CNN-LSTM Hybrid Model for Predictive Analytics

Jeremy Beal
Department of Electrical & Computer Engineering
Rowan University
Glassboro, New Jersey, United States
bealje59@students.rowan.edu

***Abstract*—In the field of time-series analysis, a variety of techniques and methods have been utilized to great success. With the advent of deep learning, certain models have emerged as equally or even more effective for forecasting in these scenarios. Common choices include recurrent neural networks (RNN), and more specifically, Long Short-Term Memory (LSTM) networks. LSTMs are particularly valued for their capability to retain information from previous inputs over extended periods. This project looks into the use of convolutional layers, predominantly utilized in object detection and image-related tasks, for initial feature extraction before passing into LSTM layers. The model showed efficiency, especially for engines with a single failure mode, whereas engines with multiple conditions faced challenges due to a conservative RUL prediction cap at 125 cycles. By refining the statistical approach to focus on critical RUL predictions and adjusting the piecewise linear degradation function, the model's performance improved. This research suggests that with appropriate data preprocessing, the CNN-LSTM framework can be a valuable tool for multivariate time-series forecasting in predictive maintenance and beyond, with potential enhancements from further tuning and feature selection.**

I. Introduction

Predictive analytics encompasses a diverse range of applications across various industries and research areas. This project zooms in on a specific subset of predictive analytics: predictive maintenance. In contrast to preventative maintenance, which is conducted periodically, and reactive maintenance, which occurs post-failure, predictive maintenance focuses on determining the remaining useful life (RUL) of a system. RUL estimates the lifespan left in a system, usually in units like time or cycles, based on data inputs.

In the context of predictive maintenance, multivariate time series data, such as sensor readings, play a crucial role. The aim is to feed a sequence of data into a model that predicts the system's RUL. Long Short-Term Memory (LSTM) networks, known for their strength in time series analytics due to their ability to retain information over extended periods, are well-suited for this task. However, we also explore the use of Convolutional Neural Networks (CNNs), typically used in grid-like data formats like images, for their feature extraction capabilities in time series data. In the proposed model, the data first passes through convolutional layers, allowing the model to identify spatial relationships within the data. Subsequently, it enters LSTM layers that focus on capturing longer-term, temporal dependencies. This approach aims to leverage the strengths of both CNNs and LSTMs for a more accurate predictive maintenance strategy.

This hybrid model approach has demonstrated effectiveness across various domains, particularly in analyzing time series data. For instance, the study [1] integrates diverse brain signal data into a model structure similar to the proposed one, to explore and predict sleep disorders.

II. Methods

A. NASA C-MAPSS Jet Engine Simulated Data

To conduct a thorough assessment of the proposed model architecture, a suitable dataset was essential. After evaluating various options in the field of preventative maintenance, NASA's C-MAPSS Jet Engine Simulated Data was selected for its comprehensive and detailed nature. Known as a standard in the field, C-MAPSS offers simulated scenarios that mimic aircraft engine wear and breakdown, aiming to forecast engine failures and estimate the RUL of engine components.

This dataset is a multivariate time series collection, featuring an array of sensor data and operational metrics, including fuel flow, fan speed, temperature, and pressure, each correlated with specific operational cycles and a unique unit identifier. Depth and variety in the data offer the potential for deep learning algorithms to learn how to predict engine life over time.

C-MAPSS is divided into two segments: training and testing datasets. The training sets lack explicit RUL values, requiring a strategic interpretation and application of the data. This characteristic of the dataset not only makes it ideal for validating the effectiveness of the model but also encourages innovative approaches to predictive analysis in maintenance. There are 4 total datasets representing 4 different simulated engine instances with specific failure conditions and fault modes.

TABLE I
Summary of C-MAPSS Dataset

Data Set	Train Units	Test Units	Conditions	Fault Modes
FD001	100	100	ONE	ONE
FD002	260	259	SIX	ONE
FD003	100	100	ONE	TWO
FD004	248	249	SIX	TWO

In reference [3], the process of modeling and simulating the jet engine's failure has been explained. For instance, engine 1 fails when it is at sea level due to HPC (High-Pressure Compressor) degradation. Engine 2, on the other hand, is more

complex as it fails due to six different conditions. Engines 3 and 4 are similar to engines 1 and 2 respectively, but they have an additional fan degradation failure mode. Each dataset has 26 distinct features shown below in Table 2.

