REMAINING USEFUL LIFE ESTIMATION FOR AIRCRAFT ENGINES USING XGBOOST AND LONG SHORT-TERM MEMORY NETWORKS

HERIBERTO ARGIL III

1. Introduction

Predictive maintenance plays a critical role in the aviation industry as balancing maintenance schedules and avoiding unexpected engine failures can significantly impact operational costs. Accurate prediction of Remaining Useful Life (RUL) allows for timely maintenance decisions, reducing both risk and expense.

This study focuses on evaluating and comparing two machine learning models for RUL prediction using the C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) dataset, developed by engineers at NASA to support research in predictive maintenance. The C-MAPSS dataset simulates run-to-failure scenarios for turbofan engines and contains multivariate time-series data, including sensor readings and operational settings recorded over each engine's life cycle[1]. Due to its detailed simulation of engine degradation and structured failure data, it is widely used as a benchmark in predictive maintenance studies [1, 2, 3].

To explore different modeling approaches for RUL prediction, this study uses XGBoost and a Long Short-Term Memory (LSTM) network, as each offers different strengths in predictive accuracy and interpretability. The models are evaluated using Root Mean Squared Error (RMSE) for comparison both against each other and against other model architectures proposed in the literature for this dataset. XGBoost serves as a baseline and is interpreted using Shapley Additive Explanations (SHAP) to highlight feature influence [4], while the LSTM model is chosen for its ability to learn temporal patterns and is trained with a custom quantile loss function to generate both point predictions and prediction intervals[5].

The remainder of this paper is organized as follows. Section 2 presents the exploratory data analysis, Section 3 describes the modeling methodology, Section 4 discusses results and comparison to related works; and Section 5 concludes the paper and outlines future work.

2. Exploratory Data Analysis

2.1. Dataset Overview. The C-MAPSS dataset consists of four subsets, labeled FD001 through FD004. Each subset represents turbo aircraft engines, with 100 - 250 engines of the same type operating under different conditions. These varying conditions lead to various operating faults, which are intentionally left with generic names. This omission is deliberate, as noted in [1], to simulate a practical scenario where not all sensor details are available due to proprietary restrictions.

Date: May 5, 2025.

This study specifically focuses on the FD001 subset. FD001 contains time series data representing complete engine lifespans, from startup to shutdown (measured in cycles), for 100 individual turbo engines. The dataset includes engine ID, cycle count, three operational settings, RUL, and sensor readings from Sensor 1 - Sensor 21. No missing values were observed in this subset.

2.2 Preprocessing Steps. To prepare the data for modeling, Standard Scaling was applied to all features. This technique transforms each feature to have a mean of 0 and a standard deviation of 1, using the formula :

$$x_{\text{stand}} = \frac{x - \mu}{\sigma} \tag{0.1}$$

where μ is the mean and σ is the standard deviation of the feature.

Standardizing ensures that all features are on a comparable scale, as it is crucial for the LSTM model to maintain gradient stability during training. Although XGBoost is generally robust to feature scaling, applying it uniformly across models simplifies preprocessing and avoids potential discrepancies in feature importance calculations.

2.3 Sensor and Operational Distributions. As shown in in Figure 1, many of the sensors follow a normal distribution, and this pattern is consistent across most features in the dataset. Because of this, only a small subset was plotted. Though most features follow a consistent pattern, a few differ slightly, such as Sensor 6 and Operational Setting 3. Sensor 6 shows very little variation, and Operational Setting 3 remains nearly constant across all engines. No major feature excursions were found, and the distributions were stable enough to move forward with model development without further filtering.

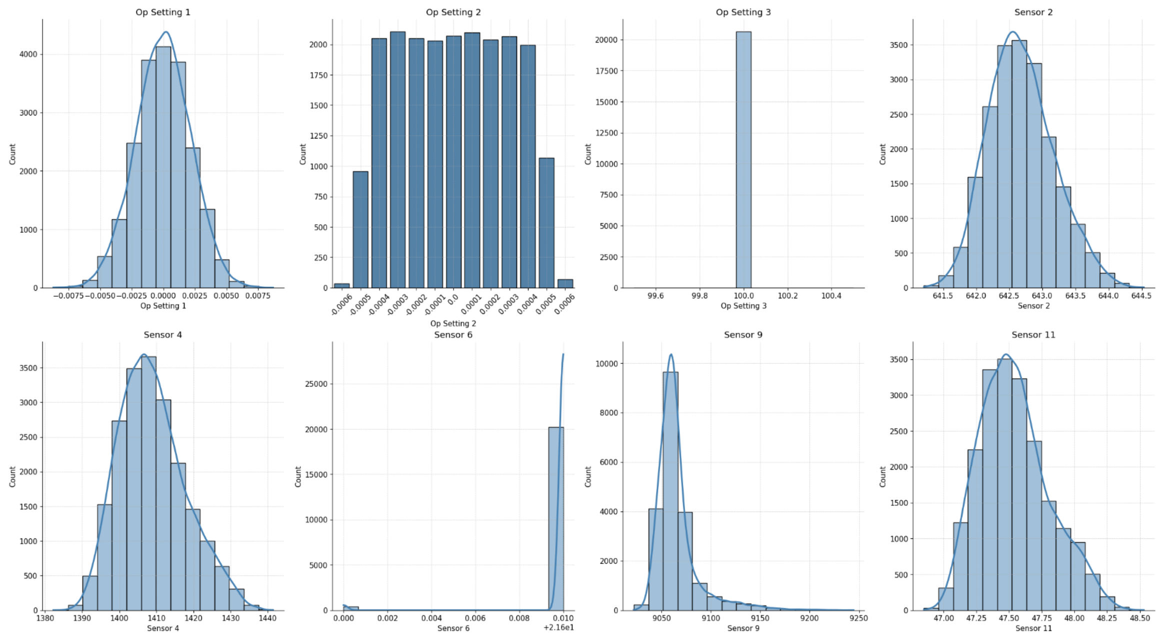

Figure 1. Distributions of selected operational settings and sensor features.

2.4 Engine Run Length Distribution. Figure 2 shows the distribution of engine run lengths (maximum cycles before failure) for FD001. The histogram shows a positive skew, with the majority of engines failing at 100–250 cycles, and a small number reaching 300–350 cycles. This skew suggests that the models will need to generalize across both short and long lifespans. This variability also shows why it is important to use models that account for uncertainty, so they can better capture how engine degradation differs from case to case.

