A Comparative Study of Manual Machine Learning and AutoML Approaches for Predicting Remaining Useful Life in NASA's Turbofan Engine Degradation

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Shuhui(Melinda),Yu

December, 2023

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

Dec 11, 2023

Advisor: Dr. Michael Soltys Date

Dr. Jason Isaacs Date

Dr. Kevin Scrivnor Date

APPROVED FOR THE UNIVERSITY

Dr. Joe Shapiro Date

A Comparative Study of Manual Machine Learning and AutoML Approaches for Predicting Remaining Useful Life in NASA's Turbofan Engine Degradation

by
Shuhui (Melinda), Yu

Computer Science Program
California State University Channel Islands

Abstract

Predicting the remaining useful life (RUL) of machinery, such as NASA's Turbofan engines, is crucial for maintenance and reliability. Traditionally, this task has relied on manual machine learning (ML) approaches that require domain expertise and extensive feature engineering. However, the emergence of AutoML (Automated Machine Learning) has promised to simplify this process, making it accessible to non-experts by automating tasks like model selection, hyper-parameter tuning, and feature engineering. This study conducts a comprehensive comparative analysis between traditional manual ML and AutoML approaches for RUL prediction in Turbofan engines. The Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset is used to evaluate the performance of both methods. Various factors such as predictive accuracy, model interpretability, and ease of use are considered. The results of this study provide valuable insights into the strengths and weaknesses of manual ML and AutoML in the context of RUL prediction. This research contributes to a deeper understanding of the applicability of AutoML in critical domains like aerospace engineering and highlights when each approach may be preferable. Ultimately, the study indicates that the manual LSTM model has an accuracy advantage over the automated machine model, by approximately 26%. However, the total time spent on manual models is hundreds of times greater than AutoGluon.

CONTENT

Figures

1. INTRODUCTION

The remaining useful life (RUL) is used to describe the progression of faults in prognostics and health management (PHM) applications [1]. RUL prediction is a crucial aspect of turbofan engine maintenance, turbofan engines are used in aircraft and other aerospace applications and are subjected to harsh operating conditions. According to the National Transportation Safety Board (NTSB), data from 1981 to 2001 reveals that 36% of all mechanical failures in the United States' 7,571 aircraft accidents were attributed to propulsion system issues [2]. Within the civil aviation sector, airlines allocate approximately 30 billion dollars annually to aircraft maintenance, with engine maintenance alone constituting roughly 31% of this expenditure [3]. According to a report from the International Air Transport Association (IATA), there were 39 total aviation accidents in 2022, and there were five tragic incidents resulting in fatal accidents that led to the loss of passengers, and crew members' lives. The number of fatalities rose from 121 in 2021 to 158 in 2022 [4]. The degradation of engine components is a natural phenomenon that occurs over time, and it is critical to predict the remaining useful life of these components accurately to avoid catastrophic failures and ensure optimal performance. RUL prediction for turbofan engine degradation involves using advanced analytical methods and machine learning techniques to analyze engine data and predict the remaining useful life of various engine components. This can include analyzing data from sensors that measure engine temperature, pressure, and vibration, as well as data on the engine's operating conditions, such as altitude and airspeed. The primary goal of RUL prediction is to provide maintenance teams with

accurate and timely information on the remaining useful life of engine components. This allows them to plan predictive maintenance (PdM) activities for condition-based monitoring (CBM) more effectively [5], reduce unscheduled downtime, and optimize engine performance. In addition, RUL prediction can help airlines and other operators save money by substantially reducing maintenance costs and avoiding costly engine replacements [6].

Estimating RUL poses a formidable challenge. RUL is not a straightforward target variable derived from sensor data; instead, it requires inferring trends in degradation patterns over an extended period. The primary challenges in this task revolve around data preprocessing and defining the target RUL variable for effective machine learning model training. Additionally, choosing the right learning algorithm is a complex decision given the multitude of available options. This selection often assumes that the researcher or end-user possesses the requisite knowledge, ability, and time for such choices. The algorithm's design and hyper-parameter decisions, alongside those made during data preprocessing, create a daunting task for end-users. Consequently, many opt for pre-selecting an algorithm or limiting themselves to a predefined list during initial experiments. This approach may inadvertently lead to the neglect of potentially suitable learning schemes, hindering the discovery of more appropriate algorithms for addressing the problem effectively.

