our methods to squash the impact of outliers, we inadvertently squash the impact of the real low data we get in this plot since it is at an extreme. To address this, we recommend implementing partwise scaling procedures such that the portion of the run where pileup is flat is scaled separately from the region where pileup is decaying. Further optimization of the model with a specific focus on this region is also recommended. We tried to improve the performance at the end of the run by duplicating that section in the training data so that there was more of a balance between the amount of data in the flat pileup range and the amount in the low pileup range, but we did not see a significant impact on performance.

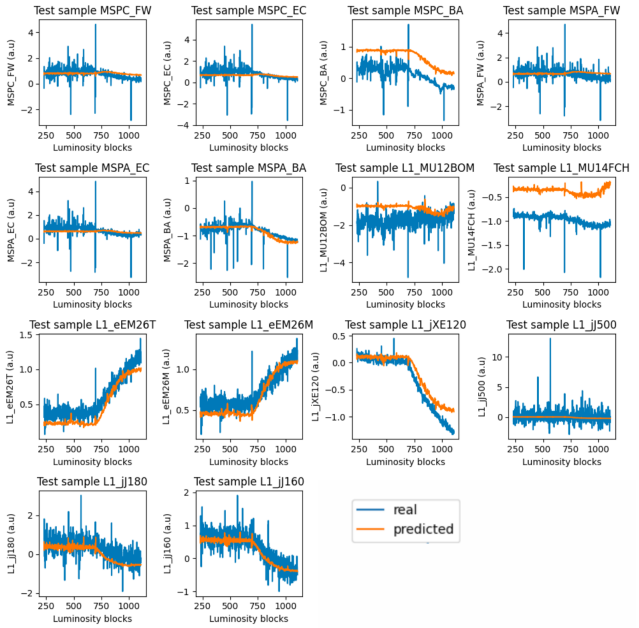

FIG. 5. Real vs predicted test data by feature. Overall trends in data are consistent between real and predicted but some shifts in data and poor performance in low pileup region.

B. Artificial anomalies

At the end of the day, what we really care about is how well our model catches anomalies so we modified our test data, introducing artificial anomalies, and assessed the output our model generated when we fed it this modified sample. One of these tests involved increasing a single feature by 5% over the first 30 LBs. We generated 14 different modified datasets each corresponding to a change in one of the 14 features. We then plotted the receiver operating characteristic (ROC) curves for each of those datasets as can be seen in Figure 6.

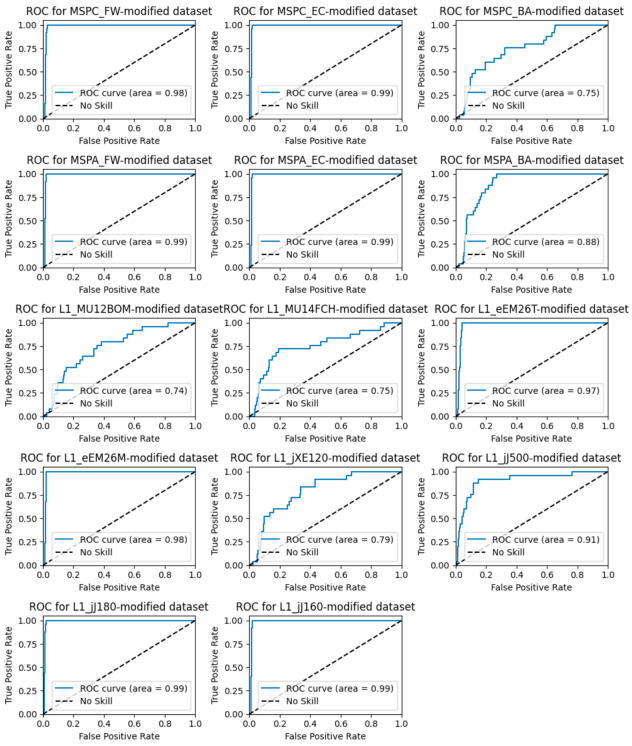

FIG. 6. ROC Curves. Each ROC curve corresponds to a test dataset where a single feature has been modified by 5% for the first 30 LBs. The true positive rate is indicative of how often we label a real anomaly anomalous and the false positive rate indicates how often we label a clean datapoint anomalous.