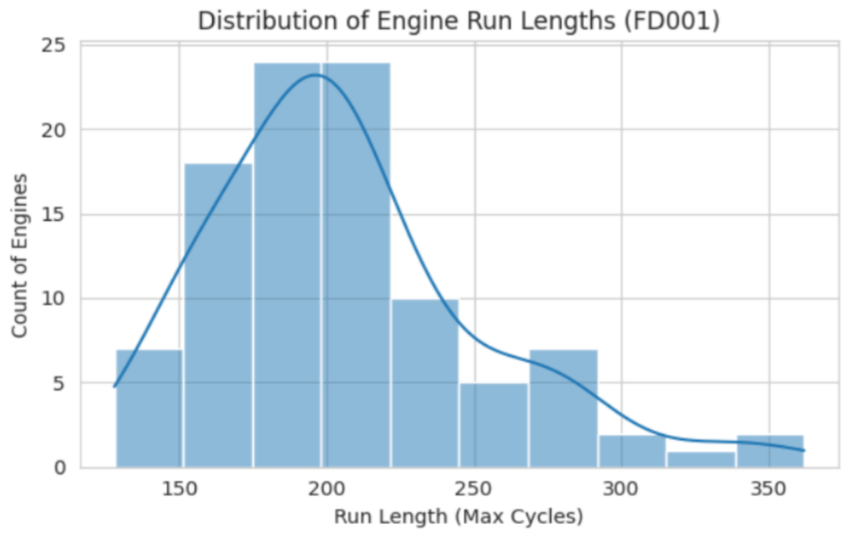

Figure 2. Distribution of engine run lengths in the FD001 subset.

2.5 Summary of Findings. In conclusion, the dataset was found to be clean and complete. After standardization, no anomalous behavior was observed across the sensor readings. Therefore, the data is ready to be modeled.

3. Model Development and Training

3.1 XGBoost Overview. The first model employed is XGBoost, a sequential, iterative tree-based algorithm designed to correct the errors of previous trees at each stage of learning. As shown in Figure 3, given an input pair (x, y), multiple decision trees (weak learners) are trained sequentially, where each tree attempts to improve upon the residuals of the previous one. These weak learners are then ensembled to produce a strong overall prediction for $\hat{y}$. XGBoost is particularly known for its efficiency, scalability, and high predictive performance.

A deeper look at the objective function, shown in Equation 0.2, really just shows that it minimizes a combination of loss (how wrong the predictions are) and regularization (to prevent overfitting), as described in [7]. Mathematically, the objective is expressed as:

$$\text{Obj} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{0.2}$$

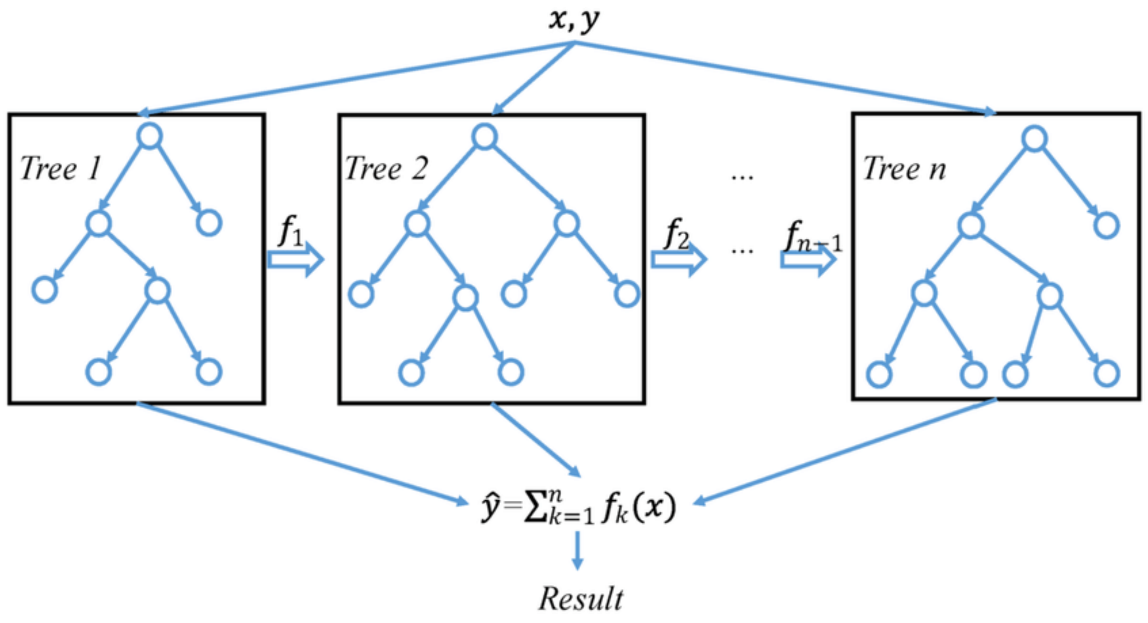

Figure 3. Architecture adapted from [6].

In this formula, y_i is the actual value we are trying to predict, and $\hat{y}_i$ is what the model predicts for the i^{th} data point. The loss function $L(y_i, \hat{y}_i)$ measures how far off those predictions are, summed across all n data points. Each f_k is a decision tree added during training, and $\Omega(f_k)$ is a regularization term that helps keep the model from overfitting by discouraging overly complex trees. Altogether, the model uses K of these trees to make final predictions.

3.2 LSTM Overview. While XGBoost provides a strong baseline, it struggles to capture temporal patterns in the data. To address this, a second model was used, the Long Short-Term Memory (LSTM) network. LSTMs are a special type of Recurrent Neural Network (RNN) designed to learn long-term relationships in sequential data, which makes them a good fit for tracking gradual changes in engine behavior. Unlike traditional RNNs, which can often lose important information over time due to the vanishing gradient problem, LSTMs use a memory cell structure that helps them retain relevant information across many time steps.

Figure 4 shows the three gates that control how the memory cell updates during training. The input gate decides how much new information to add. The forget gate determines which information is no longer needed and removes it. The output gate controls how much of the remaining memory is passed on to the next time step. Each gate uses learnable parameters and activation functions (sigmoid and tanh) to decide, at every step, what to keep and what to drop based on the input sequence.

