EMORY | COLLEGE OF ARTS AND SCIENCES

Emory University (2023)

RESEARCH ARTICLE

Assessing Predictive Maintenance Models: A Comparative Examination of Leading Model Architectures in Machine Learning and Deep Learning

Jonathan Zhang,* Sherry Rui,* and Lucas Goldfein*

College of Arts and Sciences, Emory University, Atlanta, GA, United States
*Corresponding authors. Emails: jonathan.zhang2@emory.edu; sherry.rui@emory.edu; lucas.goldfein@emory.edu

Abstract

Predictive maintenance (PdM) is the practice of leveraging real-time data to identify operational anomalies and anticipate equipment failures *before* they occur. Across various industries, PdM has recently emerged as a crucial endeavor that not only mitigates costly downtime but also extends machinery lifespan, optimizes resource allocation, and enhances operational efficiency. This paper delves into the application of deep learning within the PdM realm and examines the tradeoff these architectures offer over leading machine learning methods for remaining useful life estimation and anomaly detection. This comparative analysis considers deep time series forecasting models, including LSTMs and attention mechanisms, with traditional and ensemble techniques such as linear regression, random forest, and gradient boosting. In both regression and classification tasks, LSTM architectures outperformed machine learning models and excelled in autonomously extracting complex patterns as evaluated by R^2, *MSE*, AUROC, and precision-recall scores.

Keywords: predictive maintenance, long short-term memory, attention, neural networks

1. INTRODUCTION

Traditionally, maintenance schedules are determined by estimated machine lifetimes, leading to either premature maintenance or unexpected breakdowns. The goal is to develop machine learning and deep learning models that can predict, with high accuracy, how much remaining useful life an equipment has and whether it is likely to fail within a specific time frame (e.g., within the next week or two). This prediction is based on sensor data, focusing on patterns that indicate increasing wear and tear. Successful predictions would allow industries to shift from traditional, static maintenance schedules to dynamic, data-driven ones, greatly optimizing maintenance resources and minimizing unexpected downtimes.

2. BACKGROUND

Among the myriad of machine learning models employed in predictive maintenance, gradient boosting trees and random forests have gained prominence. In the regression realm, a recent conference paper investigated algorithms like random forest, linear regression, XgBoost, and K-nearest neighbors and found that XgBoost was the best performing one with an *RMSE* of 27.7277 (Ajay, Krishnna, and Jhajharia 2023). With respect to classification, previous work on tree-based models also highlight gradient boosting trees for predicting maintenance necessity, achieving an accuracy of 0.862 and an F-score of 0.860 on heldout test dataset. In the same paper, the authors also underscore

random forest's performance in classifying maintenance type, securing an accuracy of 0.70. Notably, the paper acknowledges that future directions include utilizing deep neural networks to develop a unified model for the prediction of other maintenance tasks (Allah Bukhsh et al. 2019).

Other works have focused on deep learning algorithms, forgoing standard machine learning methods. In a review paper of different deep learning models applied to predictive maintenance, various deep learning models resulted in a wide range of *RMSE* loss values (Serradilla et al. 2022). Among the most prominent models, all of them incorporate LSTMs. Namely, LSTM + FFNN achieved an *RMSE* of 16.1; RBM + LSTM (a semi supervised model) resulted in the lowest *RMSE* of 12.6; and online LSTM led to an *RMSE* of 14.0. In this study, we want to conduct a comparative study that would allow us to consolidate knowledge from machine learning and deep learning models and potentially discover more generalizable models through this comprehensive approach.

3. METHODOLOGY

Three machine learning models and two varieties of sequence-based deep learning algorithms were evaluated. The machine learning techniques also act as comparison baselines with respect to the deep learning approaches.

3.1 Linear Regression

Linear regression is a fundamental method used in predictive modeling. It establishes a relationship between a dependent variable and its independent variables using a linear equation. This approach is beneficial for its simplicity, interpretability, and computational efficiency.

3.2 Random Forest

The random forest regressor is an ensemble machine-learning technique that leverages bootstrapped samples and multiple decision trees to improve overall predictive accuracy and model generalizability. As a state-of-the-art method that decreases model bias without drastically increasing variance, random forests have been used prior in the PdM realm to classify maintenance type and trigger status. In this paper, the regression variant is used to predict remaining useful life.

3.3 Gradient Boosting Tree

Gradient boosting trees are another state-of-the-art ensemble method that sequentially builds a series of weak decision tree learners to correct the errors of the preceding models (albeit more prone to overfitting than random forests). Previous work has shown gradient boosting trees' effectiveness in classifying maintenance necessity, and this paper presents the regression variant from XGBoost to handle potential data nonlinearity and high dimensionality.

3.4 Long Short-Term Memory

Long short-term memory is a type of recurrent neural network (RNN) architecture. The term "recurrent" in RNN refers to the recurrence of information over time through the hidden state, allowing the network to maintain a form of memory. At every time step, the model *unrolls* itself, and the hidden state, modeled in the recurrence formula, can be influenced by the input at that time step and the previous hidden state.

$$\mathbf{h}_t = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

Using this recurrent connection, RNNs capture sequential dependencies in data. To train vanilla RNNs, gradients of the loss function must be backpropagated through time. For simplicity, consider an RNN without bias parameters, whose activation function in the hidden layer uses the identity

function. For any time step $1 \leq t \leq T$, the gradient computation involves the following formula where the $\mathbf{W}$'s are the model parameters (weights in the model's recurrence formulas).

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^{T} (\mathbf{W}_{\text{hh}}^{\top})^{T-i} \mathbf{W}_{\text{qh}}^{\top} \frac{\partial L}{\partial \mathbf{o}_{T+t-i}}$$

The challenge lies in that $\mathbf{W}_{\text{hh}}$ can have very large powers in terms of total time step T. Mathematically, eigenvalues in $\mathbf{W}_{\text{hh}}$ that are smaller than 1 vanish, and eigenvalues larger than 1 diverge. As such, backpropagation through time is numerically unstable given a vanilla RNN. Often, there are issues of vanishing gradients.

LSTMs were designed to address the vanishing gradient problem through their specialized architecture: a memory cell, input gate, forget gate, and output gate.

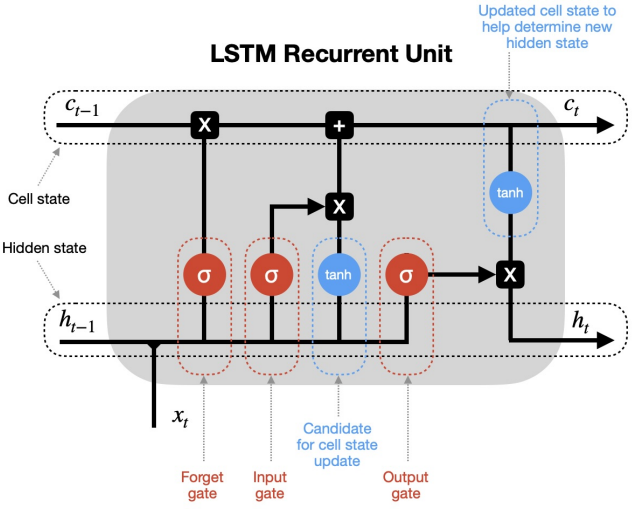


Figure 1. LSTM unit architecture

The key concept is the memory cell, which is a long-term storage unit that retains information for long durations without suffering from vanishing gradients. At the same time, the three essential gates regulate the flow of information. The input gate selectively determines which information from the current time step is stored in the memory cell. The forget gate decides what information is discarded from the memory cell from previous time steps. And the output gate controls the flow of relevant information from the memory cell to the next hidden state. The sigmoid activation functions play a vital role in these gates as it squashes input values between 0 and 1.

Given that LSTMs are particularly well-suited to analyzing sequential time series data, they possess high compatibility for handling the NASA Turbofan Jet Engine dataset. The selective information update, adaptability to varying time intervals, and automatic feature learning make LSTMs adept at learning complex patterns inherent in predictive maintenance data.

3.5 Attention Mechanism

Attention mechanisms are an improvement on RNN architectures aimed to address their weakness where sequence data is successively processed only one at a time, causing the model to favor more

recent input at the end of a sequence. Attention mechanisms enable a neural network to selectively focus on different parts of the input when making predictions, and its output is computed as follows:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are query, key, and value representations obtained via linear projections of the input sequences $\mathbf{X}_q$, $\mathbf{X}_k$, and $\mathbf{X}_v$ using learnable weights.

Consider the query as the part of the input sequence that is seeking information and the key as the elements in the input sequence against which the query is compared to determine relevance. Attention mechanisms compute attention scores as scaled dot products between $\mathbf{Q}$ and $\mathbf{K}$ that measure the similarity or relationship between different elements. Higher dot products indicate higher relevance, allowing the model to focus on specific parts of the input sequence that are more relevant to the query.

Suppose $\mathbf{V}$ is the information associated with each element in the input sequence. Attention mechanisms output a weighted sum of $\mathbf{V}$ where the weights are given by the attention scores after the softmax function (attention weights). The weighted sum allows attention layers to selectively emphasize certain elements and de-emphasize others, providing a flexible way to represent and aggregate information.

Given that the predictive maintenance task does not depend on information from multiple sources or sequences, self-attention is proposed to capture temporal dependencies and understand how past observations within the same sequence impact future outcomes. Mathematically, $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are learned from the same input sequence $\mathbf{X}$:

$$Q = \mathbf{W}_Q \cdot \mathbf{X} \qquad K = \mathbf{W}_K \cdot \mathbf{X} \qquad V = \mathbf{W}_V \cdot \mathbf{X}$$

Furthermore, multi-head attention is used to overcome any limitations of single-head attention by using multiple sets of query, key, and value projections:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\mathbf{head}_1, \ldots, \mathbf{head}_h)\mathbf{W}_o$$

where $\mathbf{head}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i)$.