Recent research has made significant strides in estimating the RUL of turbofan engines through the application of deep learning techniques, including convolutional neural networks (CNNs), long short-term memory (LSTMs) networks, and their various combinations and modifications. Significantly, LSTM networks have demonstrated superior performance when

compared to CNN-based models [7], [8]. LSTM's effectiveness stems from their suitability for handling time-series data, their capacity to capture temporal features in multivariate systems, and their ability to minimize the root mean square error (RMSE) concerning target predictions. This paper proposes an LSTM-based model for predicting the RUL of turbofan engines, capitalizing on these advantages.

Moreover, this study also introduces an automated machine learning (AutoML) predicting method to estimate RUL. AutoML simplifies the complex task of RUL prediction by automating many of the traditionally labor-intensive and technical aspects of machine learning. AutoGluon is an open-source AutoML framework developed by Amazon Web Services (AWS). In two well-known Kaggle competitions, AutoGluon outperformed 99% of the participating data scientists with just 4 hours of training on the raw data [9].

The contributions in this paper are as follows:

A. An LSTM-based model with effective preprocessing steps, i.e. correlation analysis with data normalization and filtering is proposed.

B. A method for estimating the RUL, based on the use of AutoGluon which can automatically generate a suitable pipeline.

C. The comparison of LSTM-based model prediction method and novel AutoGluon prediction method.

2. BACKGROUND

A RUL prediction model is a computational model or algorithm that is designed to estimate the remaining operational lifespan of a machine, system, or component. This model leverages historical data, sensor measurements, and other relevant information to make predictions about when a particular asset is likely to fail or no longer meet its performance requirements. RUL prediction models are commonly used in predictive maintenance and reliability engineering to optimize maintenance schedules, reduce downtime, and prevent unexpected equipment failures. These models can take various forms and employ different techniques, including physics-based models [10], data-driven models [11], and hybrid models [12].

2.1 Physics-based Technique

Conventional model-based approaches typically utilize algorithms such as the Kalman filter (KF), extended Kalman filter (EKF), and particle filters to develop mathematical representations of machines based on multi-sensor time series data sequences [13] [14]. The model-based approach begins with a thorough comprehension of the machine's physical structure, followed by the application of physical principles to derive a mathematical model for estimating RUL [15]. Moreover, the corrosion model [16], abrasion model [17], and Taylor tool

wear model [18] are examples of model-based techniques. A physics-based approach is suitable when a reliable degradation model, like fatigue crack growth, is available [19] [20]. However, it demands substantial prior knowledge or measurement data and necessitates the consideration of noise effects [21].

2.2 Data-driven Technique

Data-driven techniques leverage historical sensor data and machine learning algorithms to build predictive models. These models learn from the patterns and trends observed in the data without explicit knowledge of the underlying physical processes. The data-driven technique is both rapid and straightforward to implement, with the potential to reveal previously unnoticed relationships. Polynomials are often used in data-driven approaches for modeling because they offer a flexible and adaptable way to represent relationships between variables. When patterns in the data suggest non-linear relationships, polynomial functions can be employed to capture these non-linearities more accurately. Nonetheless, it requires a judicious approach as there is a risk of over-learning and overgeneralization [22]. Also, regression analysis, neural networks, Bayesian, random forest (RF), support vector machine (SVM), and relevance vector machine (RVM) are examples of data-driven methodologies [23]. Given the numerous variations in today's supervisory control and data acquisition (SCADA) data, it becomes crucial to remove any extraneous or unnecessary information [24].

2.3 Hybrid Technique

The hybrid approach combines a physical-based approach and a data-based approach [25] [26]. The prediction accuracy of the hybrid approach is improved by combining the polynomial advantages obtained from the data-driven method with the low error afforded by the physics-based method's degradation model [27]. The data is employed for training model parameters, and expertise in the physical processes guides the selection of the appropriate regression analysis method to apply such as linear, polynomial, exponential, etc.. Examples of hybrid techniques include particle filters, Kalman filters, and others [28]. On the other hand, it requires a comprehensive grasp of the system along with the acquisition of pertinent data [29].