In this study, the LSTM model was trained using a 40-cycle sliding window, selected to balance historical context and training speed. The model's architecture comprised two stacked LSTM layers (64 units each) followed by a dense output layer. Training ran for

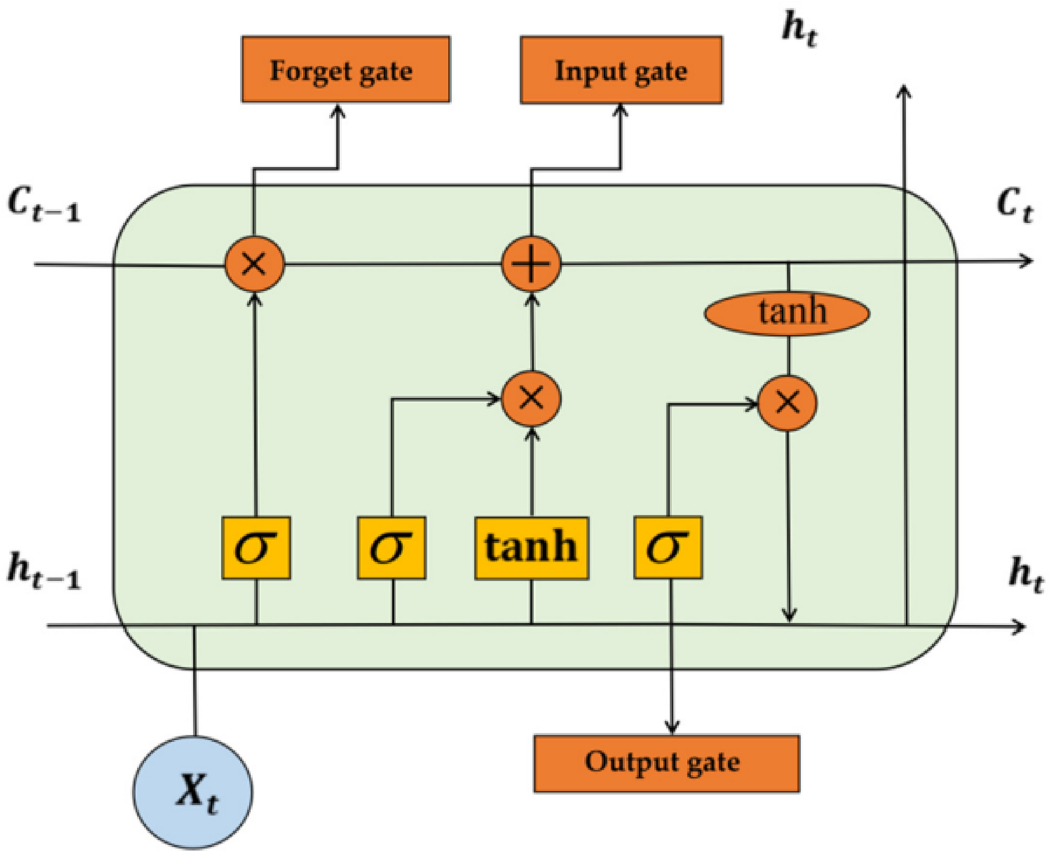

Figure 4. Structure of a typical LSTM cell, showing input, forget, and output gates. Image adapted from [5].

110 epochs using the Adam optimizer (learning rate = 0.001, batch size = 64) and included a 20-epoch warm-up to stabilize gradients and prevent large initial weight updates.

A key distinction in this study is the replacement of the standard mean squared error (MSE) loss with a quantile (pinball) loss function, allowing the model to predict not just a point estimate of RUL but an entire uncertainty band. The loss is defined as follows in equation 0.3:

$$\text{Loss}_q = \max\left(q \cdot (y_{\text{true}} - y_{\text{pred}}),\ (q-1) \cdot (y_{\text{true}} - y_{\text{pred}})\right) \tag{0.3}$$

In equation 0.3, q is the quantile level (such as 0.1 or 0.9), y_{true} is the actual value, and y_{pred} is the predicted value. The loss is asymmetric, meaning it penalizes under-predictions and over-predictions differently depending on the quantile. This allows the model to learn conditional quantiles instead of just the average. As a result, the LSTM can generate 10th and 90th percentile bounds for RUL predictions, which gives more informative and useful outputs for planning maintenance.

3.3 Data Splitting and Validation Strategy. Data leakage between training and test sets is a key concern in time-series analysis. A random 80/20 split may allocate parts of the same engine's timeline to both sets. This overlap inflates performance estimates by exposing test data during training.

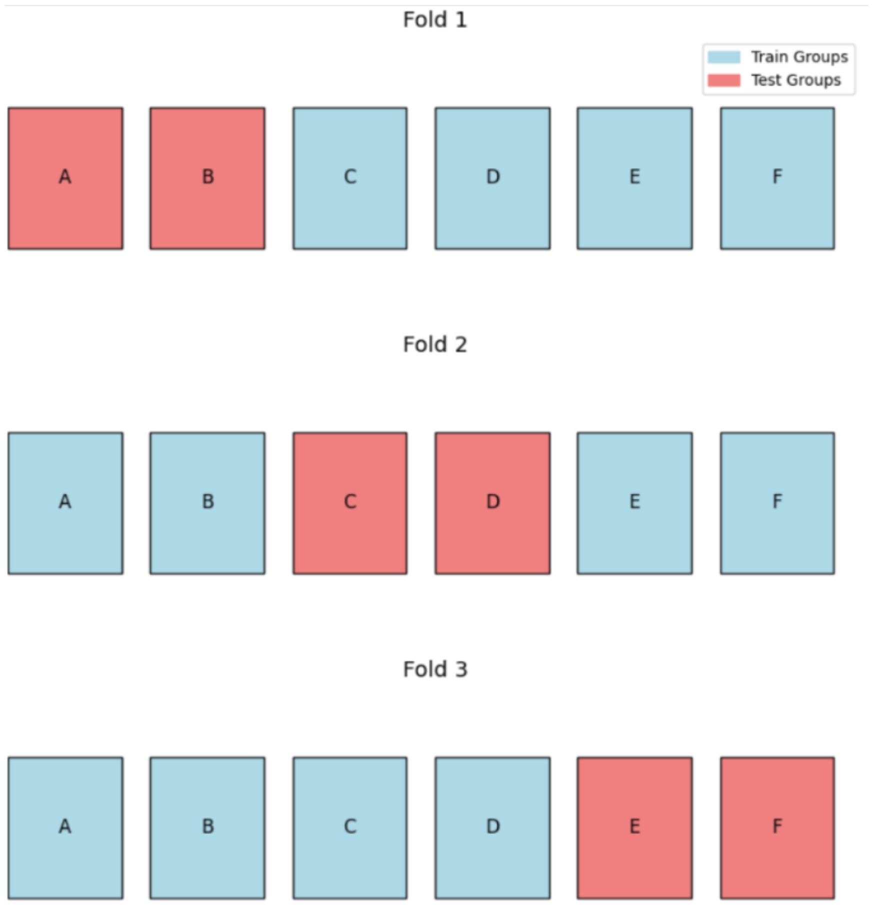

Figure 5. Example of GroupShuffleSplit cross-validation. Each fold keeps complete engine groups in either the training or test set to avoid data leakage.