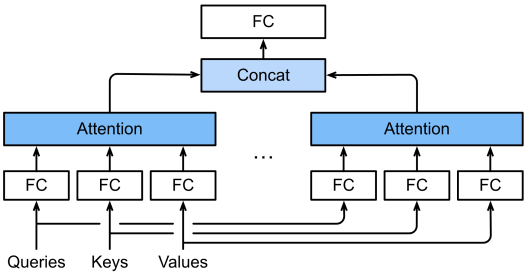


Figure 2. Multi-head attention layer

For each head, the attention mechanism calculates attention scores independently. As such, the model can focus on different parts of the input sequence for different heads, capturing diverse patterns.

4. DATASET DESCRIPTION

The dataset used in this study is the NASA CMAPSS Jet Engine dataset[1] consisting of multiple multivariate time series data from engine degradation simulation. It is separated into four sub-datasets, each of which is already divided into training and test subsets. This study was carried out using dataset FD001.

In the dataset, all the time series came from a fleet of 100 engines of the same type. All the engines are operating normally at the start of each time series but have different degrees of initial wear and manufacturing variation which is unknown. Each engine develops a fault at some point during the series. In the training set of 20,631 records, the fault grows in magnitude until system failure. In the test set of 13,096 records, the time series ends sometime prior to system failure.

The features in the dataset include unit number, timestamp (in cycles), three operational settings that are known to have a substantial effect on engine performance, and 21 sensor measurements. All of the three operational settings and 21 sensor measurements are metrical (i.e., no nominal variables). The label in this dataset is the remaining useful life (RUL) values. These values are not included in the training data but are given as an additional vector corresponding to the last time series for each engine in the test data.

5. DATA PROCESSING

Overall, the dataset did not contain any missing values. All other data preprocessing steps were conducted based on the model of interest.

5.1 Regression: Linear Regression, Ensembles, LSTM & Attention

5.1.1 Exploratory Data Analysis:

1. Calculate RUL in training data using the difference in total number of cycles an engine was recorded in the dataset and the current cycle it is on.
2. Standard scaling training data features, and transform testing data features.
3. Drop the features that resulted in 0 mean and 0 standard deviation: setting3, sensor1, sensor5, sensor10, sensor16, sensor18, and sensor19. Pearson correlation of the remaining features is shown in Figure 3.

Code found in `reg-lstm-attn/RUL Data Processing.ipynb`.

5.1.2 Data Labeling for Linear Regression

For every record in test data, the difference between the maximum number of cycles recorded and the current cycle for that engine is computed. The record labels are generated from the sum of these differences and the true RULs given that engine.

Code found in `LinearReg/Linear Regression Code.ipynb`.

5.1.3 Data Labeling for Ensemble Methods

Each record in test data is labeled using the true RULs for each engine and adjusted for the prior time series by the difference in cycles. The ground truth vectors are generated using a limit of 125 cycles on the true RULs[2]. In other words, any RUL label higher than 125 remaining cycles is reduced to 125.

1. https://data.nasa.gov/Aerospace/CMAPSS-Jet-Engine-Simulated-Data/ff5v-kuh6/about_data
2. The decision to perform a cutoff at 125 is motivated by initial LSTM model training as it would struggle with inputs where the labels drastically exceeded 125 in earlier epochs (before it overfit). Ensemble methods saw similar struggles. More information in Appendix 1

Code found in `reg-lstm-attn/RUL Data Processing.ipynb`.

5.1.4 *Sequence Generation for LSTM & Attention*

The time series data is divided using a moving (overlapping) window of size 30 cycles and a shift of 1 for each engine. For each window per engine, the corresponding RUL label is taken from the next immediate time series record. The 125-cycle cutoff was used here as well. The same windowing approach is applied to test data, with an additional parameter to specify the generation of a maximum of the last five windows per engine. In other words, the final RUL prediction for each engine will be based on the last five sequences recorded (using the average prediction). For 100 engines, this resulted in 17,631 training sequences and 497 test sequences.

Code found in `reg-lstm-attn/RUL Data Processing.ipynb`.

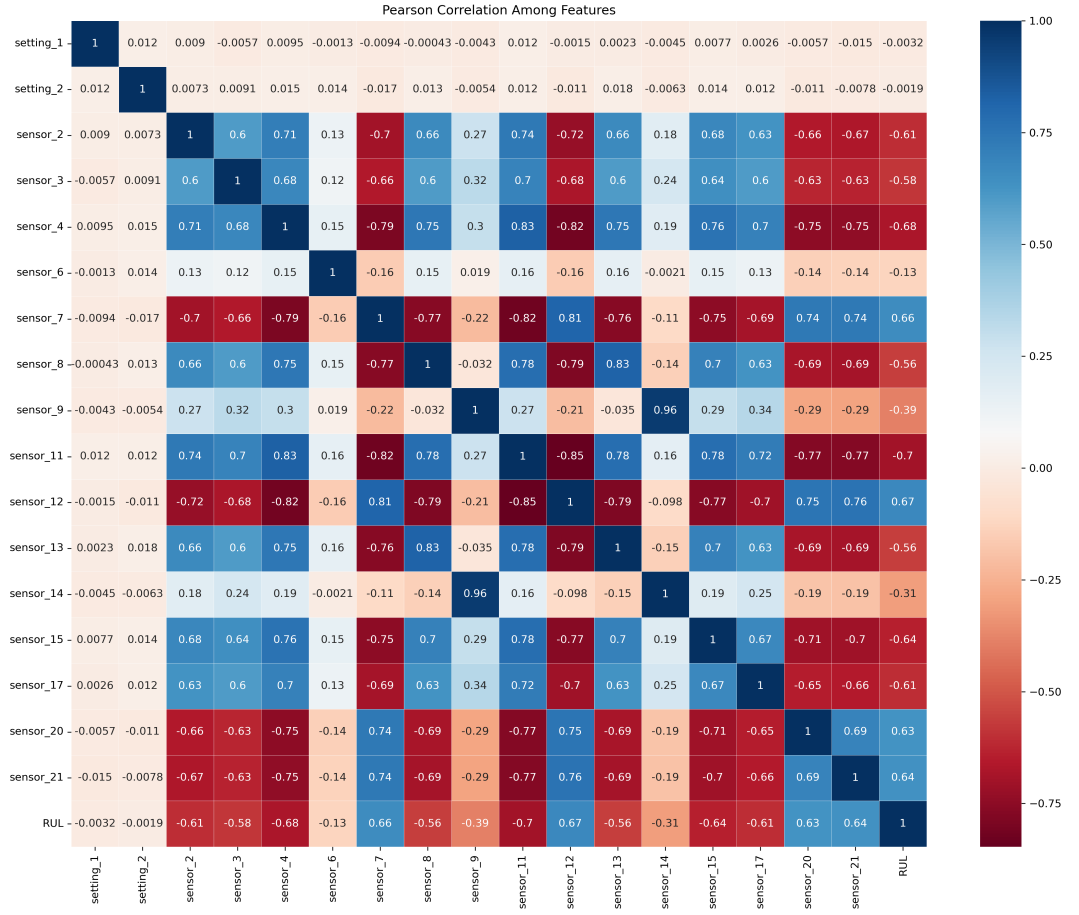


Figure 3. Pearson Correlation Matrix

5.2 *Classification: LSTM*

5.2.1 *Exploratory Data Analysis*

1. Data Transformation: normalized cycles and features.

2. Normalization and Scaling: `MinMaxScaler` for feature normalization, crucial step as LSTMs are sensitive to the scale of input data.
3. RUL Calculation: Computed RUL for each engine; merged with original data.

5.2.2 Label Creation

The classification labels allow the model to learn from the temporal progression towards failure by setting thresholds(w_0 = 15, w_1 = 30) to define the time windows.

1. Binary Labels (`label1`):
   - Label 1 (Positive Class): If an engine's RUL is 15 cycles or fewer ($\leq w_0$). The label '1' indicates that the engine is at a high risk of failing soon and requires immediate attention.
   - Label 0 (Negative Class): If an engine's RUL is more than 15 cycles ($> w_0$). The label '0' suggests that the engine is not at immediate risk of failure.
2. Multiclass Labels (`label2`): Distinguishing between more imminent failure (within 15 cycles) and less imminent failure.
   - Label 0: Engines with an RUL greater than w_1 (not expected to fail within the near future).
   - Label 1: Engines with an RUL between w_0 and w_1 (moderate risk of failure).
   - Label 2: Engines with an RUL of w_0 or less (high risk of failure).

5.2.3 Sequence Selection

The time series data is converted into sequences with a length of 50 cycles, which balances the amount of temporal information the model sees against the computational cost and memory constraints.

- Sequence Generation: A generator function is used to slide a window across the sensor data to create sequences, ensuring that each sequence is a continuous block of fixed number of cycles.
- Sequence Creation: Reshaped data into LSTM-compatible input format (samples, time steps, features), creating overlapping windows (sequences) of a fixed number of cycles (`sequence_length`).

5.2.4 Final Preparations

- Test Data: The same preprocessing steps are applied to the test set to maintain consistency.
- Data Split: The dataset is split into 80% training and 20% testing sets, with `random_state=42` for randomness in splitting.

Code found in `lstm/lstm_data_processing.py`.

6. RESULTS

6.1 Regression Performance Evaluation

Evaluation Metrics

1. *MSE*: $\frac{1}{N}\|\mathbf{y} - \hat{f}(\mathbf{X})\|_2^2$
   All regression models were trained using the mean squared error loss. Penalizes large errors more heavily than small errors.
2. *RMSE*: $\sqrt{\frac{1}{N}\|\mathbf{y} - \hat{f}(\mathbf{X})\|_2^2}$
   Root of MSE. This is used to evaluate model performance on test data.
3. *MAE*: $\frac{1}{N}\|\mathbf{y} - \hat{f}(\mathbf{X})\|_1$
   Also used to evaluate model performance on test data.

4. R^2: $1 - \frac{\|\mathbf{y} - \hat{f}(\mathbf{X})\|_2^2}{\|\mathbf{y} - \bar{y}\|_2^2}$
   Determines the proportion of variance in the true RULs explained by the model predictions.