2.4 Machine Learning

Over the past decade, researchers have increasingly turned to data-driven prognostic methods, which are designed to estimate the RUL by scrutinizing degradation trends and sensor data trajectories. In recent years, artificial neural networks (ANNs) [30], particularly deep neural networks (DNNs), have emerged as effective tools for achieving highly accurate RUL predictions, particularly for nonlinear and intricate systems. A DNN is a type of ANN characterized by numerous hidden layers between the input and output layers. Ongoing research efforts have aimed to enhance prediction results through various algorithms. Among machine

learning algorithms used for turbofan engine RUL prediction, comparisons have been made between the multilayer perceptron (MLP), support vector regression (SVR), relevance vector regression (RVR), and the more recent CNN algorithm. Notably, the CNN algorithm demonstrated superior accuracy [30], with further improvements achieved by constructing a deep CNN model incorporating five CNN layers [31]. Concurrently, in the realm of recurrent neural networks (RNNs), the long short-term memory (LSTM) algorithm has been employed to address the gradient descent problem that arises as the depth of the network increases during training [32]. In comparative evaluations, LSTM demonstrated superior performance when contrasted with alternative methods, including MLP, SVR, RVR, and CNN [33] because of LSTM models' suitability for handling time-series data, their ability to capture temporal patterns in multivariate systems, and their capability to minimize the root mean square error (RMSE) in relation to target predictions.

Therefore, in the context of this study, an LSTM-based model is proposed for predicting the RUL of a turbofan engine. While LSTM networks can effectively learn the relationships between target RUL values and sensor data, they do face certain limitations. These limitations include the presence of outliers, sensor data noise, unnormalized data, and uncorrelated sensor values, all of which can impact the performance of an LSTM network [34]. This paper focuses on addressing these challenges and implementing solutions to enhance predictive accuracy.

2.5 Automated Machine Learning

AutoML is a transformative approach that streamlines the machine learning model development process by automating various stages, from data preparation and feature engineering to model selection and hyper-parameter tuning [35]. It aims to make machine learning more accessible and efficient for both seasoned data scientists and those with limited expertise in the field. AutoML platforms and tools typically provide user-friendly interfaces that simplify complex machine learning tasks, enabling users to build high-performing models with minimal manual intervention. These tools can automatically preprocess data, handle missing values, select relevant features, choose appropriate algorithms, and fine-tune hyper-parameters. By automating these tasks, AutoML accelerates the model development cycle, reduces the barrier to entry for machine learning adoption, and democratizes artificial intelligence (AI) by allowing organizations to harness the power of machine learning for various applications, from predictive analytics to computer vision and natural language processing.

AutoGluon stands out as an open-source AutoML framework, introduced by Erickson et al. in 2020 [9]. Compared to traditional AutoML or artificial intelligence platforms, AutoGluon boasts a remarkable advantage: it excludes the need for extensive manual work in model selection and hyper-parameter optimization processes that conventional AutoML platforms demand [36]. Unlike traditional machine learning, which typically relies on a single model for training, AutoGluon adopts a distinctive approach by amalgamating results from multiple models [37]. Empirical evidence from data prediction projects demonstrates that AutoGluon outperforms

individual training models during its training phase. Moreover, AutoGluon exhibits remarkable predictive accuracy even when working with raw, untreated databases [38]. AutoGluon effortlessly handles a wide array of structured data types. If the predefined training model doesn't align with the original dataset, AutoGluon autonomously resolves the issue, eliminating the need for human intervention or choices. Furthermore, it offers high-level data preprocessing, supports deep learning and multi-layer model integration, and automatically categorizes data in each column, including specialized handling of text fields. AutoGluon also optimizes AutoML processes, such as intelligent model network design. Users have the flexibility to pause or resume the training process, enabling them to set the training duration as needed and receive timely results for adjustments based on their requirements.