The resulting ROC curves indicate that our model is quite effective at detecting this type of anomaly for any feature but its success varies across the features. For the datasets with one of MSPC_BA, MSPA_BA, L1_MU12BOM, L1_MU14FCH, L1_jXE120, or L1_jJ500 modified, the area under the curve (AUC) is in the range of 0.74-0.91 while the rest have an almost perfect AUC of 0.98 or 0.99. These high values are promising because they indicate we can get a high true positive rate with a very low false positive rate. Keeping the false positive rate low is of critical importance because we expect much more clean data than anomalies so even a low false positive rate can have a big impact on the usefulness of the models when it is operating across the large amount of negative signals.

We have also explored how these ROC curve results change with different sizes of anomalies, affecting different numbers of LBs, and occurring at different points in the run (beginning, middle, or end). The performance looks similar to that shows in Figure 6 when only 15 LBs are modified with the other conditions held constant. Performance is slightly worse in the

middle of the run and rather improved at the end of the run (but significantly worse for the datasets with L1_MU14FCH or L1_jXE120 modified), likely because the model is not as effective at making predictions in that region so even with clean data we observe an uptick in MSE for several features

Moving left to right across a ROC curve, we follow the results as the threshold MSE dividing clean and anomalous data is lowered from infinity to zero. Figure 7 gives us a better visual of what is actually happening when such a threshold line is drawn. The distinction between anomalous and clean data points is much better defined in Figure 7a where the MSE for the single feature is plotted. However, the actual classification in our current model is based on the average MSE across all features as seen in Figure 7b. We can see that if we are to draw a line in 7b to include the modified initial data points, we will be forced to label several other data points as anomalous. This seems problematic until we remember that the test data we modified may have had some anomalies of its own even before we implemented our shift. For example, we notice a spike in MSE around LB 700. Looking at Figure 8, we see a large spike in several variables specifically at LB 700. So these high MSE values in Figure 7b do generally correspond to anomalies in the data — just not the anomalies we introduced and tagged.

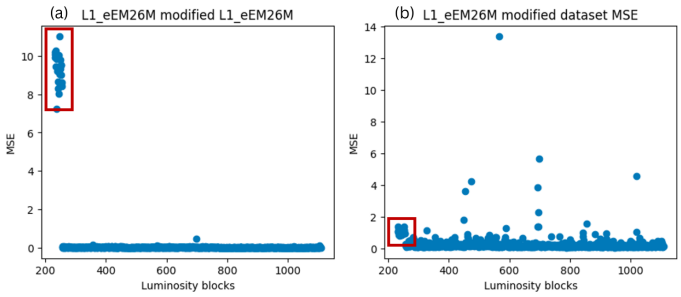

FIG. 7. MSE for anomalous test sample. a) MSE of L1_eEM26M, the single feature that was modified in this dataset. b) Average MSE across all features for this modified dataset.

To get a quantitative look at how our model performs with the optimal threshold on average MSE, we used confusion matrices. We take the optimal threshold from the corresponding ROC curve and see the actual numbers for true positives, false positives, true negatives, and false negatives. For example, in the case of the L1_eEM26M-modified anomalous dataset (which was one with an almost perfect ROC curve) we correctly identify 24 of the 25 anomalies and falsely classify 19 of the 859 clean data points as anomalies. This is a 96% true positive rate with a 2.2% false positive rate. However, it is worth noting that we only labeled the first 25 outputs as anomalous because the first 30 LBs were the only values we manually modified. But there are likely other anomalies present in the data that naturally came up in the course of the run and if our model is working well it should flag those as anomalous even though we have labeled them as clean. Based on our plots of the input data where the LBs our model flagged as anomalous are indicated in red (see Figure 8), we feel confident that the false positive rate is in fact lower than that which we report here.