6.1.1 *Linear Regression*

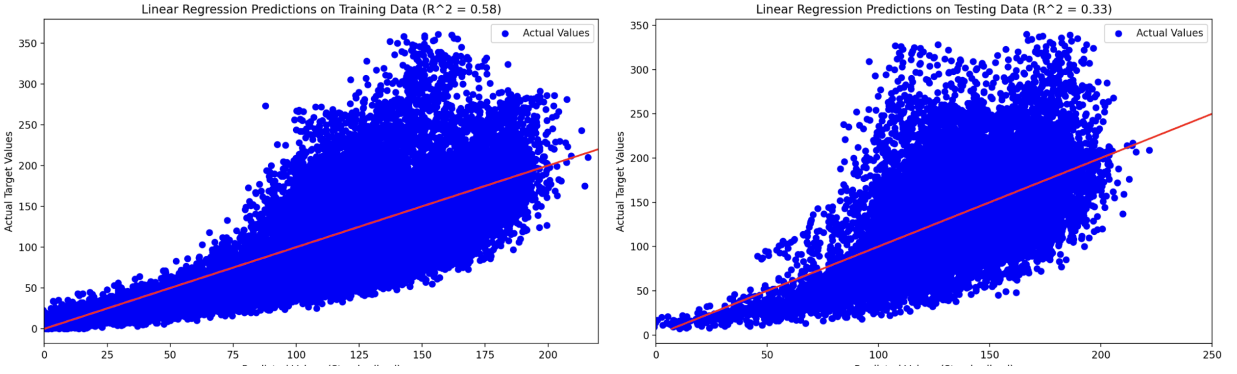


Figure 4. Naive linear regression performance

The naive linear regression model, using all the preprocessed features, performed acceptably on training data, explaining the majority of the variance. However, it does not perform very well on the testing data. It has a noticeably low R^2 value, at around 0.33, and a high mean squared error, at 2342.5101. As this is a potential sign of overfitting, a Lasso-regularized model was tested.

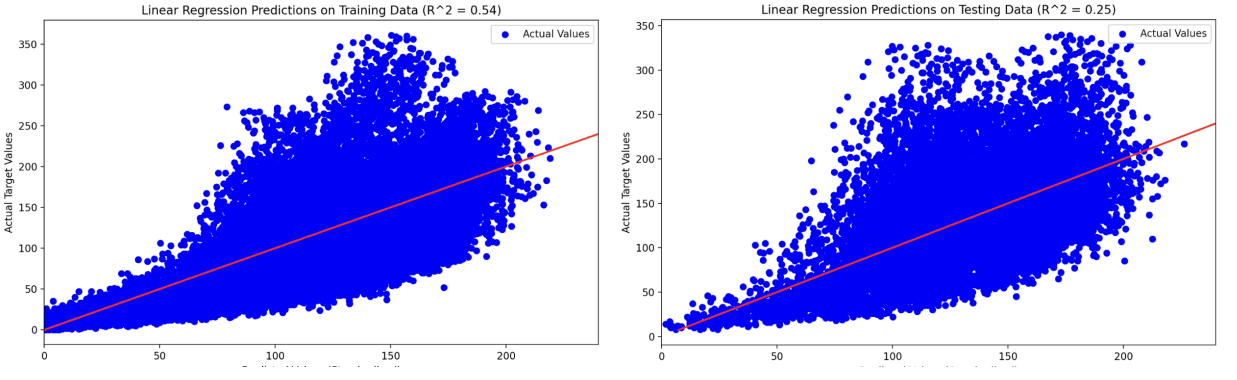


Figure 5. Lasso regression performance

With Lasso, the model reduces to six parameters, including the bias term, while sacrificing a small amount of explained variance in training:

$$\hat{f}(\mathbf{x}_i) = 107.8 - 13.4(sensor4) + 7.4(sensor7) - 16.4(sensor11) + 8.2(sensor12) - 9.5(sensor15)$$

However, with the Lasso regularization, the extrapolation to the testing data does not improve and slightly worsens. The R^2 value is now at 0.25 and the MSE 2619.7135.

Table 1. Linear regression test data metrics

	MSE	*RMSE*	*MAE*	R^2
Naive Linear Regression	2342.5101	48.3995	–	0.33
Lasso Regression	2619.7135	51.1831	–	0.25

Both linear models can fit the training data well, but they both fail to predict new data reliably. The non-negligible residuals of the linear models suggest model misspecification and data nonlinearity. More complex models are needed to capture the data accurately.

Code and results found in `LinearReg/Linear Regression Code.ipynb`.

6.1.2 Random Forest (RF) Regressor

Hyperparameter Tuning:

Grid search was applied using out-of-bag error, and the best set of hyperparameters found was {'n_estimators': 140, 'max_features': 6, 'max_depth': 9, 'min_samples_leaf': 8}. See figures 6 and 7.

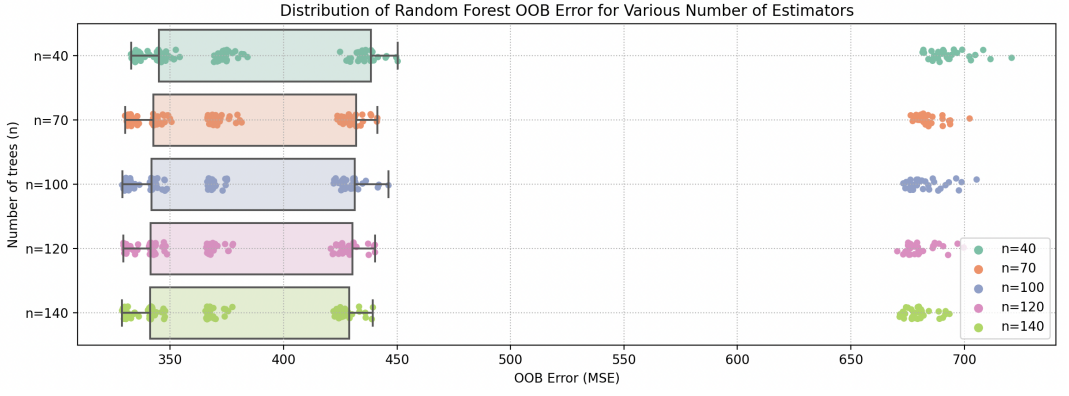


Figure 6. Performance distribution for random forest regressors with 40, 70, 100, 120, and 140 trees

In figure 6, out of five options for the number of trees, the optimal number is 140, producing slightly better performance than when the number of trees is 120.

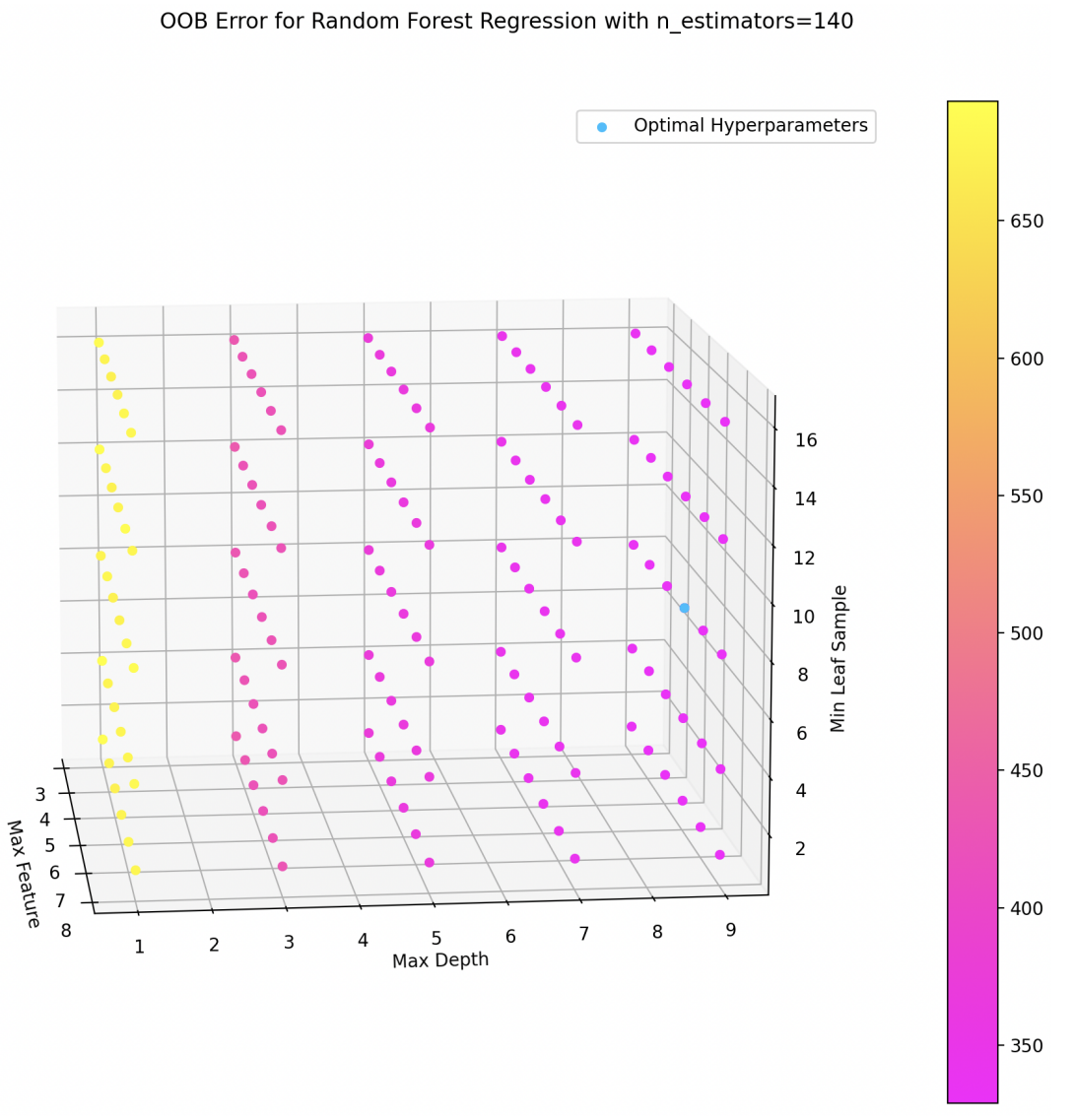


Figure 7. Tuning different tree-related parameters (fixing the number of trees in each random forest to 140)

Across all the regressors trained during tuning with 140 trees, the best-performing model, the blue point in figure 7 uses a maximum of 6 features, a maximum depth of 9, and a minimum of 8 leaf samples to train each tree. Relative to the other two hyperparameters in figure 7, the max depth of each tree affects the model performance the most. When max depth is 1, setting max feature to 1 appears to increase out-of-bag error slightly. Otherwise, max feature and minimum leaf sample have negligible effect on the model performances.