TABLE II
C-MAPSS DATASET FEATURES

No.	Feature Description
1	Unit number
2	Time, in cycles
3	Operational setting 1
4	Operational setting 2
5	Operational setting 3
6	Sensor measurement 1
7	Sensor measurement 2
...	
26	Sensor measurement 26

The general flow of the data within each dataset involves values for sensors, measurements, and operational settings for every available unit as time (in cycles) progresses. As previously mentioned, RUL is up to the interpretation of the user.

Although there are technically 24 unique features which consist of 3 operational settings and 21 sensor values/measurements, only a smaller subset of features was used for training. The operational settings were removed for all 4 engines, with a focus on learning patterns in sensors/measurements exclusively. For engines 1 and 3, the operational setting values remained mostly constant, so including them could possibly add more noise to the training data. However, engines 2 and 4 show significant fluctuations in operational settings 1 and 3 as time progresses across different units. It may be worth considering the inclusion of operational settings in future training experiments for engines 2 and 4 to examine their impact.

In addition to removing the operational settings, columns related to sensors and measurements were also eliminated if their standard deviation was less than 0.01. These columns were deemed insignificant in determining the RUL since they were nearly or exactly constant.

B. Data Windowing/Sequencing

Both CNNs and LSTMs require sequential input data for optimal functioning. However, generating sequences from datasets like C-MAPSS poses specific challenges, including determining the appropriate sequence length or window size and deciding whether to incorporate overlapping sequences/windows.

In this report, I followed the approach of finding the maximum possible window length for each dataset, the smallest number of cycles per unit in the test dataset dictates the maximum possible window length. Since test datasets often have less data, they typically yield smaller possible windows compared to the larger train datasets. Additionally, if the chosen window length was larger than the number of cycles that a unit in the test dataset lasted, that unit could not be used for testing. For instance, unit 83 in the test set for engine 2 has the shortest runtime in any unit run between both the test and train sets, with 21 cycles. The max window length for this dataset would be set to 21, and a list of increasingly smaller window lengths would be generated for training as a hyperparameter to tune.

In Figure 1, you can see the basic structure of the raw dataset for a single unit, compared to the overlapping windowed structure that was created for training. The overlapping windowing approach was chosen to make sure that any dependencies between windows were not disrupted. This approach ensures that successive data windows maintain some level of continuity and context, which is essential for capturing temporal dependencies in the data.

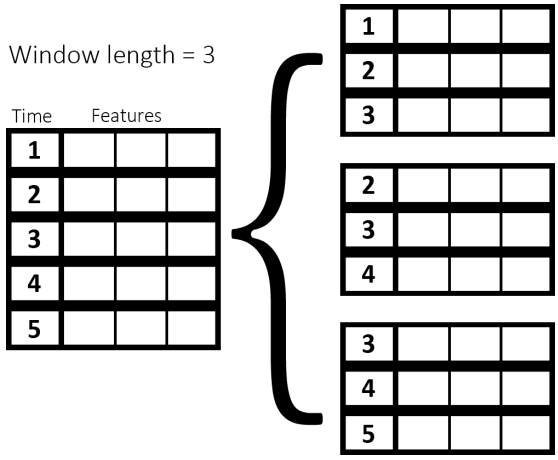

Fig. 1. Creating Windows from Unit Data

C. RUL in Training Data

There are various methods to create RUL (Remaining Useful Life) labels for train datasets. One simple approach is to use a linear degradation trend. In this method, the RUL value for the beginning of a unit's data is set as the known failure time of that unit. As time progresses, the subsequent RUL values gradually decrease until they reach zero at the end of the unit's data. This approach was commonly used and moderately effective. However, recent publications, such as [2], have suggested a piecewise linear degradation trend. This trend starts with a static RUL value until the time for a unit reaches a threshold where it can gradually decrease to 0 from that static value. The rationale behind this is that the sensor values do not signify degradation in the system in earlier time steps. The integration of this piecewise model has proven to be more indicative of the story that the sensor values/measurements are conveying and thus has led to more accurate models. A comparison between the two popular trends is shown below in Figure 2.