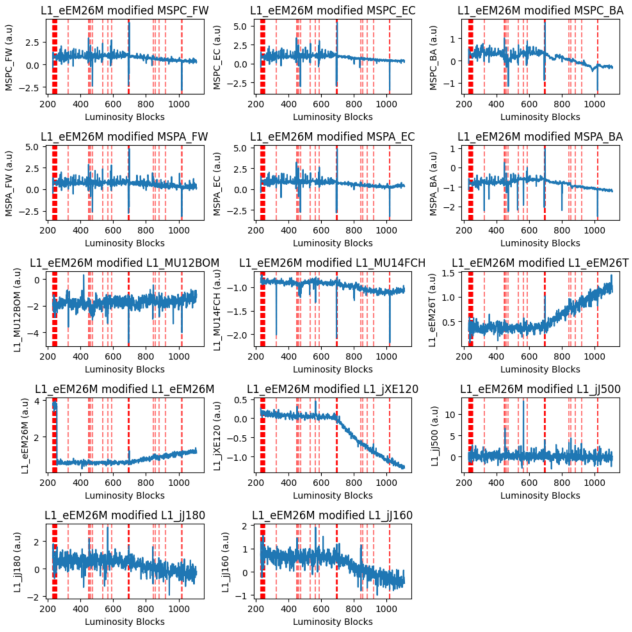

FIG. 8. Model-Identified Anomalies. Test anomalous dataset with modified L1_eEM26M plotted by feature in blue. Red lines mark LBs where model identified an anomaly.

Examining the data in Figure 8, we note that the red lines marking our model's anomalous labels frequently align with significant spikes in the data. It is reassuring that these anomalies align with visual changes in the data. It is however harder to see anomalies resulting from small fluctuations in the data extended over time. Future work should involve building more representative labeled datasets that incorporate different types of anomalies we may expect to show up so we can assess how well our model performance may differ across these cases.

We have done some initial exploration of basing our final labeling output on the by feature plots (as seen in Figure 7a) with custom thresholds set for each feature such that if a data point exceeds the threshold for any feature it is labeled as anomalous. With a first pass run with this strategy, again looking at the L1_eEM26M-modified anomalous dataset we improved our true positive detection from 24/25 to 25/25 but at a cost of an increased false positive rate from 19/859 to 48/859. However this still may be a better option to use for the final AD step when we consider the earlier discussion of how the false positive rate often picks up on real anomalies in what we are naively calling clean data and can be improved by tuning the threshold for each feature.

C. Muon end cap shutdown

While our tests on artificial anomalies give us a nice controlled scenario where we can tune the cases we study, we want to know whether our model can handle the real types of

anomalies that may pop up in detector operation. Towards this goal, we pulled a case study run in which a sector of the muon end cap (accounting for 1/8 of one side of the end-cap and forward detectors or 1/32 of the full muon trigger detector) was disabled on side C. For the first 100 LBs included in our sample, the detector operated as normal with the shutoff occurring around LB number 650. Examining our plots of real vs predicted data by feature, we see the large drop in real data for the MSPC_FW and MSPC_EC features while our predicted curve continues relatively flat with just a small dip where the drop occurs.