In summary, this innovative AutoGluon AutoML platform streamlines the intricate AutoML workflow, reducing the technical barriers for users and making AutoML methods more accessible to a broader audience. Users can execute AutoML processes without grappling with complex coding; the implementation of AutoML algorithm functions can be achieved in just a few lines of code, making advanced algorithmic tools more user-friendly [39]. In [40], AutoGluon brings advantages such as the ability to identify non-linear associations among variables and the inclusion of a larger number of variables in the model-building process. Hence, the other main point in this study is using AutoGluon to predict RUL for NASA's turbofan engine.

3. DATASET DESCRIPTION

The Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) [41] dataset was developed by NASA Ames Prognostics Center of Excellence [42] and contains simulated data of turbofan engines under different operating conditions and fault scenarios.

3.1 The Turbofan Engine

The turbofan engine comprises several key elements, including the nozzle, fan, high-pressure chamber (HPC), high-pressure turbine (HPT), low-pressure turbine (LPT), and low-pressure chamber (LPC) as shown in Figure-1.

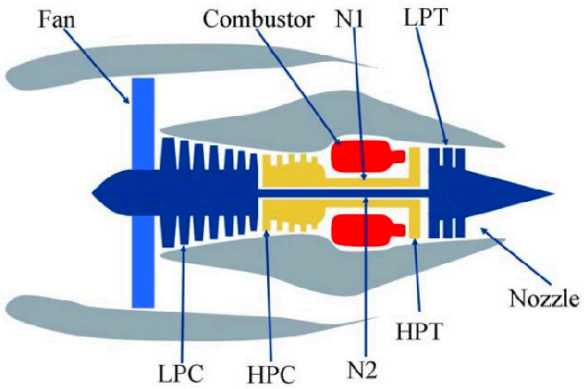

Figure-1: Structure of turbofan engine[41].

3.1.1 The Cost of Turbofan Engine

Turbofan engines come in various sizes and complexities, each tailored to specific aircraft needs and passenger capacities. Small turbofan engines, found in business jets like the Gulfstream G650, typically cost between $1.5 and $3.5 million per unit. These engines are designed for aircraft of this size, accommodating 18-28 passengers, with a primary focus on fuel efficiency [43].

Moving up in size and complexity, medium-sized turbofan engines, exemplified by models like the Pratt & Whitney PW1500G and CFM International LEAP, are built for 100-200 passenger capacity aircraft such as the Airbus A320 and Boeing 737. The cost of these engines falls in the range of $10 to $15 million. Large turbofan engines, responsible for powering some of the world's largest airliners like the Boeing 777, Airbus A350, and Airbus A380, are significantly larger and more intricate in design. These engines, with their substantial power requirements, can come with a price tag ranging from $25 to $45 million [43].

Regarding the maintenance expenses for aircraft engines: Smaller business jet engines usually incur overhaul costs in the range of $200,000 to $300,000 each. On the contrary, overhauling modern large business jet engines can amount to approximately $1 million to $3 million each [44].

3.1.2 The Function of Turbofan Engine

While in flight, passengers are only able to observe a sizable fan positioned at the front and a relatively small exhaust pipe situated at the rear of a jet engine in Figure-2. However, there exists a substantial interplay of components between these two visible parts. The primary constituents of a turbofan engine encompass the fan blades, a compressor section, the combustion chamber, turbines, and the exhaust are given in Figure-3.

A turbofan engine works in four simple steps: suck, squeeze, bang, and blow. At the outset, a large fan draws air into the engine. This incoming air, traveling at high velocity, proceeds to the second stage, where it undergoes compression through a sequence of low-pressure and high-pressure compressor blades. At this stage, the air has become approximately 40 times denser than usual, and temperatures have risen to several hundred degrees. The highly compressed air proceeds into the combustion chamber, where fuel is sprayed to create a mixture. This mixture is then ignited, causing the gases to rapidly expand, ultimately exiting through the exhaust nozzles. In this scenario, the high-velocity exhaust gases exiting the engine generate thrust, which propels the aircraft forward with an equal and opposite force [45].

Figure-2. Outward of the turbofan engine [43].

Figure-3. Internal structure of the turbofan engine[46].