To avoid data leakage, the model was validated using a GroupShuffleSplit strategy. This approach keeps each engine's full life cycle together and assigns it entirely to either the training or test set. Figure 5 shows how the engines are grouped without being split across sets. The engine groups were shuffled and placed randomly into folds, helping avoid bias and yield more consistent results. Specifically, ten-fold cross-validation was applied, and performance scores were averaged across folds.

4. Model Evaluation and Results

Model performance was measured using Root Mean Squared Error (RMSE) for both the XGBoost and LSTM models. RMSE was chosen because of the ease of comparing results with other studies in predictive maintenance, especially those based on the C-MAPSS dataset. The values are reported in raw engine cycles.

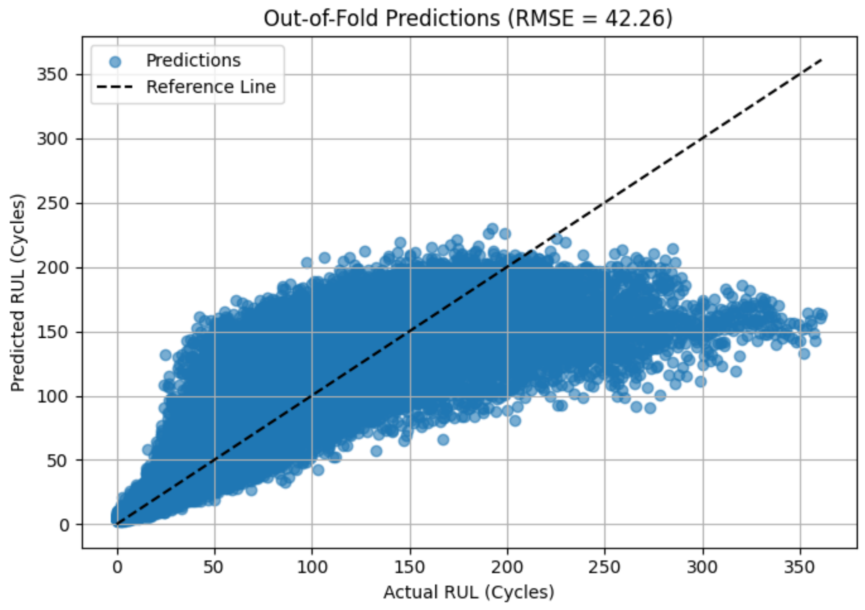

Figure 6. Predicted vs. actual Remaining Useful Life (RUL) for the XGBoost model. The dashed line represents perfect prediction $y = \hat{y}$. The RMSE is 42.26 cycles.

4.1 XGBoost Results. Figure 6 shows the XGBoost model's predicted versus actual RUL. Most predictions follow the reference line $y = \hat{y}$, with a Root Mean Squared Error (RMSE) of 42.26 cycles. However, the plot flattens after 200 cycles, showing that the model struggles more with higher RUL values. This could be due to class imbalance, since fewer engines last beyond 200 cycles and are underrepresented during training, as previously shown in Figure 2. On average, the model achieved a training RMSE of 36.91 cycles and a validation RMSE of 41.81 cycles across the ten folds. Fold-by-fold performance, shown in Figure 7, confirms that the model generalizes well with only small variations across splits.

Figure 8 shows the distribution of out-of-fold prediction errors for the XGBoost model. The peak at zero means that predictions are close to the true value. However, the long tail on the left, with errors reaching as far as -200 cycles, shows a pattern of underprediction. This may be due to the data imbalance mentioned earlier. While conservative predictions are not always a problem and can even be useful in safety-critical settings, they need to match the risk tolerance of the application. In this case, the model leans conservative in order to help prevent unexpected failures.

Finally, Figure 9 shows the SHAP values for the XGBoost model, which highlight how each feature affects the predictions. The most important contributors were Sensor 11, Sensor 9,

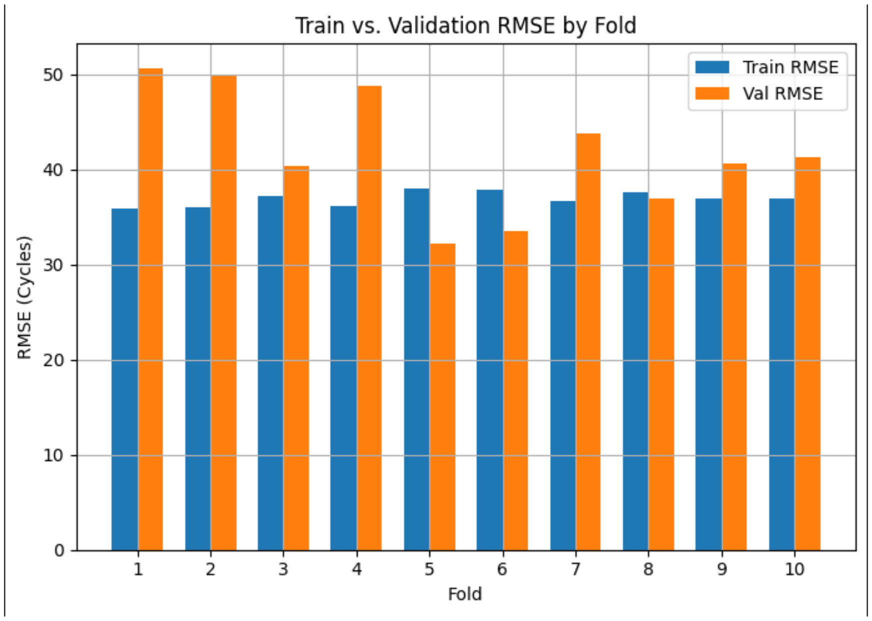

Figure 7. Training and validation RMSE across all ten folds for the XG-Boost model. Fold-level metrics provide insight into consistency and generalization.