To calculate the piecewise trend for a specific unit, the first step is to determine the total number of possible sequences

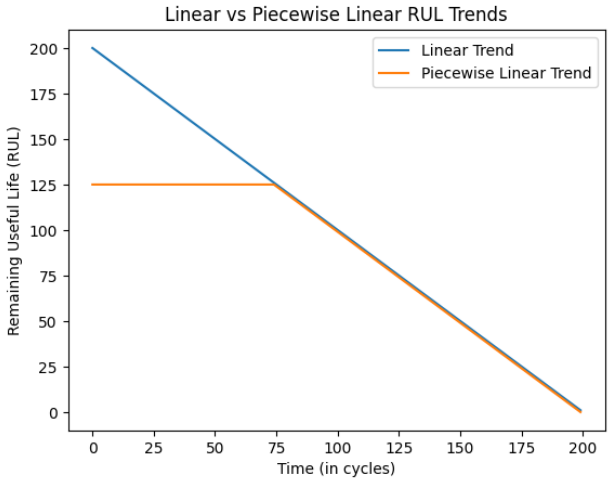

Fig. 2. Comparison of Linear vs Piecewise Linear RUL Trends.

for that unit. The windows or sequences overlap and proceed sequentially. As a result, the final window will begin from the length of the sequence subtracted from the overall runtime of the unit plus 1.

$$\text{Last Window Start} = \text{Total Unit Time} - \text{Sequence Length} + 1 \quad (1)$$

In addition to (1) being the index of the start of the last window, it also denotes the total number of windows for the unit. Now, the piecewise function can be constructed on a per-unit basis.

Algorithm 1 Generate Train RUL for Single Unit

procedure GETTRAINTARGETS(num_seq_train)
  if $num\_seq\_train < rul\_max$ **then**
    return RangeList($num\_seq\_train - 1, -1, -1$)
  else
    $rep\_part \leftarrow$ RepeatList(rul_max, $num\_seq\_train - rul\_max$)
    $dec\_part \leftarrow$ RangeList($rul\_max - 1, -1, -1$)
    return Concat(rep_part, dec_part)
  end if
end procedure

If the number of sequences is less than the maximum static value, the RUL follows a basic linear degradation trend. Otherwise, concatenate a repeated list with the static value to the degradation part.

D. Model Architecture

As previously mentioned, the proposed model architecture involves both convolutional and LSTM layers. At this point, it's worth delving into why convolutional layers are effective in this scenario. Though primarily known for their strength in working with input data that comes in a grid-like format (i.e. images), they are also adept at processing sequential data, such as time series or signal data, which can be thought of as a 1D grid. Convolutional layers can identify and extract local patterns within the data due to their convolutional filters, which slide across the input space [4]. This is particularly beneficial for time-series data where such local patterns can represent important features like trends, spikes, or drops. Including these layers at the start of the model helps in reducing the dimensionality of the input data. By learning higher-level features from the raw data through a hierarchy of filters, each is responsible for detecting features at a different level of complexity. This hierarchical feature extraction is particularly useful in simplifying the data before it is fed into LSTM layers.

LSTM layers are a standard and well-tested choice for time series analysis including predictive maintenance, primarily because they are designed to remember information for long periods of time, which is essential for capturing long-term temporal dependencies in sequential data. LSTMs have gates that regulate the flow of information: the input gate decides which values are important to keep, the forget gate decides what data can be discarded, and the output gate determines what the next hidden state should be. This architecture allows LSTMs to learn which patterns are important for making predictions and which can be ignored, making them particularly good at dealing with the noise that is often present in real-world sensor data.

The idea is that the convolutional layers will extract relevant, high-level features before passing them to the LSTM layers. The LSTMs should excel in understanding the underlying temporal dependencies these features bring. In a more general sense, the convolutional layers will gather *spatial* info while the LSTMs will pick up on temporal info [4]. Note that the term spatial is used cautiously because, in the context of time series data, it doesn't refer to physical space but rather to the structure and relationships within the data at a given time step (Unlike image data, where spatial information pertains to the arrangement of pixels in two or three dimensions). It is worth noting that this type of hybrid model is not a completely original concept. These models have been employed for various tasks including ones similar to the task at hand (see [1] [4]). Table 3 shows the baseline architecture with some exceptions.