3.2 The C-MAPSS Dataset

The C-MAPSS dataset consists of multiple sub-datasets containing FD001, FD002, FD003, and FD004, each corresponding to a different combination of operating conditions and fault modes. These sub-datasets provide diverse scenarios for testing predictive maintenance models The datasets encompass multiple multivariate time series, with each dataset further partitioned into training and test subsets. A "cycle" generally refers to a specific operational period or a unit of time during which data is collected from a particular engine. These cycles could represent different phases of an engine's operation, such as takeoff, cruising, or landing, depending on the dataset's design and the specifics of the simulated engine systems. Each time series corresponds to a different engine, collectively representing a fleet of engines of the same type. Notably, these engines initiate their journeys with varying degrees of initial wear and undisclosed manufacturing variations, which are deemed normal and non-fault conditions. In addition, the dataset incorporates three operational settings that exert significant influence over engine performance, all of which are included in the data. It's important to note that the data is affected by sensor noise.

At the outset of each time series, the engine operates normally, eventually developing a fault at some point during the series. In the training set, the fault escalates in severity until system failure occurs. In contrast, the time series in the test set concludes before reaching system failure. The primary objective of the approach is to predict the number of remaining operational

cycles before failure in the test set, specifically the number of cycles that the engine will continue to operate after the final observed cycle. Additionally, a vector of true RUL values is provided for the test data, serving as a reference for evaluation. The dataset has played a significant role in advancing the field of predictive maintenance and RUL prediction, helping researchers develop and validate models for early fault detection and optimizing maintenance schedules in aviation.

Moreover, Figure-5 provides information about 21 sensors, including their specifications. The last column of Figure-5, labeled as "Trends," indicates the descriptions of the sensor data degradation patterns over time. In this context, "~" signifies irregular sensor behavior, " ↑ " denotes an increasing parameter trend with time, and " ↓ " signifies a decreasing parameter trend with time [47].

Data Set	Train Trajectories	Test Trajectories	Conditions	Fault Modes
FD001	100	100	1 (Sea Level)	1 (HPC Degradation)
FD002	260	259	6 (Fault)	1 (HPC Degradation)
FD003	100	100	1 (Sea Level)	2 (HPC Degradation, Fan Degradation)
FD004	248	249	6 (Fault)	2 (HPC Degradation, Fan Degradation)

Figure-4. The subsets of the C-MAPSS dataset [41].

Sensor Number	Symbol	Description	Units	Trend
1	T2	Total temperature at fan inlet	°R	~
2	T24	Total temperature at LPC outlet	°R	↑
3	T30	Total temperature at HPC outlet	°R	↑
4	T50	Total temperature at LPT outlet	°R	↑
5	P2	Pressure at fan inlet	psia	~
6	P15	Total pressure in bypass-duct	psia	~
7	P30	Total pressure at HPC outlet	psia	↓
8	Nf	Physical fan speed	rpm	↑
9	Nc	Physical core speed	rpm	↑
10	epr	Engine pressure ratio	–	~
11	Ps30	Static pressure at HPC outlet	psia	↑
12	Phi	Ratio of fuel flow to Ps30	pps/psi	↓
13	NRf	Corrected fan speed	rpm	↑
14	NRc	Corrected core speed	rpm	↓
15	BPR	Bypass ratio	–	↑
16	farB	Burner fuel-air ratio	–	~
17	htBleed	Bleed enthalpy	–	↑
18	Nf_dmd	Demanded fan speed	rpm	~
19	PCNfR_dmd	Demanded corrected fan speed	rpm	~
20	W31	HPT coolant bleed	lbm/s	↓
21	W32	LPT coolant bleed	lbm/s	↓

Figure-5. List of sensors [48].

4. PROPOSED METHODOLOGY

4.1 : Data Preprocessing

Data preprocessing is an essential step in the data analysis and machine learning workflow, with its specific methods and techniques tailored to the data's nature and analysis objectives. Effective data preprocessing enhances the accuracy and significance of insights and model predictions. In the model training phase, the original turbofan engine data undergoes preprocessing to generate the necessary model parameters. This preprocessing encompasses defining the RUL labels for both the training and test sets [Figure-6], finding the maximum cycle [Figure-7], data splitting [Figure-8], feature selection [Figure-9], and data standardization [Figure-10]. The FD001 dataset comprises 21 sensor features and 3 operational parameters (flight altitude, Mach number, and throttling parser angle). Moreover, the number of running cycles is considered as a feature, resulting in a total of 25 features.

```
def add_rul(g):
    g['RUL'] = max(g['cycle']) - g['cycle']
    return g

train = train_data.groupby('id').apply(add_rul)
```

Figure-6. Adding RUL label function.