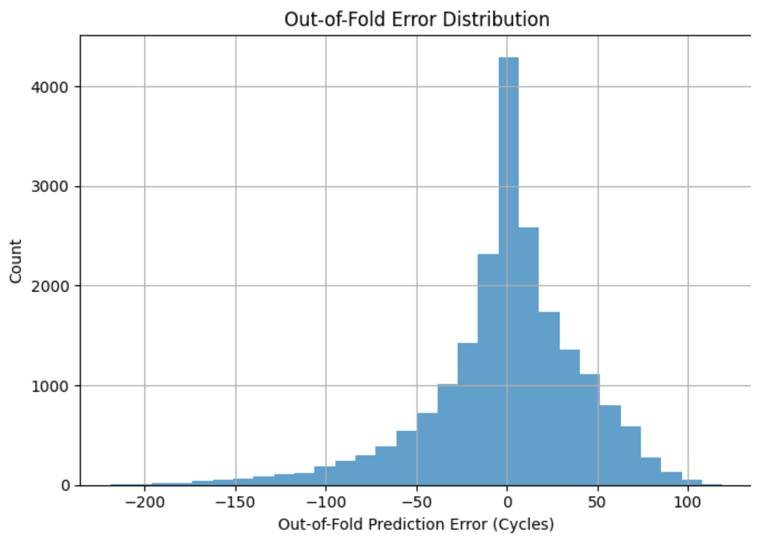

Figure 8. Distribution of out-of-fold prediction errors for the XGBoost model. The histogram shows a peak near zero and a longer left tail, indicating a tendency to underpredict.

and Sensor 14. These features showed a pattern where higher values led to lower predicted RUL. On the other hand, features like Sensor 6 and Operational Setting 3 had little impact on the model's output. This matches what was seen earlier during exploratory analysis.

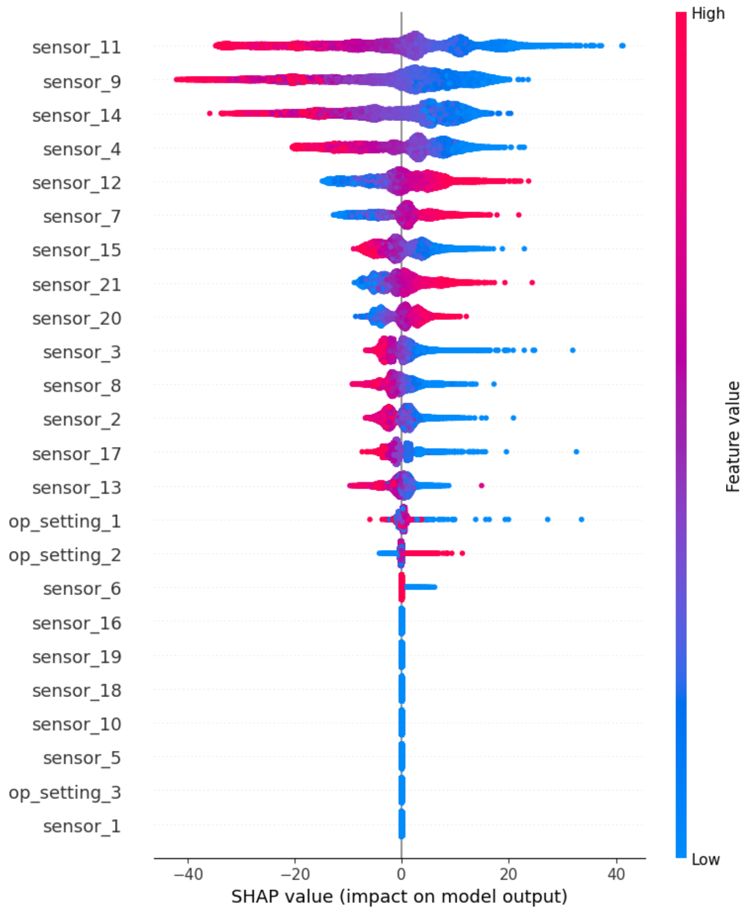

Figure 9. SHAP summary plot showing feature contributions to XGBoost predictions.

4.2 LTSM Results. The LSTM model was trained using a 70-cycle sliding window and a custom quantile loss (pinball loss) to produce both point predictions and prediction intervals. The model demonstrated a significant improvement over XGBoost in terms of RUL prediction accuracy.

As shown in Figure 10, the LSTM model's predictions follow the actual RUL values closely. It reached a cross-validation RMSE of 25.87 cycles and a final out-of-fold RMSE of 20.84 cycles shown in Table 1. The predictions stay close to the reference line, especially in the lower and mid-cycle ranges. The plot stops short of higher cycle values because of how the model is trained. Since the LSTM uses a sliding window of 70 cycles, it can only make predictions where a full window is available. This limits predictions for engines that run much longer and explains why we don't see values extending beyond 200 cycles like we did with XGBoost.

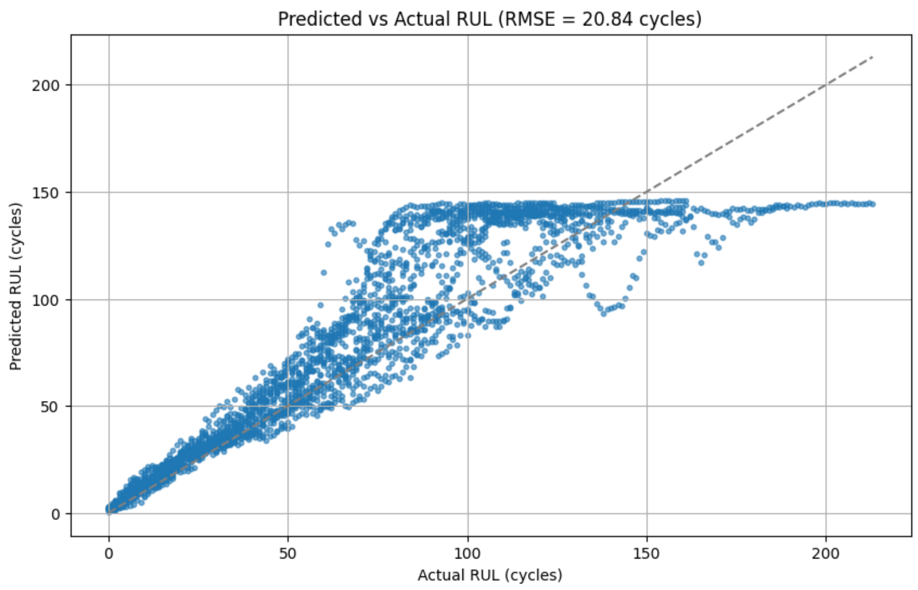

Figure 10. LSTM predicted vs actual RUL. An Out-Of-Fold prediction of RMSE = 20.84 cycles.

Figure 11 shows the out-of-fold prediction error distribution. Most errors are centered around zero and fall within ±25 cycles, which suggests the model stays consistent and performs reliably across different engines.