The number of LSTM layers is taken as a hyperparameter in the range [2, 3, 4], and the hidden size for each layer was set to 32. The first block of convolutional layers has kernel sizes of 3 to gather smaller, local features in the data. The second block ups the kernel size to 5 in hopes of gathering more global, abstract features. This decision also sacrifices spatial resolution to some extent. Max pooling layers in this setup serve to shorten the sequence length, which makes processing faster and less resource-intensive. At the same time, these layers retain the most important features in each segment of the data, ensuring that key information is preserved for further analysis. Dropout is in place after both the convolutional layers and the LSTM layers to avoid overfitting.

TABLE III
RUL PREDICTION MODEL ARCHITECTURE

Layer	Input/Output	Kernel/Stride	Parameters
Conv1d	in_feat/out_64	k:3, p:1	-
ReLU	-	-	-
Conv1d	in_64/out_128	k:3, p:1	-
ReLU	-	-	-
MaxPool1d	-	k:2, s:2	-
Conv1d	in_128/out_128	k:5, p:2	-
ReLU	-	-	-
Conv1d	in_128/out_256	k:5, p:2	-
ReLU	-	-	-
MaxPool1d	-	k:2, s:2	-
Dropout	-	-	p:0.5
LSTM	in_256/h_size	-	batch_first
...	...	-	...
Dropout	-	-	p:0.5
Linear	h_size/out_cls	-	-

E. Training Process

The train datasets created from the sequencing and windowing process were divided into training and validation tests. The training sets contained 80% of the data, while the remaining 20% was used for validation. Pytorch data loaders were used for batching (batch size = 64). The train sets were shuffled to promote generalization and discourage overfitting, but the validation sets were not to ensure consistent evaluation.

During the training process, I chose three hyperparameters, namely sequence/window length, number of LSTM layers, and learning rate. The sequence length parameter was generated dynamically based on the current engine or dataset, as explained in the *Data Windowing/Sequencing* section. I also varied the number of LSTM layers from 2 to 4, as previously mentioned. Additionally, I experimented with different learning rates such as 0.001, 0.0005, and 0.0001. Adam was the optimizer of choice due to its reputation and success in similar tasks [5].

F. Evaluation Process

During the training loop, the model was set to evaluation mode, and gradient calculation was disabled for validation and testing. For training and validation loss, mean-squared error was used.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2}$$

where y_i represents the actual values, $\hat{y}_i$ represents the predicted values by the model, and n is the number of observations in the dataset.

To get a testing metric, RMSE, or root mean squared error, was used.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

During the hyperparameter sweep, a dictionary stored each engine's best test RMSE (initially set to float("inf")) and saved the model if a new best score was achieved. While RMSE is widely accepted and used, it's a symmetrical scoring function which means it assigns equal weights or penalties to both RUL prediction underestimates and overestimates. In the predictive maintenance realm, an asymmetrical scoring function could be of use. For example, a prediction that ends up being an underestimate in this scenario could very well be more costly as opposed to an overestimate. The following metric was sourced from [6].

$$\text{score} = \sum_{j=1}^{N} s_j \tag{4}$$

$$s_j = \begin{cases} e^{-\frac{h_j}{13}} - 1, & \text{if } h_j < 0 \\ e^{-\frac{h_j}{10}} - 1, & \text{if } h_j \geq 0 \end{cases} \tag{5}$$

The total difference between each true RUL and each predicted RUL is calculated and added up. The score is asymmetrically calculated. If the difference between the true and predicted RUL is negative, indicating that the prediction overestimated the RUL, a smaller value will be added. Conversely, if the difference is positive, a larger value will be added. A lower score is preferred. This metric ensures that incorrect predictions will lead to undesirable scores in either direction, adding complexity to better suit the problem.

III. RESULTS

After training through each hyperparameter value and using test RMSE to determine the best models for each engine, the following hyperparameter combinations and model architectures were found. These combinations are listed in Table 4.