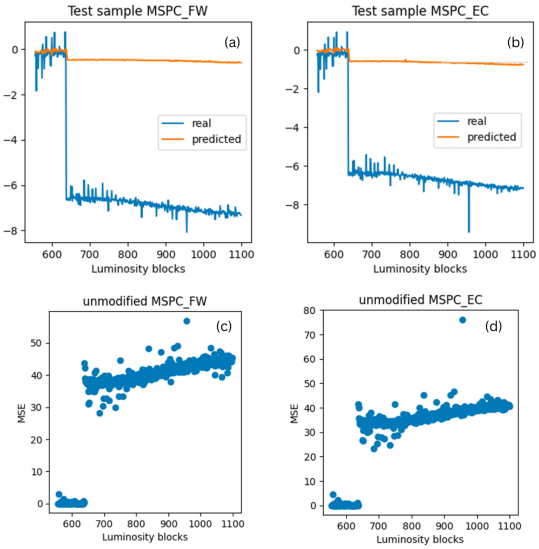

FIG. 9. Model Output with Disabled Muon End Cap Sector.

We can see similar output providing further evidence of the model's ability to detect such an anomaly by looking at the MSE plots over LBs for the same two features where we see the spike we expect. While these plots are what we expect to see with the current setup of our model, this is not how we would want the model to respond in real time in the control room because that would mean our model would continuously be warning about anomalies for the entire rest of the run. We just want to be warned of the anomaly when the initial change occurs, but then we would like the model to adapt to the new conditions so we can continue to use it to look for other anomalies that may pop up. For this reason, we recommend adjusting the model so that it retrains at regular interval throughout the course of the run, allowing it to take in the new data and make predictions that better reflect the current conditions of the detector.

D. Testing on Recent Run

To check how our model would work if it were to be used today, we fed in a test run and observed where anomalies were detected. In Figure 10 we plot the top 11 anomalies which are those flagged when the threshold is set to eight. The anomalies appear to align well with spikes in the rates but as mentioned above, further work is required to assess the performance here more quantitatively and to investigate performance across different types of anomalies.

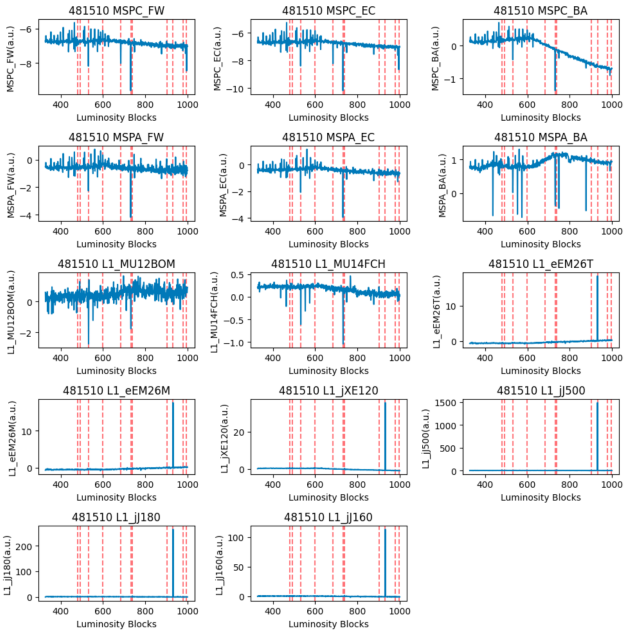

FIG. 10. Identified Anomalies in Recent Run. The top 11 anomalies identified by our model are plotted in red against the feature data for run number 481510.

VI. CONCLUSION

The predictive LSTM autoencoder proposed here shows promise for online AD in the ATLAS control room. After iterating through several different machine learning models and drawing inspiration from recent AD approaches, we have optimized the procedure to take advantage of the benefits of both autoencoders and LSTMs. We observed a high AD rate of 96% and associated low false positive rate of around 2% with errors of 5% applied across 30 minutes of data. Such good results with such a simple model make us optimistic that automated AD will be effective in the ATLAS control room and that these tools can be made operational on a relatively short time scale.