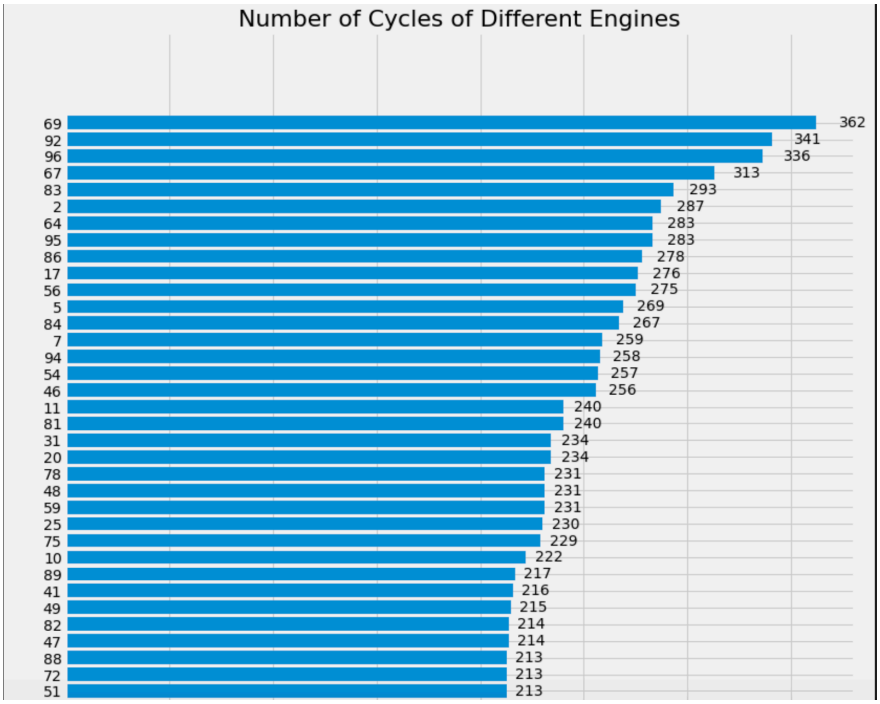

Figure-7. Number of cycles of different engines, where the x-axis represents the number of cycles., and the y-axis represents the engine ID.

```
[229]
    processed_train_data, processed_val_data, processed_train_targets, processed_val_targets = train_test_split(processed_train_data,
                                                                                                                 processed_train_targets,
                                                                                                                 test_size = 0.2,
                                                                                                                 random_state = 83)

    print("Processed train data shape: ", processed_train_data.shape)
    print("Processed validation data shape: ", processed_val_data.shape)
    print("Processed train targets shape: ", processed_train_targets.shape)
    print("Processed validation targets shape: ", processed_val_targets.shape)

...  Processed train data shape:  (14184, 30, 15)
     Processed validation data shape:  (3547, 30, 15)
     Processed train targets shape:  (14184,)
     Processed validation targets shape:  (3547,)
```

Figure-8. Data splitting.

4.1.1 : Correlation Analysis

Well-engineered features can help machine learning models discover relevant patterns and relationships in the data, leading to improved predictive accuracy. Feature engineering can simplify complex datasets by reducing dimensionality [48], focusing on relevant information, and removing noise. Furthermore, feature engineering can enhance the interpretability of models by creating features that align with domain knowledge or human-understandable concepts. Careful feature selection and engineering can reduce the risk of overfitting by removing noisy or irrelevant features that may cause the model to generalize poorly.