TABLE IV
BEST MODEL CONFIGURATIONS

Engine	LR	Num LSTMs	Seq. Length
1	5e-4	4	31
2	5e-4	2	11
3	5e-4	2	28
4	1e-3	4	9

It is important to mention that even though the tuned hyperparameters performed better than other configurations, most of the other configurations were able to come close to the best configuration. Moreover, the parameters did not entirely determine the success or failure of the model but instead helped to assess different configurations. You can find the raw RMSE and S-metric scores obtained using the above *best* configurations in Table 5.

TABLE V
BEST MODEL RESULTS

Engine	RMSE	S-Metric
1	13.64	267.24
2	27.73	7366.42
3	13.94	359.90
4	30.75	8890.43

It appears that engines 1 and 3 may have presented easier problems to solve, at least in the context of machine learning. Both have significantly lower RMSE. The difference is further exaggerated when examining the S-metric. This will be explored further in the ”Conclusions” section.

It’s important to understand how the test set is used. Each engine has a test dataset with only *one* RUL value provided for each unit. This raises a question about what data should be used to get the best prediction from the trained model. One approach is to extract the last possible window of data from each unit and pass it into the model. However, this approach may not be efficient as it may not make use of all the available test data for the unit. To carry out tests, a more sophisticated method involves examining a specific number of test windows, commencing from the last feasible test window. These testing procedures involved the specification of five test windows. If the five test windows could not be obtained due to limitations between the window length and total unit runtime, the maximum possible number of windows was obtained as an alternative. The test function would then make predictions using each of these sequences independently, and the mean between each forecast would be calculated as the final prediction. This approach allowed more of the available test data to be incorporated into the testing process.

Figure 3 shows the true RUL (blue) vs the averaged predicted RUL (orange) for each unit (for all engines). Note that the x-axis is in units as opposed to something like time in cycles.

The performance of engines 1 and 3 has been objectively good, as indicated by their RMSE, S-metric score, and the visual representation in Figure 3. This performance is particularly noticeable when the RUL is relatively low. The models appear to have effectively identified patterns and trends in the sensor values over time that are related to degradation. There are only a few instances of underestimation, and even when they do occur, the predicted values are not significantly off, resulting in a good S-metric score.

Engines 2 and 4, as discussed, are a bit more complex in comparison. However, Figure 2 highlights a potential issue with the training process. There are a large number of overestimates, particularly when the true RUL is over 125 (which is the static value that was chosen as part of the piecewise linear degradation function before pure linear degradation starts). This, again, will be discussed in the conclusions.