Best Random Forest Regressor Test Performance:

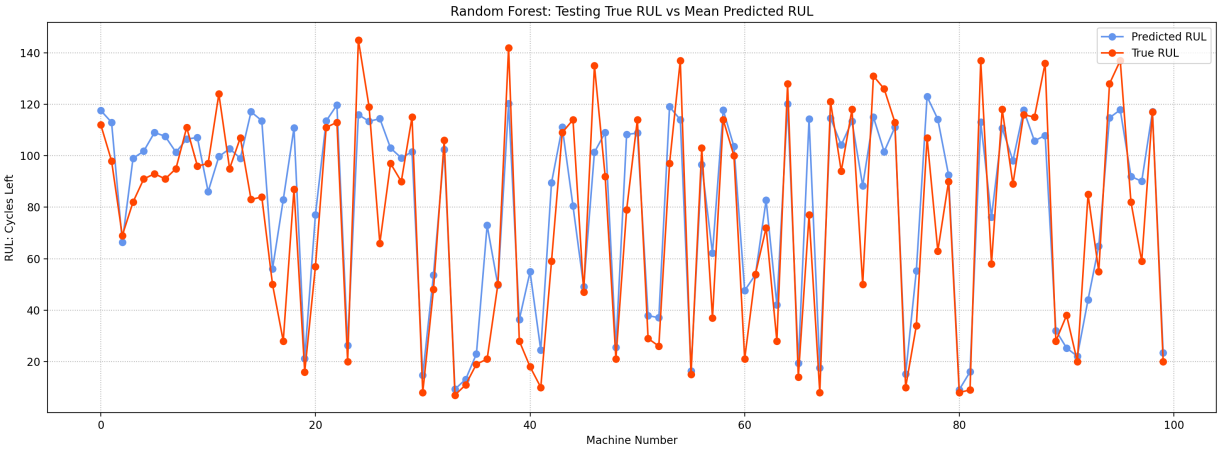


Figure 8. Random forest regressor test RUL predictions for 100 engines

Table 2. Random forest regressor test data metrics

	MSE	*RMSE*	*MAE*	R^2
Mean Prediction	371.4459	19.2729	14.4061	0.7849
Last Prediction	316.3018	17.7849	12.9322	0.8168

According to figure 8 as well as the R^2 values, random forest performs well in predicting RUL even without advanced feature engineering techniques. Interestingly, the performance increases when the random forest regressor makes its prediction only on the last time series record for each engine instead of averaging over the predictions for the last five records. The R^2 value increased by 3.19%.

Code and results found in `reg-lstm-attn/Ensemble regression.ipynb`.

6.1.3 Gradient Boosting Tree (GBT) Regressor

Hyperparameter Tuning:

- Given the number of hyperparameters for gradient boosting trees, the tuning method first searched for the optimal tree-related parameters for a fixed, reasonable learning rate of 0.2 using grid search with 5-fold cross-validation (with mean squared error loss).
- The optimal set of tree-related hyperparameters for GBT regressor was {'max_depth': 3, 'min_child_weight': 50, 'n_estimators': 140} (minimum child weight is analogous to minimum leaf samples in random forests).
- Then, 5-fold CV was used again to tune the learning rate using these optimal hyperparameters to the optimal value 0.1.

See figures 9 and 10.

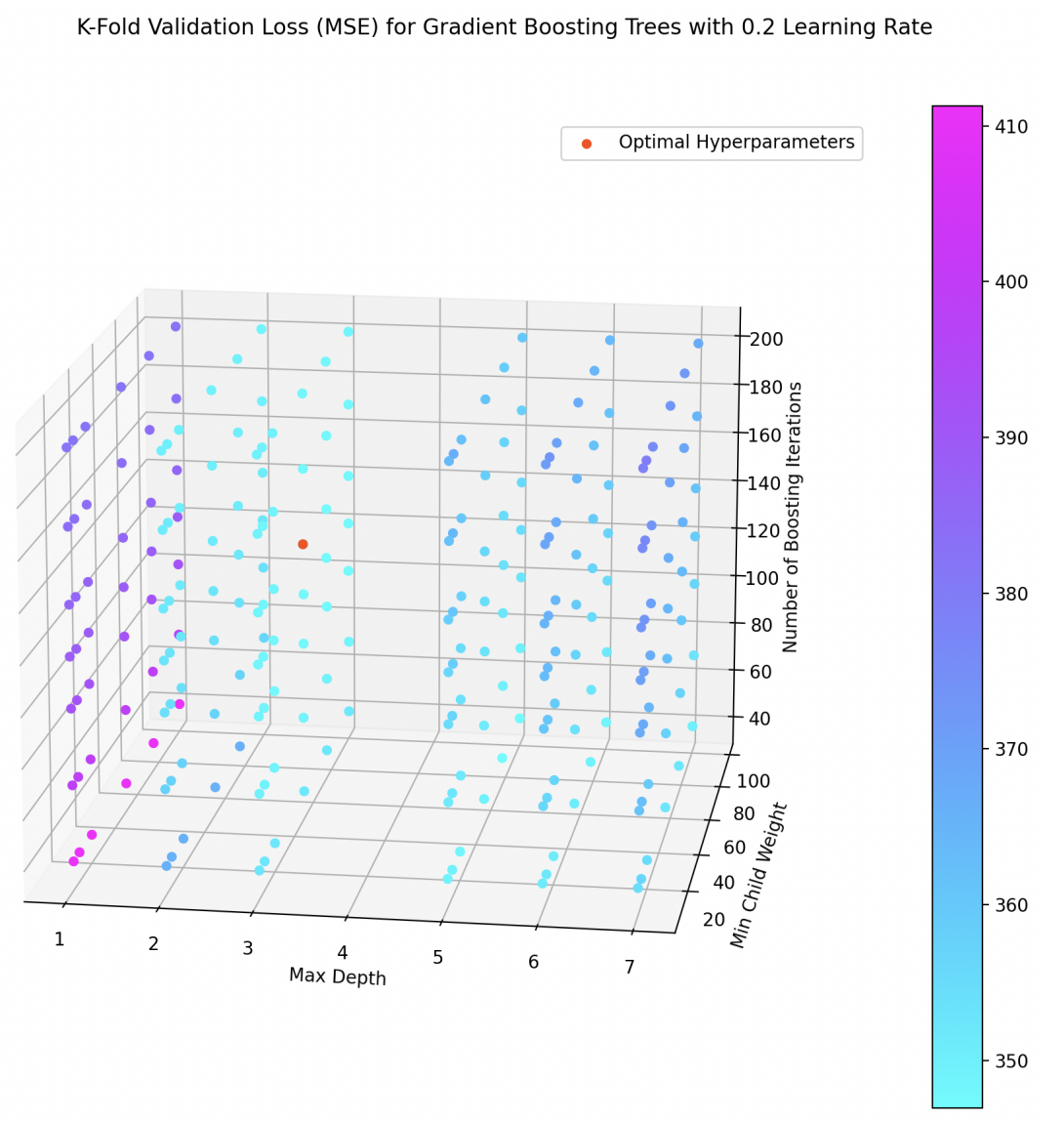


Figure 9. Tuning different tree-related parameters for a fixed learning rate of 0.2

Among 252 model candidates at a learning rate of 0.2, the best-performing GBT regressor, the red point in figure 9 uses a maximum depth of 3, a minimum child weight of 50, and 140 boosting iterations to train. Notice that the maximum depth emerges again as the hyperparameter that affects the model performance the most. However, for gradient boosting trees, a more moderate maximum depth of 3 is optimal (instead of 9 for random forests). This is consistent with the idea that gradient boosting utilizes many "weak" learners to successively train its model.

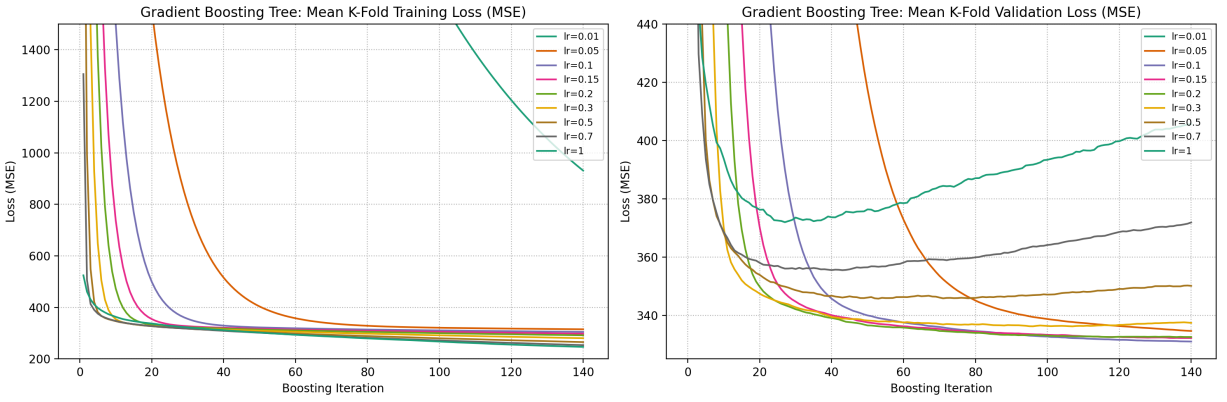


Figure 10. Learning rate comparison for gradient boosting trees using optimal tree-related hyperparameters

In the validation loss plot in figure 10, the model that used a learning rate of 0.1 achieved the lowest validation loss after 140 boosting iterations. Following it closely are models with 0.15 and 0.2 learning rates. Higher learning rates (e.g., 0.7 & 1) exhibit clear signs of overfitting when validation loss climbs after reaching a local minimum.