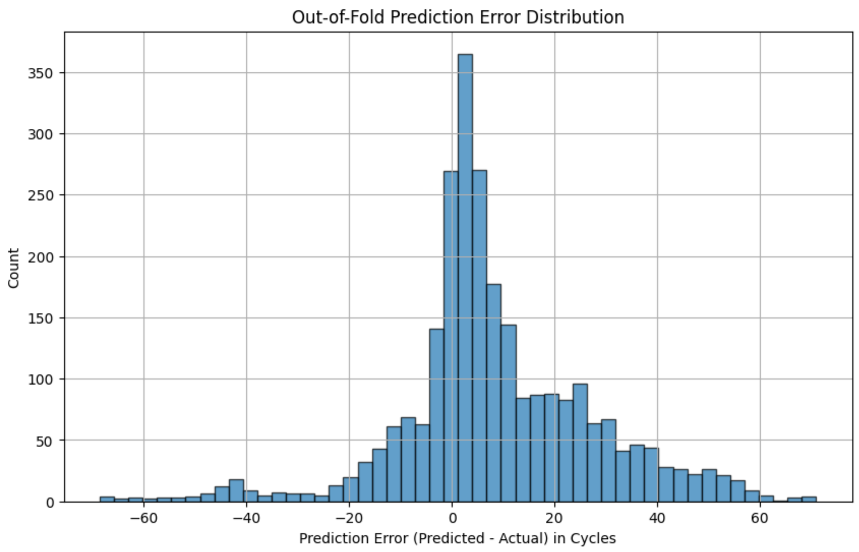

Figure 11. Out-of-fold prediction error distribution for the LSTM model.

In addition to point estimates, the model produced 90 prediction intervals using the 10th and 90th quantiles. As shown in Figure 12, the interval coverage was 86.99%, slightly conservative compared to the nominal 80 target. This over-coverage is often desirable in safety-critical systems, where underestimating risk can be costly.

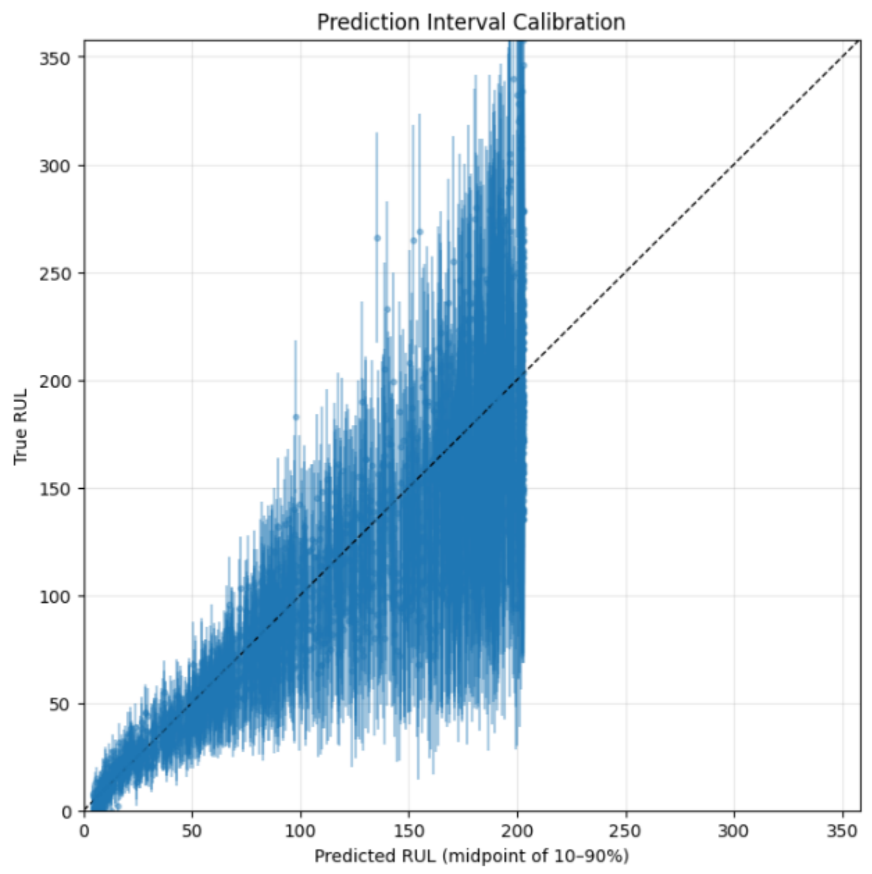

Figure 12. Prediction interval calibration

To better visualize interval behavior across engine lifespans, Figure 13 divides engines into four bins and shows the RUL prediction intervals over time. Each subplot demonstrates that the model tracks the degradation curve accurately while maintaining consistent and reasonably narrow intervals, especially in early-to-mid engine life.

Overall, these results confirm that the LSTM model is not only more accurate than XG-Boost but also provides credible prediction intervals, making it a robust tool for predictive maintenance.

4.3 Comparison Results and Other Works. In this subsection, we compare our models to other state-of-the-art approaches in the field, including CNN–XGB hybrids and attention-based DCNNs; although exact replication of their data and preprocessing pipelines is challenging, this comparison serves to benchmark our work and guide future efforts toward faster and more accurate RUL prediction models.