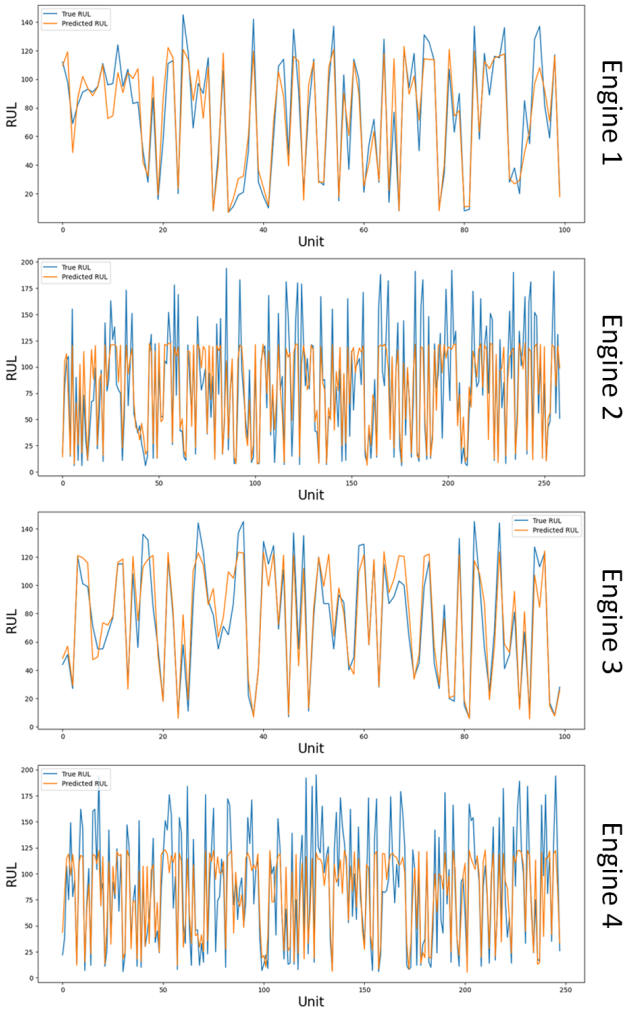

Fig. 3. True RUL, Mean Predicted RUL per Unit

IV. Analysis / Conclusions

Referencing the ”Methods” section of this report, namely subsection *A*, engines 1 and 3 only contain one condition for failure. This is in contrast to engines 2 and 4, which have 6 conditions for failure. Although the problem could potentially be more challenging to learn, the issue with engines 2 and 4 is evident in the calculation for train RUL. During training, the maximum value each engine can reach in terms of RUL is manually set to 125. As a result, the models are usually reluctant to predict values beyond 125. Although this is an expected behavior, it negatively affects overall performance when the true RUL is more than 125, as the model is incapable of predicting that value. There are some merits to this, and there is a solution to extract a condensed/refined set of statistics that may better represent the performance of each model.

Accurate predictions for any set of sensor values are desirable in industry or practical applications. However, if you are not looking for a long-term time-planning strategy, there may be a lesser need to know if a system, such as a jet engine, is predicted to fail after more than 125 cycles. The critical predictions would come when failure starts to become more and more likely and this is where these displayed models are most effective. It has been observed that the engine models created for engines 1 and 3 are better than the models created for engines 2 and 4. The predictions generated by the models for engines 2 and 4 tend to be underestimated more frequently than those created for engines 1 and 2, which is a more undesirable outcome. This, mixed with the abundance of overestimates, results in a higher S-metric score.‘

To create a more refined set of statistics, it is necessary to remove any RUL values that are greater than 125. The

reasoning behind this is that any value greater than 125 is practically the same as 125, as our main focus is on accurately predicting lower RUL values that are more critical. Doing so results in a new table of results, Table 6.

TABLE VI
BEST MODEL RESULTS (REFINED)

Engine	RMSE	S-Metric
1	12.51	221.95
2	19.44	2679.12
3	13.13	314.05
4	21.88	4403.35

We have observed that there are slight improvements in the performance of engines 1 and 3, while engines 2 and 4 have shown substantial improvements. This was expected, as the predictions made by the model for engines 2 and 4 had larger actual RUL values, leading to overestimation and an increase in both the RMSE and S-metric. To be precise, engine 2's model showed a decrease of 4687.3 in S-metric score, while engine 4's model had a decrease of 4487. Figure 4 displays the updated plot, representing the true RUL and mean predicted RUL per unit.

The dramatic overestimations are no longer occurring since they have been eliminated. The non-ideal S-score metric now largely stems from the underestimates seen in Figure 4.

It has been found that the hybrid CNN LSTM network is efficient when applied to NASA's C-MAPSS dataset. It is expected that this effectiveness can be extended to other multivariate time-series datasets provided that the data pre-processing is appropriate.

The introduction of convolutional layers before LSTM layers allows for an effective abstraction of features from the raw time-series data, which the LSTM layers can then process to capture temporal patterns. This hierarchical approach to feature extraction and time-series analysis has proven to be beneficial, as indicated by the results obtained from the C-MAPSS dataset.

The models' tendency to limit predictions to a maximum RUL value of 125 suggests that while the piecewise linear degradation function is a good fit for engines with simpler failure modes, it might need adjustment for more complex scenarios. Future work could involve exploring different configurations of the piecewise function, possibly by incorporating domain knowledge or by learning the degradation pattern directly from the data.

The refined results highlight the importance of focusing on the critical window where accurate predictions of RUL are most valuable. In practical applications, precise predictions of RUL, when systems are nearing failure, can be more impactful than long-term predictions, which may have less immediate practical use.

In conclusion, this project has demonstrated the potential of CNN-LSTM hybrid models in the domain of predictive maintenance. The study has demonstrated that the approach is strong and flexible, capable of adjusting to the intricacies of multivariate time-series data. Further investigations should be conducted to assess the scalability of this approach to other datasets and domains. Furthermore, I suspect that the current model suggested here is not performing at its maximum potential. This could be improved by modifying the training RUL assignment process, adjusting hyperparameters, such as the constant before degradation, and carrying out more advanced feature selection.