Best Gradient Boosting Tree Regressor Test Performance:

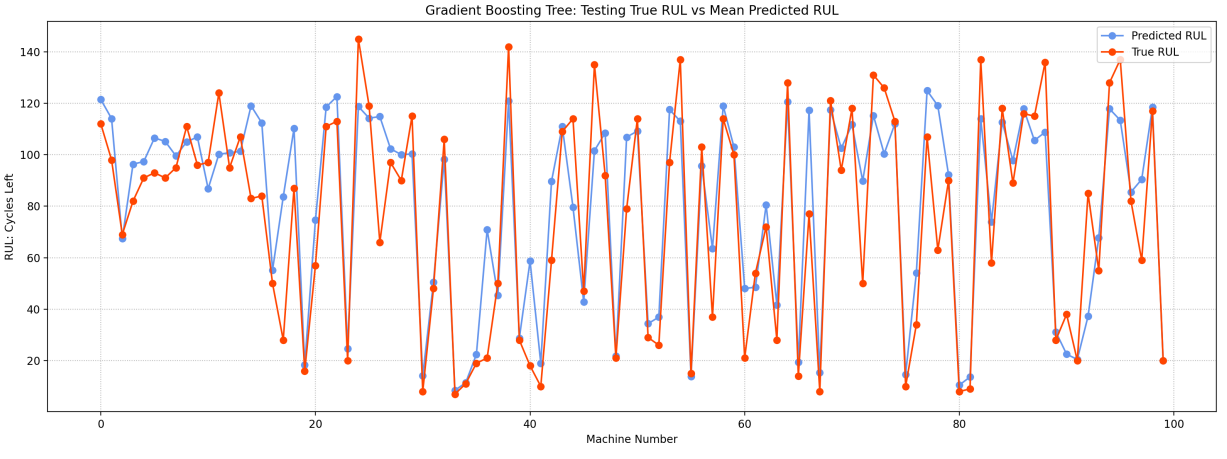


Figure 11. Gradient boosting tree regressor test RUL predictions for 100 engines

Table 3. Gradient boosting tree regressor test data metrics

	MSE	*RMSE*	*MAE*	R^2
Mean Prediction	380.7650	19.5132	14.1366	0.7795
Last Prediction	322.6763	17.9632	13.1359	0.8131

Although slightly lower, the GBT regressor test metrics trail that of the random forest regressor very closely. The predictions are almost indistinguishable between figures 8 and 11. Once again, the performance increases when the GBT regressor makes its prediction only on the last time series

record for each engine instead of averaging its predictions for the last five records. In general, both ensemble techniques have similar performances. Neither outperforms by a significant amount given the *RMSE* and R^2 values.

Code and results found in `reg-lstm-attn/Ensemble regression.ipynb`.

6.1.4 Regression with LSTM

Model Architecture:

- The first two LSTM layers have 128 and 64 units, respectively, and use 20% dropout rate to control overfitting. These two layers return sequences which are fed into a final LSTM layer. This stacked architecture enhances the model capacity and the ability to learn hierarchical features.
- The final LSTM layer has 32 units without any dropout. As the final LSTM layer that captures data dependencies, it does not return a sequence, outputting the last time step after the LSTM layer and feeding it into a multi-layer perceptron.
- The MLP consists of two fully connected, dense layers with ReLU activation functions. These layers have 96 and 128 units respectively, with one 20% dropout layer in between.
- The final RUL prediction is outputted by a 1-unit dense layer.

Training Process:

The model is trained for 10 epochs using Adam optimizer with a batch size of 128 and an initial learning rate of 0.001 given that SGD is more prone to the initial learning rate and often requires more precise finetuning. Training also includes a learning rate scheduler that reduces it by a factor of 10 after 5 epochs and an early stopping callback that monitors validation loss with a patience of 5 epochs. These callbacks generally improve stability and convergence while making the training process more robust and efficient. During training, two custom callbacks were written to dynamically plot model performance for visualization (see figures 12 and 13). Data-wise, 20% train-validation split was conducted randomly every time the model was trained to tune model architecture.

Figure 12. Regression LSTM training and validation losses curve

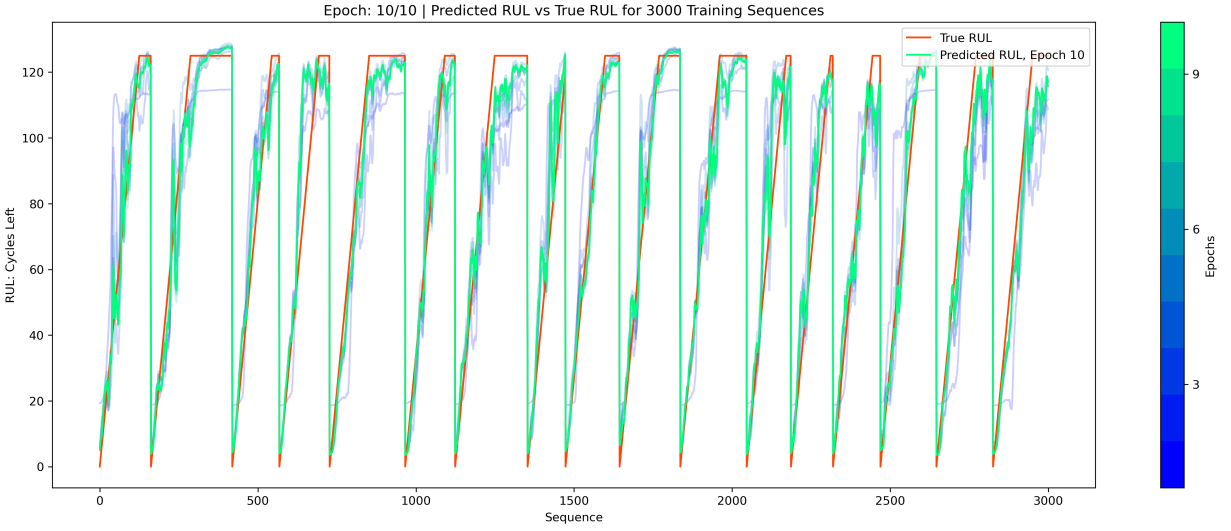


Figure 13. Regression LSTM performance on sample training data per epoch

Regression LSTM Model Performance:

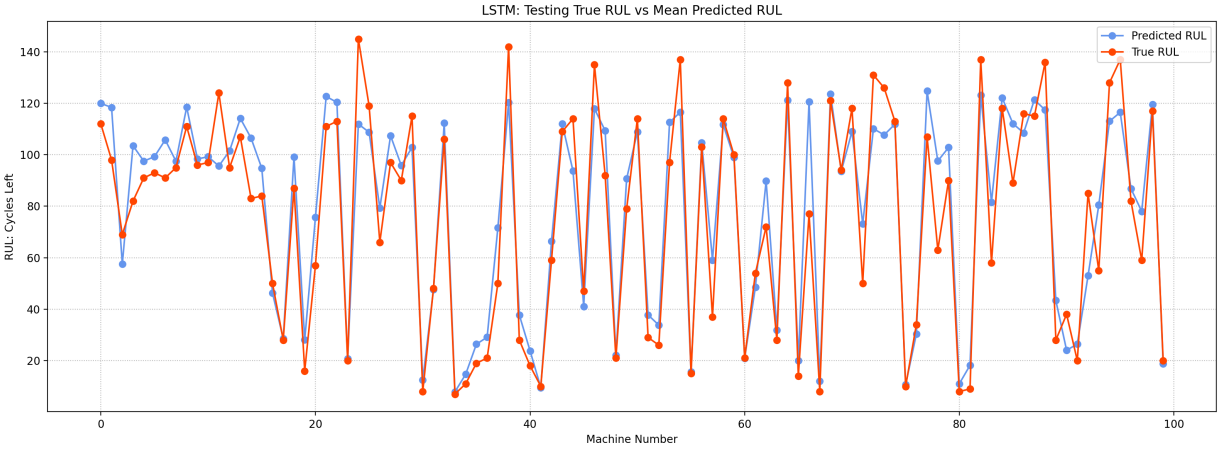


Figure 14. Regression LSTM performance on test data for 100 engines

Table 4. Regression LSTM test data metrics

	MSE	*RMSE*	*MAE*	R^2
Mean Prediction	198.1926	14.0781	10.8985	0.8852
Last Prediction	197.6742	14.0597	10.4926	0.8855

In contrast to the ensemble methods for regression, LSTM outperforms them noticeably with a significant decrease in test data MSE by almost 200 and a sizeable increase in R^2 value by around 10%. Additionally, the LSTM prediction using the last sequence as compared the its prediction using

the average of the last five sequences are very similar, unlike the trend seen in ensemble techniques. As such, it could be reasoned that LSTMs are better at analyzing dependencies given a stream of input data.

Code and results found in `reg-lstm-attn/LSTM regression.ipynb`.

6.1.5 Regression with LSTM and Attention

Model Architecture:

Instead of using stacked LSTM layers, the model has one LSTM layer and a multi-headed self-attention layer that attends to the LSTM output sequence.

- The first LSTM layer has 64 units and returns sequences which are fed into the multi-headed attention layer.
- The attention layer uses 8 heads and a key dimension of 96. The combination of LSTM and attention layer together captures long-range dependencies and refines the model focus onto relevant information.
- The attention layer output is average pooled through time to reduce the sequential output data dimension before passing it to the output dense layer.
- The final RUL prediction is outputted by a 1-unit dense layer.