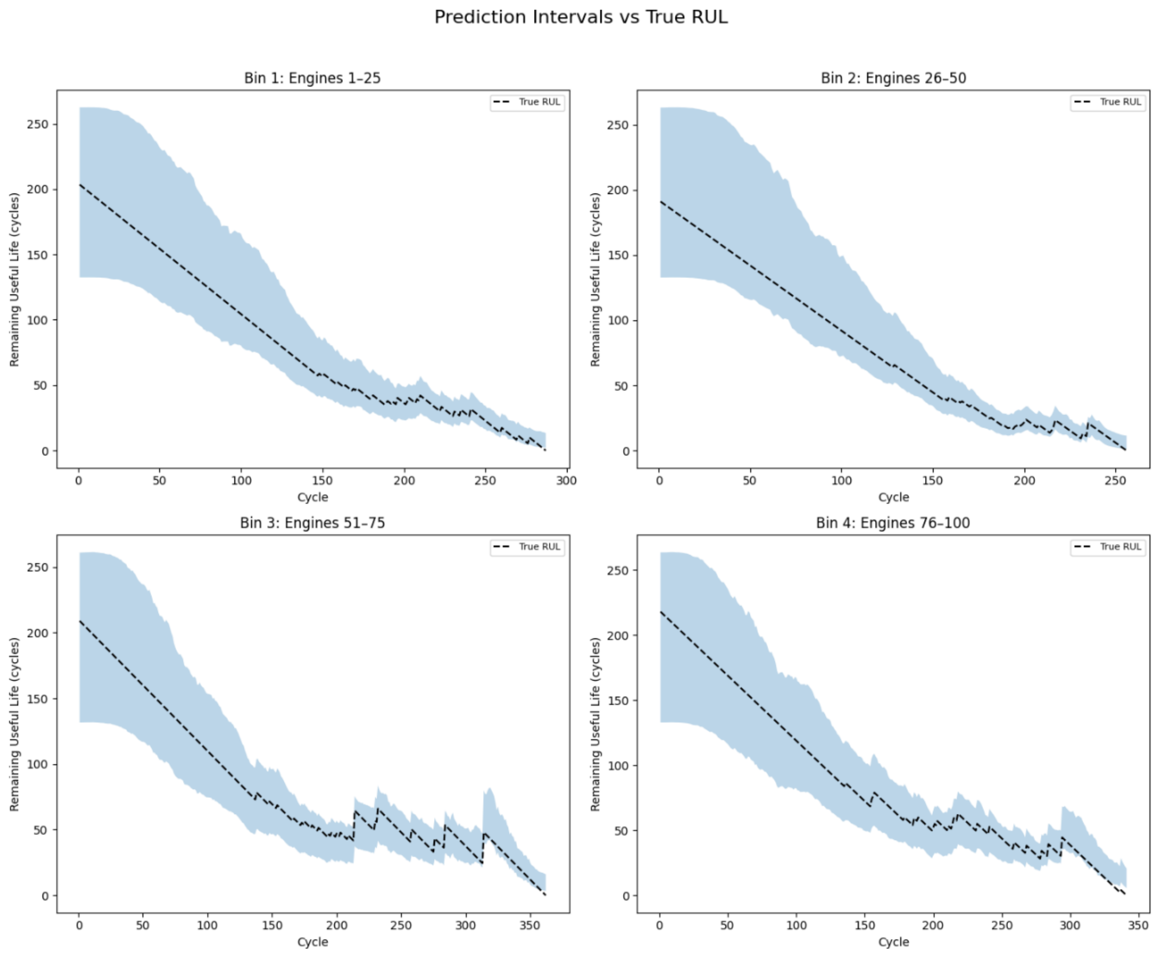

Figure 13. Prediction intervals grouped by engines. The shaded areas represent 10–90% bounds.

Table 1. Cross-Validation RMSE Comparison for LSTM v1 and LSTM v2

Fold	**LSTM v1 RMSE (Cycles)**	**LSTM v2 RMSE (Cycles)**
Fold 1	21.53	17.95
Fold 2	24.81	16.26
Fold 3	25.08	16.05
Fold 4	24.95	18.37
Fold 5	28.71	15.71
Fold 6	28.42	15.64
Fold 7	29.54	16.30
Fold 8	18.02	16.65
Fold 9	36.03	15.35
Fold 10	21.61	16.00
Mean ± Std	**25.87 ± 4.82**	**16.43 ± 0.94**

Table 2. Comparison of LSTM models and related work on FD001 subset

Model	RMSE (Cycles)	Notes	Reference
LSTM v1	20.84	No RUL cap	This study
LSTM v2	16.47	RUL cap at 150 cycles	This study
CNN-XGB	12.61	RUL capped at 125 cycles	[8]
XGBoost	42.26	Baseline model, Optuna	This study
Deep LSTM	16.14	RUL cap at 150 cycles	[9]

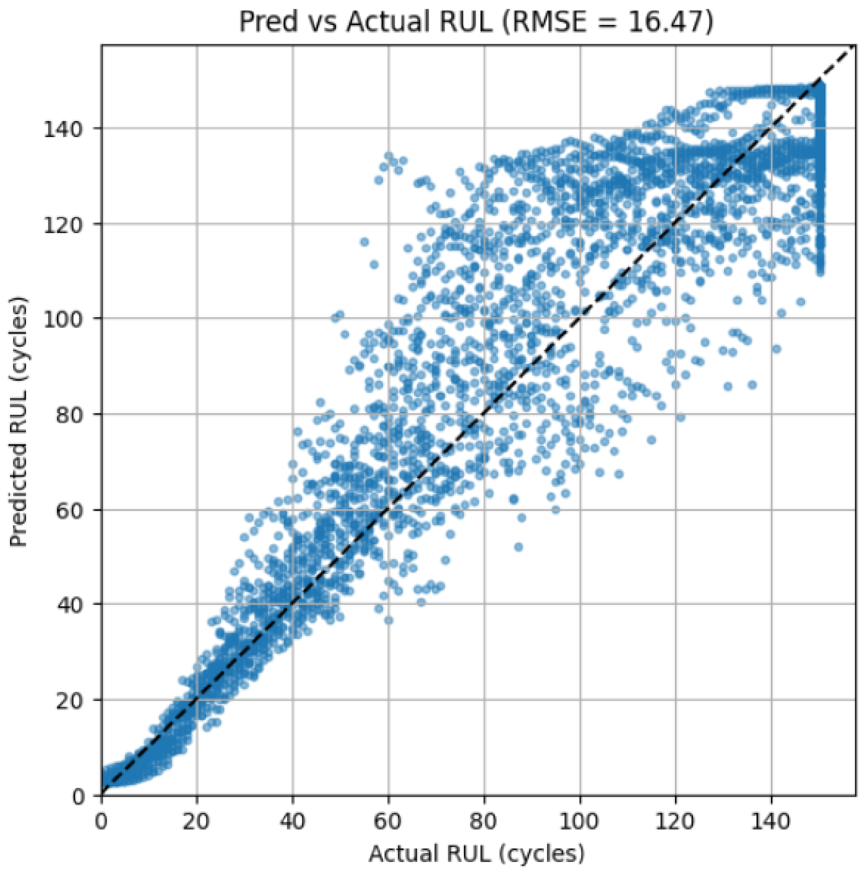

Figure 14. LSTM v2 predictions versus actual RUL for the FD001 subset. The model achieves an RMSE of 16.47 cycles. Note how many points level off near the top right. This reflects the 150-cycle cap imposed during windowed training, which prevents predictions beyond that threshold.