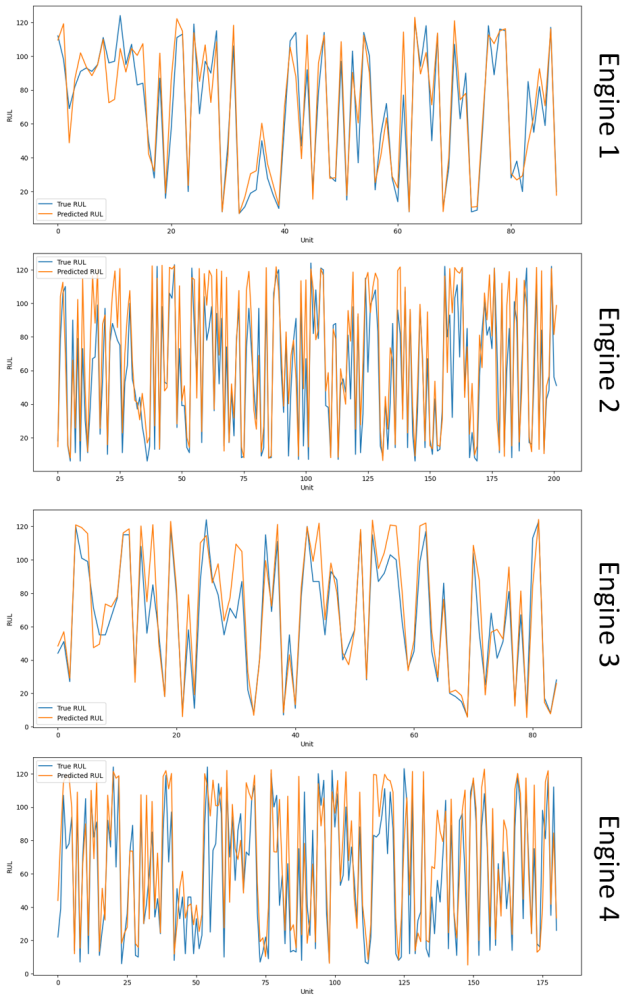

Fig. 4. True RUL, Mean Predicted RUL per Unit (refined)

REFERENCES

[1] Zhao D, Jiang R, Feng M, Yang J, Wang Y, Hou X, Wang X, "A deep learning algorithm based on 1D CNN-LSTM for automatic sleep staging," Technol Health Care, vol. 30, no. 2, pp. 323–336, 2022, doi: 10.3233/THC-212847. PMID: 34180436; PMCID: PMC9028677.

[2] S. Zheng, K. Ristovski, A. Farahat and C. Gupta, "Long Short-Term Memory Network for Remaining Useful Life estimation," 2017 IEEE International Conference on Prognostics and Health Management (ICPHM), Dallas, TX, USA, 2017, pp. 88-95, doi: 10.1109/ICPHM.2017.7998311.

[3] A. Saxena, K. Goebel, D. Simon and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," 2008 International Conference on Prognostics and Health Management, Denver, CO, USA, 2008, pp. 1-9, doi: 10.1109/PHM.2008.4711414.

[4] Weiyi Zhang, Haiyang Zhou, Xiaohua Bao, Hongzhi Cui, "Outlet water temperature prediction of energy pile based on spatial-temporal feature

extraction through CNN–LSTM hybrid model", Energy, Volume 264, 2023, 126190, ISSN 0360-5442

[5] Abdulkadirov R, Lyakhov P, Nagornov N. "Survey of Optimization Algorithms in Modern Neural Networks". Mathematics. 2023; 11(11):2466. https://doi.org/10.3390/math11112466

[6] O. Asif, S. A. Haider, S. R. Naqvi, J. F. W. Zaki, K. -S. Kwak and S. M. R. Islam, "A Deep Learning Model for Remaining Useful Life Prediction of Aircraft Turbofan Engine on C-MAPSS Dataset," in IEEE Access, vol. 10, pp. 95425-95440, 2022, doi: 10.1109/ACCESS.2022.3203406.