Training Process:

The training process for this model mirrors the one prior. The model is trained for 10 epochs using Adam optimizer with a batch size of 128 and a learning rate of 0.001. The same callbacks were used (learning rate scheduler & early stopping). The train-validation split also remained the same at 20% validation during architecture tuning.

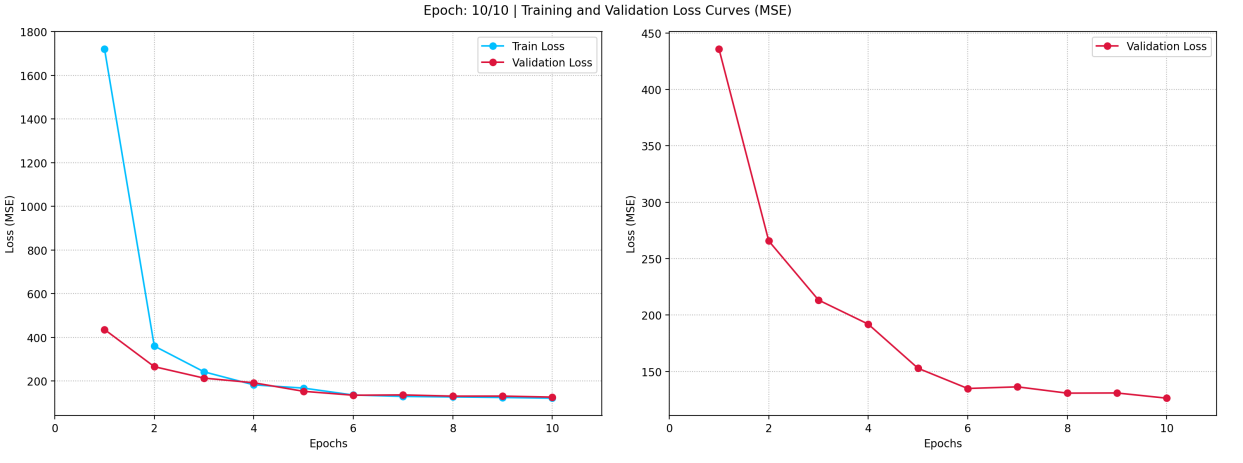


Figure 15. Regression with attention & LSTM training and validation losses curve

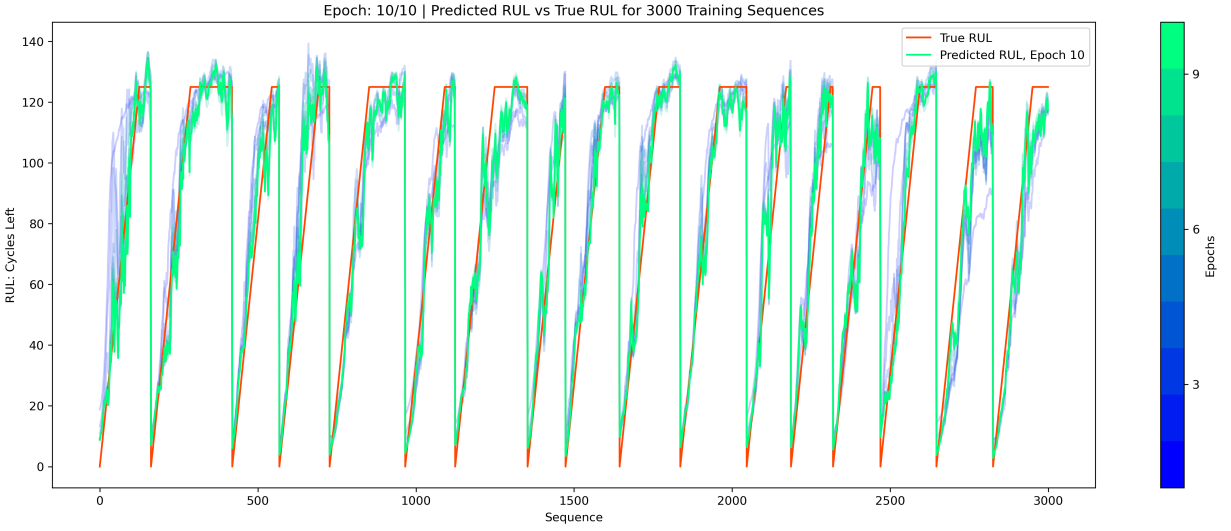


Figure 16. Regression with attention & LSTM performance on sample training data per epoch

Regression Attention & LSTM Model Performance:

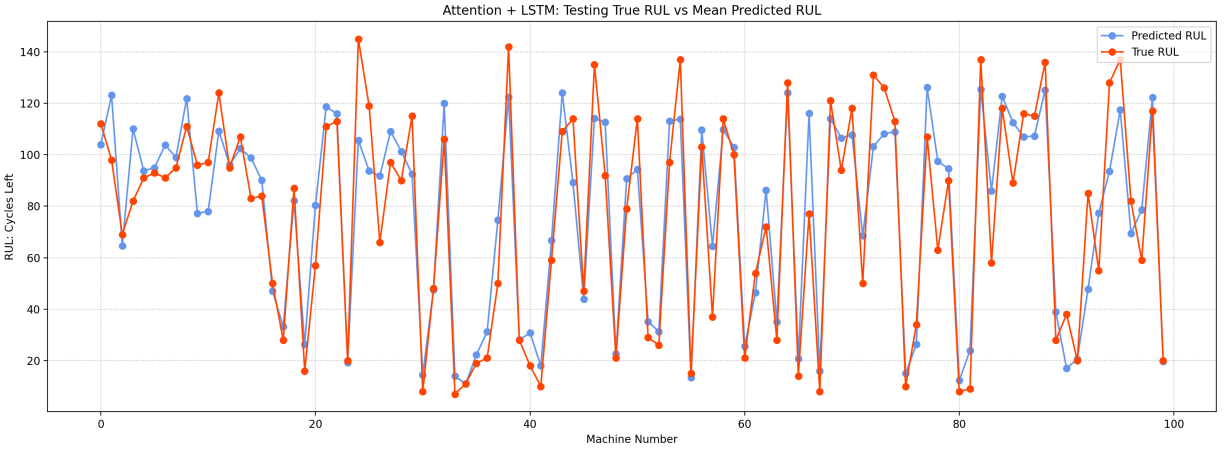


Figure 17. Regression attention & LSTM performance on test data for 100 engines

Table 5. Regression attention & LSTM test data metrics

	MSE	*RMSE*	*MAE*	R^2
Mean Prediction	250.3741	15.8232	12.5584	0.8550
Last Prediction	256.6015	16.0188	12.6121	0.8514

Compared with the regression LSTM model, the attention mechanism did not improve the model performance. While the metrics indicate the combination of attention and LSTM outperforms ensemble models, this model generalizes worse than the regression LSTM model. A potential reason

is the expressiveness of this model, which can lead to overfitting: the multi-head attention model has 220,033 learnable parameters while the stacked LSTM model has 152,289 parameters.

Code and results found in `reg-lstm-attn/Attention regression.ipynb`.

6.2 Classification Performance Evaluation

Evaluation Metrics:

1. Confusion Matrix: The confusion matrix was used to visualize the performance of the model in terms of TP, TN, FP, and FN, which is particularly useful for understanding the model's performance across different classes.
2. ROC Curve and AUC: The ROC curve and the area under the ROC curve (AUC) were employed for both models with the AUC score of each class in the multi-class model graphed. This provides a measure of the model's ability to distinguish between classes at various threshold settings.
3. Classification Report: Detailed classification reports were generated to evaluate precision, recall, and F1-scores for each class. These reports give insight into the model's performance on a class-by-class basis and avoid bias caused by the presence of heavy class imbalances.

6.2.1 Binary Classification with LSTM

In a binary classification setup, the LSTM model can be trained to identify the signs indicating imminent failure, which is determined to be within (w_0 = 15) cycles. This is useful for immediate maintenance decisions.

Model Architecture:

- The first LSTM layer has 100 units and is set to return sequences, outputting a sequence that is fed into the next LSTM layer, allowing for the capture of temporal dependencies across the entire sequence. A dropout layer follows, with a dropout rate of 0.2, to prevent overfitting by randomly dropping 20% of the units' connections during each training iteration.
- The second LSTM layer has 50 units and does not return sequences, meaning it only outputs the last step's output, condensing the temporal information into a single output vector. This is followed by another dropout layer with the same dropout rate.
- The final layer is a dense layer with a sigmoid activation function. Since this is a binary classification task, the sigmoid function is appropriate because it outputs a probability distribution between 0 and 1, allowing for a threshold-based classification.

Addressing Class Imbalance:

The dataset exhibits a significant class imbalance, with 17,531 instances of class 0 and only 3,100 of class 1. To address this, class weights were computed to ensure that the model does not become biased towards the majority class by giving more importance to the minority class during learning.

Training Process:

The model employs the Adam optimizer with a learning rate of 0.001, which was a logical choice due to its adaptive learning rate properties, allowing for quicker convergence. Early stopping is utilized with a patience of 5 epochs, to help prevent overfitting and ensure that the model generalizes well to new data. A small validation split of 5% is used to monitor the model's performance on unseen data during training as the dataset is reasonably large.