VII. FUTURE RESEARCH

We are optimistic that AD models can accomplish our goals of improving control room operations, but before our model can be implemented, there are a few more steps to undertake. First and foremost, to shift our optimism to confidence in the approach we have followed, the performance of our model should be bench-marked against non-ML models for example using a simple algorithm of the absolute difference between consecutive data points which we would expect to be

good at finding point fluctuations in the data but not very effective at picking up subtle trends. As it stands, we don't have a good sense of how our model performs for different kinds of anomalies and these point anomalies are the easiest to see so comparing with other methods can give us more insight and we can create a test set with a wider range of anomaly classes to assess performance.

One area for improvement is in optimizing the model for the low pileup region. To do so, one should explore implementing piece-wise normalization such that the region corresponding to constant pileup is scaled separately from the region with decaying pileup. This should help as the low pileup region will not stick out so far into the outlier region where it is now and is therefore somewhat ignored by the model. Another option that may help with this problem as well as improve performance overall is to separate the constant and decaying pileup regions altogether and train two models separately each with just one type of data. In the development process we at times chose to use just the flat pileup region, but chose to use the full data range for this initial model so as to best establish the usefulness of such an approach in the general case.

A mandatory step before this can be used in the control room is scaling up to a more extensive feature set. This includes incorporating some of the features we considered initially but decided to leave out such as empty rates where rates are recorded when there is an empty bunch crossing. But beyond incorporating more trigger rates, we will also want to consider expanding to include other types of data which may include graph or image-based data.

In order to make the model more adaptable to changes in conditions both between and within runs, we also recommend implementing continuous learning in such a way that the model retrains at regular periods incorporating the new data that has been collected. This way, the model can make decisions incorporating the most up-to-date data. So, for example, in cases where a section of the detector is shut off during a run, upon retraining it can learn to make predictions with this new setup and stop warning about anomalies we are already aware of.

In addition to this retraining, incorporating reinforcement learning could be beneficial[9]. For a certain range of MSE where the model is not especially confident if the data indicates an anomaly or not, it could flag a shifter and they could provide a label for the data which would allow the model to improve its future decisions.

Finally, there is plenty of room for exploration in terms of the actual architecture of the model. We have built a rather simple network here as a starting point, but upgrades could resolve some of the issues we have seen with variation in performance depending on the feature the anomaly occurred in and poor performance in certain regions of the sample. For example, transformers could be used instead of LSTM and a natural next step is to implement this as a variational autoencoder which in fact, we already have a draft of that requires some optimization work.

REFERENCES

[1] A. A. Pol, V. Azzolini, G. Cerminara, F. De Guio, G. Franzoni, M. Pierini, F. Široký, and J.-R. Vlimant, "Anomaly detection using deep autoencoders for the assessment of the quality of the data acquired by the cms experiment," 23rd International Conference on Computing in High Energy and Nuclear Physics, CHEP 2018 (2018).

[2] A. A. Pol, G. Cerminara, C. Germain, M. Pierini, and A. Seth, "Detector monitoring with artificial neural networks at the cms experiment at the cern large hadron collider," Comput Softw Big Sci 3 (2019).

[3] A. A. Pol, V. Berger, G. Cerminara, C. Germain, and M. Pierini, "Trigger rate anomaly detection with conditional variational autoencoders at the cms experiment," Machine Learning and the Physical Sciences Workshop at the 33rd Conference on Neural Information Processing Systems (NeurIPS) (2019).

[4] J. Strandberg, "Data preparation in atlas,".

[5] M. Stockton, "The atlas level-1 central trigger," Journal of Physics: Conference Series (2011).

[6] Analysis Software Group, "Pileup analysis configuration,".

[7] SciKit-Learn, "Robustscaler,".

[8] J. Starmer, "Long short-term memory (lstm), clearly explained,".

[9] O. J. Parra, J. G. Pardiñas, L. D. P. Pérez, M. Janisch, S. Klaver, T. Lehéricy, and N. Serra, "Human-in-the-loop reinforcement learning for data quality monitoring in particle physics experiments," CHIPP 2024 Annual meeting (2024).