Correlation analysis is a statistical technique used to evaluate the strength and direction of the linear relationship between two or more variables. It helps in understanding how changes in one variable are associated with changes in another [49]. Pearson's correlation coefficient analysis is a representative filter method for determining removal from a dataset based on the degree of correlation [50]. The formula for Pearson's correlation coefficient is:

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{2}$$

where r represents Pearson's correlation coefficient. X_i and Y_i are the individual data points for the two variables. $\bar{X}$ and $\bar{Y}$ are the means (averages) of the two variables. The summation ($\sum$) is taken over all data points i from 1 to n, where n is the total number of data points. The formula

calculates the covariance between X and Y divided by the product of their standard deviations. Also, $r = 1$ represents perfect positive correlation, $r = -1$ represents perfect negative correlation, and $r = 0$ represents no linear correlation.

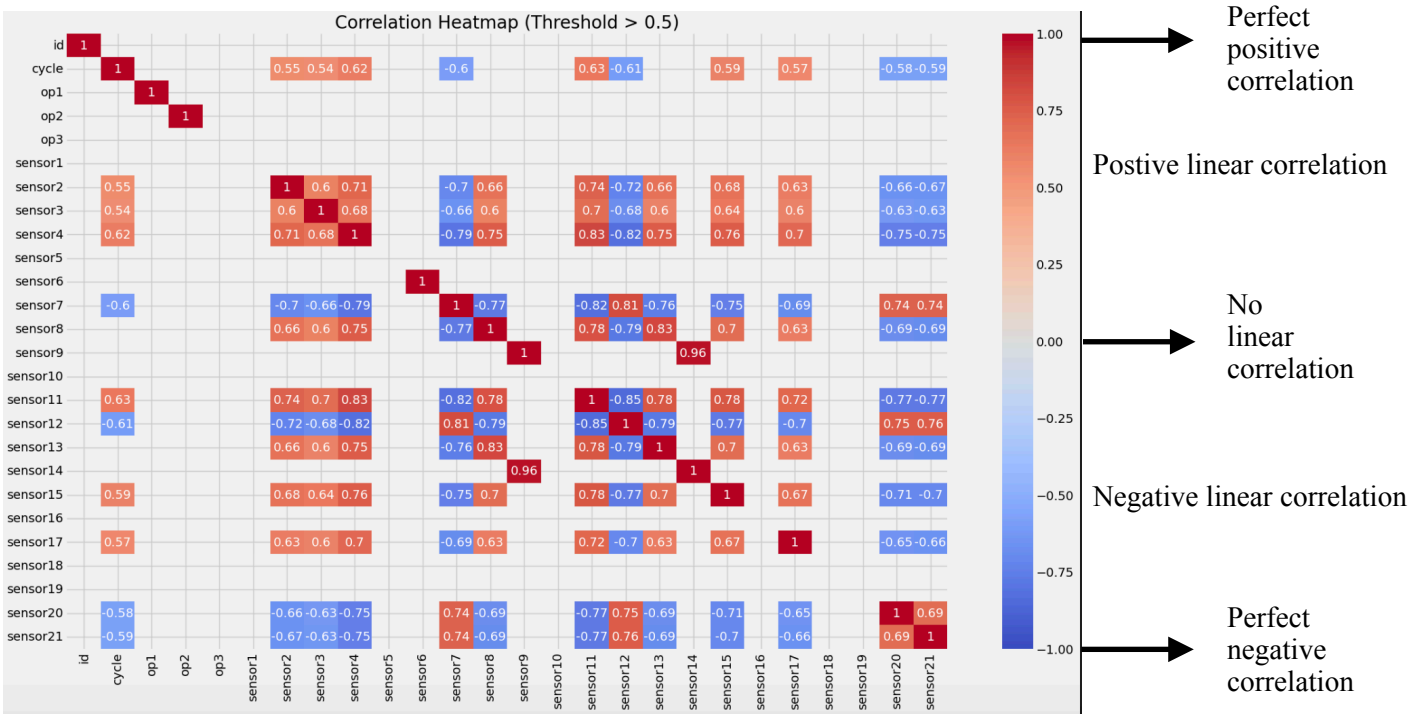

Figure-9. Correlation heat map.

Based on the information presented in Figure-9, this heatmap only displays correlations above the threshold which is 0.5. Red represents positive linear correlation, blue represents negative linear correlation, and zero indicates no correlation, displaying neither color nor a numerical value for non-linear correlation. Therefore, it is evident that certain features, namely op1, op2, op3, sensor1, sensor5, sensor6, sensor9, sensor10, sensor14, sensor16, sensor18, and sensor19, exhibit no linear correlation. Subsequently, a detailed analysis of the correlation matrix reveals that features lacking correlation will be excluded from the training procedure.