Binary LSTM Model Performance:

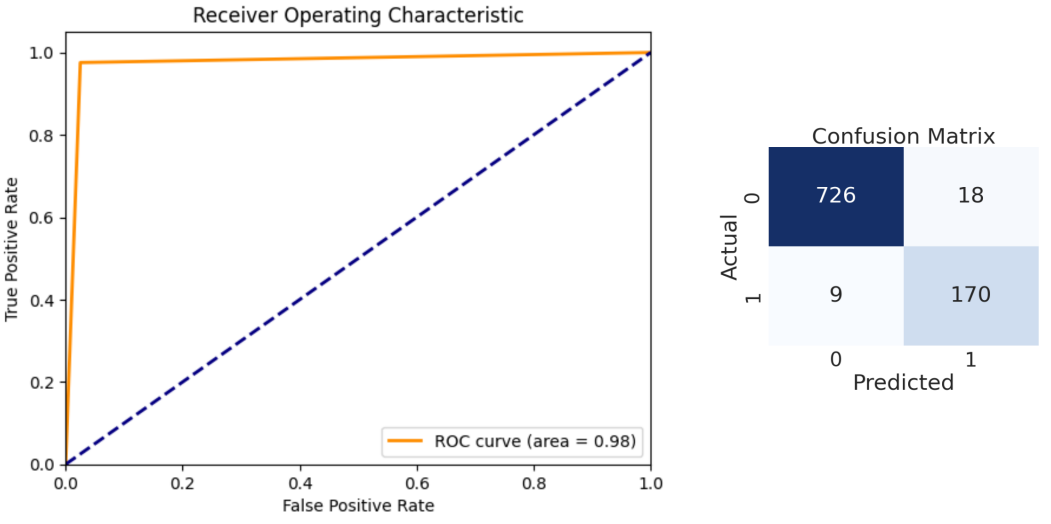


Figure 18. Binary LSTM results: ROC curve & confusion matrix

Table 6. Classification report for binary LSTM

	Precision	Recall	F1 Score	Support
Label 0	0.99	0.97	0.98	2514
Label 1	0.90	0.98	0.94	613
Accuracy	–	–	0.98	3127
Macro Avg	0.95	0.98	0.96	3127
Weighted Avg	0.98	0.98	0.98	3127

- Accuracy: The model's overall accuracy of 0.98 means that it is highly reliable in predicting the correct class for any given engine cycle.
- Class Imbalance Handling: The model successfully addresses the class imbalance issue, reflected in the high precision and recall for the minority class.
- Performance Superiority: When compared to a standard binary LSTM template model[3], this LSTM model shows superior performance, particularly in terms of recall and F1-score. This implies that while the template model was already quite precise, it failed to detect a considerable number of failures that the 2-layer LSTM model managed to catch.
- Operational Reliability: The high number of true positives and true negatives indicates that the model can be trusted in operational settings to predict engine failures and avoid false alarms effectively.

In conclusion, the binary LSTM model exhibits robust predictive performance, significantly outperforming the template model[3]. It is not only accurate but also reliable in identifying engines that are likely to fail, which is essential for proactive maintenance and operational efficiency.

Code and results found in `lstm/NASAlstmBinary.ipynb`.

3. https://cloudxlab.com/blog/predicting-remaining-useful-life-of-a-machine/

6.2.2 *Multi-Class Classification with LSTM*

For multi-class classification, the LSTM model categorizes the RUL into different time windows, such as failing within $[1, w_0]$ cycles, $[w_0 + 1, w_1]$ cycles, or not failing within w_1 cycles. The preparation of data involved an additional step of one-hot encoding the labels. This is necessary for multi-class classification tasks where the output layer of the model uses the softmax activation function. It allows the model to output a probability distribution over the multiple classes. This approach provides more granular information about the state of the engine's condition, aiding in long-term maintenance planning and resource allocation.

Model Architecture:

Initially, a more complex two-layer LSTM model similar to the one used for binary classification was implemented. However, it resulted in higher loss and lower accuracy. In response, a simpler model architecture with a single LSTM layer of 50 units was adopted. The improved performance of the simpler single-layer LSTM model for the multi-class classification task suggests that when dealing with the challenges of multi-class classification and class imbalance, a more complex model is not necessarily better. A simpler model can be more interpretable and less prone to overfitting.

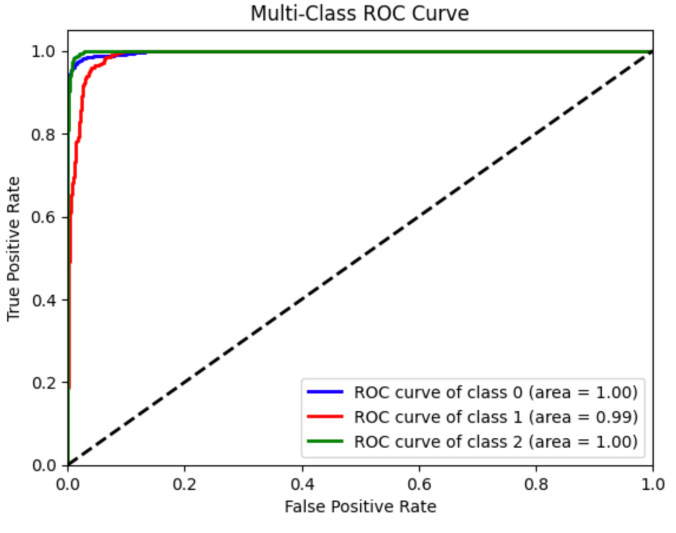
Addressing Class Imbalance:

Similar to the binary labels, the dataset displays a significant imbalance across the three classes, with the majority belonging to class 0. To address this, class weights were recalculated for the multi-class distribution and applied during training to achieve a balanced predictive performance across all classes.

Training Process:

The simpler LSTM model was trained using the Adam optimizer with a learning rate of 0.001. An early stopping callback was implemented with a patience of 5 epochs to prevent overfitting by halting the training process if the validation loss does not improve. Accounting for the increase in complexity, the validation split was increased to 10%.

Multi-Class LSTM Model Performance:



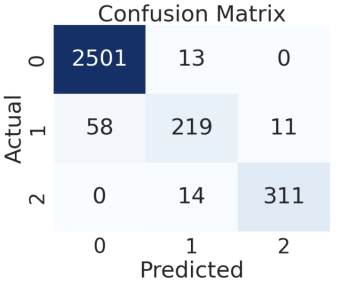


Figure 19. Multi-Class LSTM results: ROC curve & confusion matrix

Table 7. Classification report for multi-Class LSTM

	Precision	Recall	F1 Score	Support
Label 0	0.98	0.99	0.98	2514
Label 1	0.86	0.73	0.79	288
Label 2	0.95	0.95	0.95	325
Accuracy	–	–	0.96	3127
Macro Avg	0.93	0.89	0.91	3127
Weighted Avg	0.96	0.96	0.96	3127

- Balanced Performance Metrics: The precision, recall, and F1-scores across all classes show a balanced performance.
- Accuracy in Critical Class: The high degree of accuracy, especially for imminent failure, is essential for practical applications where the cost of false negatives — failing to detect an engine close to failure — can be extremely high.
- Comparison to Binary Model: Despite the inherent challenges posed by the multi-class classification, the model only shows a slight decrease in performance metrics (F1-score, precision, accuracy, recall) by 0.02 when compared to the binary LSTM model.

Code and results found in `lstm/NASAlstmMulti_class.ipynb`.

7. DISCUSSION

7.1 Regression Models Comparison

All the regression models except for linear regression computed two types of test predictions: one based on the last time series for each engine and one based on the average of the last five time series for each engine. Table 8 takes the best predictions from these models and compares them (along with the linear regression models).

Table 8. Comparison of test data performance metrics for all regression models

Algorithm	*MSE*	*RMSE*	*MAE*	R^2
Naive Linear Regression	2342.5101	48.3995	–	0.33
Lasso Regression	2619.7135	51.1831	–	0.25
RF Regressor	316.3018	17.7849	12.9322	0.8168
GBT Regressor	322.6763	17.9632	13.1359	0.8131
Regression LSTM	**197.6742**	**14.0597**	**10.4926**	**0.8855**
Regression Attention & LSTM	**250.3741**	**15.8232**	**12.5584**	**0.8550**

While machine ensemble models perform well, deep learning techniques see better generalizability in terms of all the metrics in table 8. Given the scope of this study, attention LSTM did not outperform the LSTM only model. However, it is worth noting that the regression LSTM outperforms the LSTM + FNN hybrid model (*RMSE* = 16.1) and sees very similar performance to online LSTM (*RMSE* = 14.0), both of which were mentioned in the background section.

7.2 Multi-Class Classification LSTM Novelty

The high accuracy in predicting the minority classes, despite the imbalance, underscores the model's robustness. This multi-class approach to predictive maintenance using LSTM is innovative in its application and has yet to be implemented in literature. It demonstrates LSTM's adaptability to not only binary classification tasks but also to multi-class scenarios, adding a new dimension to the prognostic capabilities of LSTMs.

7.3 Future Works

The findings here have demonstrated promising results, showcasing high-performing models in the domain of predictive maintenance. The success of LSTM architectures, particularly in regression tasks and classification of both majority and minority classes, underscores the potential of deep learning in this field. This success not only validates the effectiveness of the current approaches but also opens avenues for future enhancements and explorations. There is high potential for improvement and innovation, particularly in refining regression models for RUL estimation and expanding our methods to embrace new architectures and learning paradigms.

7.3.1 Enhanced Continuous and Transfer Learning Approaches

- Adaptive Training Over Time: Future research should focus on implementing adaptive training mechanisms that enable the model to evolve and improve over time. This includes leveraging new data as it becomes available, thus ensuring that the model remains current and effective in changing operational environments.
- Application of Pretrained Models and Fine-tuning: Exploring the use of pretrained Convolutional Neural Networks (CNNs) in the predictive maintenance framework can be a promising direction. These models, already trained on vast datasets, can provide a robust foundational knowledge base. Fine-tuning these models for specific PdM tasks could lead to improvements in accuracy and efficiency.
- Integration of Multimodal Learning: Investigating the incorporation of multimodal data sources, such as audio, vibration, and temperature readings alongside traditional sensor data, can enrich the learning process. This holistic approach can unveil intricate patterns missed by single-modality models, potentially enhancing the regression RUL prediction accuracy.