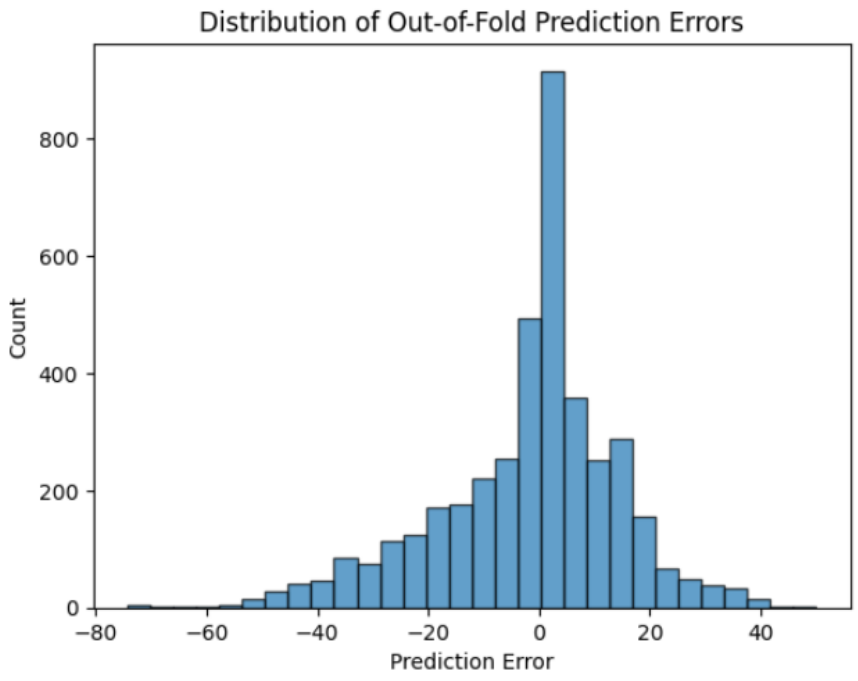

Figure 15. Out-of-fold prediction error distribution for LSTM v2. Errors are centered near zero and most lie within ±25 cycles, indicating consistent performance across engines.

Building on these benchmarks, the full breadth of each method could not be implemented and validated due to time constraints; however, capping RUL at 150 cycles was incorporated to form the basis of LTSM v2, while min max scaling and feature selection methods remain for future work. Table 2 shows how LSTM v2 compares with the original LSTM v1 and other published methods. Capping RUL at 150 cycles reduced RMSE to 16.47 cycles (a 20% improvement over v1). The XGBoost baseline reached an RMSE of 42.26 cycles but trained faster and with less tuning while lacking uncertainty quantification. Zhang [8] achieved RMSEs under 12.61 cycles using a CNN-XGBoost hybrid with extensive preprocessing and feature extraction, and Heimes [10] used an attention based DCNN for similar results. Other deep LSTM approaches add denoising or feature fusion and use custom caps for each subset. [9, 8, 2].

Despite these high-performing hybrids, XGBoost remains a practical option for real-world deployments. It trains quickly, requires minimal preprocessing, and its predictions can be readily explained with techniques like SHAP. In contrast, LSTM models need more tuning and longer training times, but their higher accuracy and built-in uncertainty estimates make them better suited to high-reliability maintenance settings. The LSTM architecture offers a solid balance of performance and interpretability and is intended to serve as a foundation for future work in predictive maintenance.

5. Conclusion and Future Work

This study compared XGBoost and an LSTM network for predicting RUL on the C-MAPSS dataset. XGBoost proved fast and easy to interpret, but it struggled with capturing long-term trends, which is where the LSTM excelled.

By training the LSTM with a custom quantile loss, we cut the RMSE nearly in half compared to XGBoost and produced 10–90% prediction intervals that covered 86.99% of true values. These intervals add a layer of uncertainty information that can guide maintenance scheduling and risk management.

Preprocessing was kept simple, so there is room to explore noise reduction, alternate scaling techniques, or other tweaks that might improve performance even further. It is expected that with those refinements, this LSTM setup could rival or exceed more complex hybrid models in the literature.

Looking ahead, the model will be tested on the other C-MAPSS subsets (FD002–FD004) to assess generalizability. SHAP-based feature analysis will be added to the LSTM for a fairer comparison with XGBoost's explainability. Finally, combining the LSTM's temporal strengths with a tree-based method in a single hybrid model could deliver both accuracy and speed, as demonstrated by Zhang et al. [8].

Overall, this work offers a straightforward, reliable approach to RUL prediction that balances performance, uncertainty quantification, and practical deployment needs in predictive maintenance.

REFERENCES

[1] Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 International Conference on Prognostics and Health Management*, pages 1–9. IEEE, 2008.

[2] Amgad Muneer, Shakirah Mohd Taib, Suliman Mohamed Fati, and Hitham Alhussian. Deep-learning based prognosis approach for remaining useful life prediction of turbofan engine. *Symmetry*, 13(10):1861, 2021.

[3] Wenjin Zhang, Dan Yang, and Hongyu Wang. Data-driven methods for predictive maintenance of industrial equipment: A survey. *IEEE Systems Journal*, 13(3):2213–2227, 2019.

[4] L. F. Matorra, A. M. Scholtens, S. Traverso, E. De Vrey, F. Funes, R. Aguero, S. Cusimano, B. Ruijsink, R. Geronazzo, M. Namias, and L. E. Juarez-Orozco. Feature importance explanation of prosthetic valve endocarditis through machine learning via shap. *European Heart Journal*, 45(Supplement_1), 2024.

[5] Xiaojia Wang, Ting Huang, Keyu Zhu, and Xibin Zhao. Lstm-based broad learning system for remaining useful life prediction. *Mathematics*, 10(12):2066, 2022.

[6] Yuanchao Wang, Z. Pan, J. Zheng, L. Qian, and Mingtao Li. A hybrid ensemble method for pulsar candidate classification. *Astrophysics and Space Science*, 364, August 2019.

[7] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–794. ACM, 2016.

[8] Xiaoyong Zhang, Pengcheng Xiao, Yingze Yang, Yijun Cheng, Bin Chen, Dianzhu Gao, Weirong Liu, and Zhiwu Huang. Remaining useful life estimation using cnn-xgb with extended time window. *IEEE Access*, 7:154683–154695, 2019.

[9] Shuai Zheng, Krsto Ristovski, Ahmed Farahat, and Chetan Gupta. Long short-term memory network for remaining useful life estimation. In *Proceedings of the 2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 88–95, Dallas, TX, USA, June 2017. IEEE.

[10] F. O. Heimes. Recurrent neural networks for remaining useful life estimation. In *Proceedings of the 2008 International Conference on Prognostics and Health Management*, pages 1–6, Denver, CO, USA, October 2008. IEEE.