7.3.2 Advanced Ensembles and Hybrid Models

- Hybrid Ensembles of LSTM and Decision Trees with CNNs: Future models could experiment with advanced ensemble techniques that combine the strengths of LSTM architectures, decision trees, and CNNs. This hybrid approach aims to capture both the sequential nature of time series data (via LSTMs), intricate feature interactions (via decision trees), and spatial features in sensor data (via CNNs).
- Specialized Ensembles for Regression RUL Estimation: Given the moderate success of attention LSTM and regression LSTM in RUL estimation, further research should aim at developing specialized ensemble models that specifically enhance regression RUL prediction. These ensembles can integrate various advanced deep learning techniques, including attention mechanisms, to focus on providing more nuanced and detailed information crucial for predictive maintenance.

7.3.3 Investigating Alternative Architectures

- Exploring Transformer Models: Given the success of attention mechanisms, there is an opportunity to explore transformer models, which have shown exceptional performance in capturing long-range dependencies in data.

- AutoML and Neural Architecture Search (NAS): Implementing AutoML and NAS to automatically discover optimal architectures for our specific PdM tasks could lead to breakthroughs in model performance, especially in complex scenarios where manual tuning is not feasible.
- Balanced Learning for Minority Classes: It's also worth noting that significant focus should be on improving the performance of models in predicting minority class instances in multi-class LSTM setups. Techniques such as synthetic data generation, cost-sensitive learning, and targeted data augmentation can be explored to ensure the models are not biased towards majority classes.

8. CONTRIBUTIONS

Jonathan Zhang: Code for ensembles, regression LSTM, and attention LSTM. Paper writing: abstract, introduction, background, methodology, dataset description, data processing and results for the four mentioned models, and regression model comparison in discussion.

Sherry Rui: Code for binary and multi-class classification LSTMs. Paper writing: data processing and results for the two mentioned models, discussion, and future directions.

Lucas Goldfein: Code for linear regression. Paper writing: data processing and results for linear regression.

9. CODE

The codebase[4]: https://github.com/sherrui09/NASA_Predictive_Maintanence/tree/main

References

Ajay, Dangeti Saivenkat, Sneegdh Krishnna, and Kavita Jhajharia. 2023. Predictive maintenance of nasa turbofan engines using traditional and ensemble machine learning techniques. In *Advances in distributed computing and machine learning,* edited by Suchismita Chinara, Asis Kumar Tripathy, Kuan-Ching Li, Jyoti Prakash Sahoo, and Alekha Kumar Mishra, 369–379. Singapore: Springer Nature Singapore. ISBN: 978-981-99-1203-2.

Allah Bukhsh, Zaharah, Aaqib Saeed, Irina Stipanovic, and Andre G. Doree. 2019. Predictive maintenance using tree-based classification techniques: a case of railway switches. *Transportation Research Part C: Emerging Technologies* 101:35–54. ISSN: 0968-090X. https://doi.org/10.1016/j.trc.2019.02.001. https://www.sciencedirect.com/science/article/pii/S0968090X18309057.

Serradilla, Oscar, Ekhi Zugasti, Jon Rodriguez, and Urko Zurutuza. 2022. Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects. *Applied Intelligence* 52, no. 10 (August): 10934–10964. ISSN: 1573-7497. https://doi.org/10.1007/s10489-021-03004-y.

4. Dataset is not in the repository, but the link is included in the readme file.

Appendix 1. Max RUL: 125 cutoff

Figures 20 and 22 visualize the regression LSTM model training when the ground truth RUL vector is not limited.

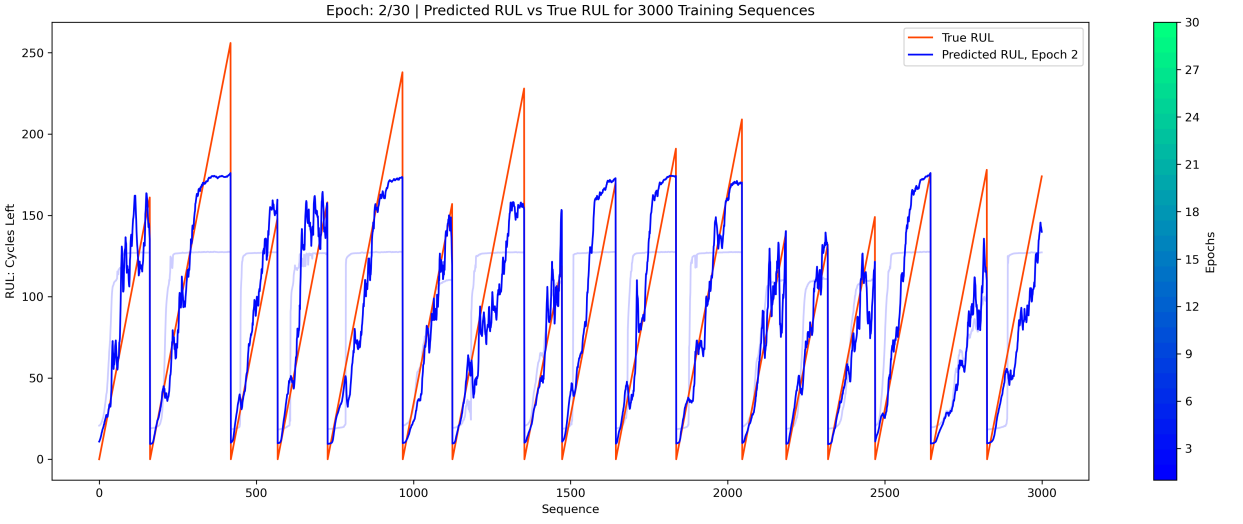


Figure 20. Regression LSTM performance on sample training data for 2 epochs: no RUL cutoff

At the beginning, the highest RUL that the model predicts is somewhere between 100 and 150.

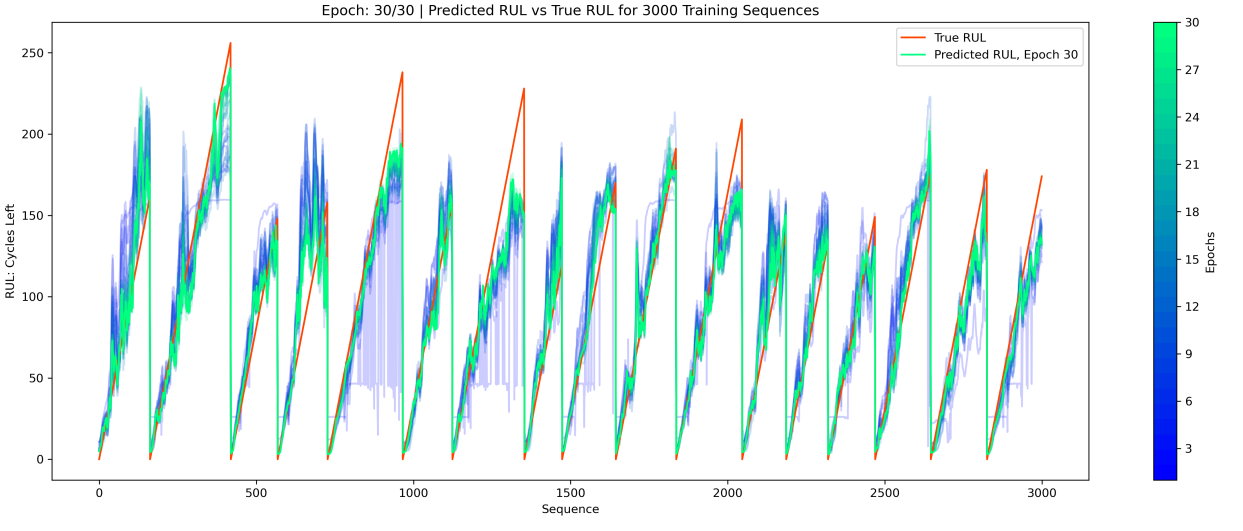


Figure 21. Regression LSTM performance on sample training data after 30 epochs: no RUL cutoff

Even after 30 epochs, the green line still shows that the model is struggling to correctly predict the peak RULs.

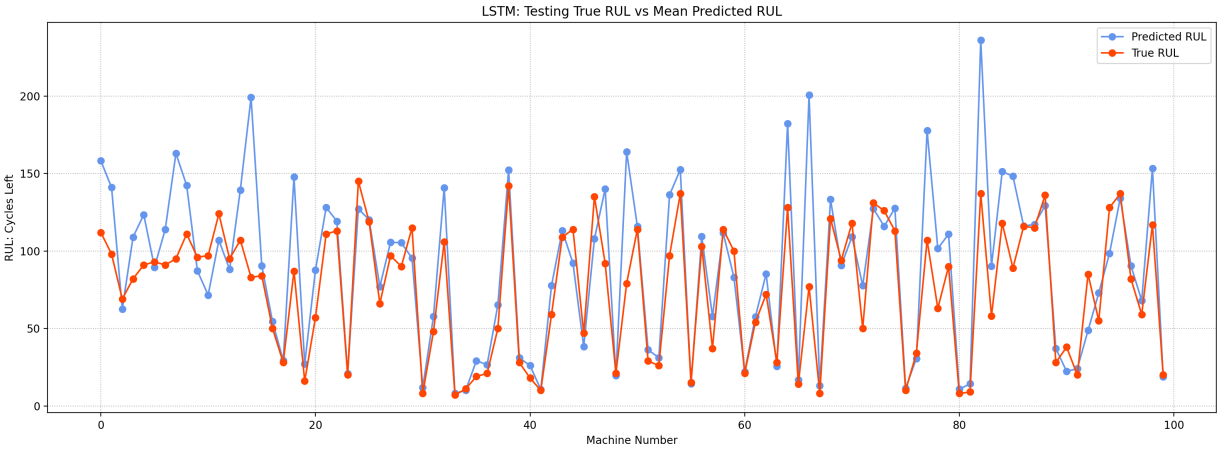


Figure 22. Regression LSTM performance on test data after 30 epochs: no RUL cutoff

However, in the test data, the model returns excellent predictions when the true RUL is low but overshoots many moderate to high true RULs. This could be caused by overfitting. However, if the model overfits sequences with low RULs before it learns to predict high RULs correctly, the solution might not be as simple as lowering the learning rate. Instead, setting a limit on the maximum RUL could help the model minimize validation loss and prevent overfitting at the same time.