4.1.2: Z-Score Normalization

The z-score, also known as the standard score or normalized score, is a statistical measure that quantifies how many standard deviations a data point is from the mean (average) of a dataset. It is a way to standardize and compare data points from different distributions. The C-MAPSS dataset contains sensor measurements and operational parameters from multiple engines, each subjected to different operating conditions and levels of degradation. To effectively analyze and compare the sensor data, z-scores are often used to standardize the features. By calculating the z-scores for each sensor reading, the data becomes normalized, with a mean of 0 and a standard deviation of 1. The formula for calculating the z-score of a data point:

$$X_{\text{std}} = \frac{X - \mu}{\sigma} \tag{1}$$

where X_{std} (Z) represents the standardized value (z-score) of the feature, X is the individual data point, μ is the mean of the dataset and σ is the standard deviation of the dataset.

```
# Scale data for all engines
scaler = StandardScaler()
train_data = scaler.fit_transform(train_data.drop(columns = columns_to_be_dropped))
test_data = scaler.transform(test_data.drop(columns = columns_to_be_dropped))

train_data = pd.DataFrame(data = np.c_[train_data_first_column, train_data])
test_data = pd.DataFrame(data = np.c_[test_data_first_column, test_data])

num_train_machines = len(train_data[0].unique())
num_test_machines = len(test_data[0].unique())
```

Figure-10. Data standardization.

4.2 : LSTM Model Architecture

Shortly after the initial development of Elman-style RNNs trained using back-propagation [51], the challenges associated with learning long-term dependencies, primarily stemming from issues like vanishing and exploding gradients, became evident. Discussions on this problem were notably presented by Bengio and Hochreiter [52][53]. While gradient clipping proved effective in mitigating exploding gradients, addressing vanishing gradients required a more intricate solution. One of the earliest and most successful solutions for tackling vanishing gradients was the introduction of the long short-term memory (LSTM) model, credited to Hochreiter and Schmidhuber [54].

LSTM is a type of RNN architecture that is designed to capture and process sequences of data. It was introduced to address some of the limitations of traditional RNNs, which often struggle with learning long-term dependencies in sequential data. What sets LSTM networks apart from traditional RNNs are their ability to capture long-term dependencies and remember information over extended sequences, and each ordinary recurrent node is replaced by a memory cell. Each memory cell contains an internal state, i.e., a node with a self-connected recurrent edge of fixed weight 1, ensuring that the gradient can pass across many time steps without vanishing or exploding [55]. LSTMs use a form of gradient descent called back-propagation through time (BPTT) to learn and adjust their parameters during training. This enables them to adapt to the patterns and relationships within the input sequences. This is achieved through a

more complex internal structure that includes memory cells, gates, and a mechanism to control the flow of information.

In LSTM models, activation functions like tanh (hyperbolic tangent) and sigmoid serve crucial roles in controlling the flow of information and regulating the output of various components within the LSTM cell. For the tanh function, squashes input values to the range of [-1, 1], making it zero-centered. This helps in mitigating the vanishing gradient problem compared to sigmoid, which is centered around 0.5. Also, it introduces non-linearities that enable the model to learn more complex relationships within the data, allowing LSTMs to capture a broader range of patterns. The derivative of tanh is higher than that of the sigmoid function, promoting a stronger gradient flow during backpropagation, which aids in training deep networks. And for the sigmoid function squeezes input values into the range of [0, 1], making it suitable for gating mechanisms in LSTMs, particularly for controlling the information flow through the gates (input, forget, and output gates). It is specifically useful in LSTM cells to regulate the flow of information through the gates, allowing the model to learn when to store or forget information from the cell state. In an LSTM cell, these activation functions are used in various components like the input gate, forget gate, output gate, and cell state operations. The tanh function is usually used for the cell state operations and candidate value computations, while sigmoid functions are commonly employed in gate mechanisms to control the flow of information in and out of the cell state.

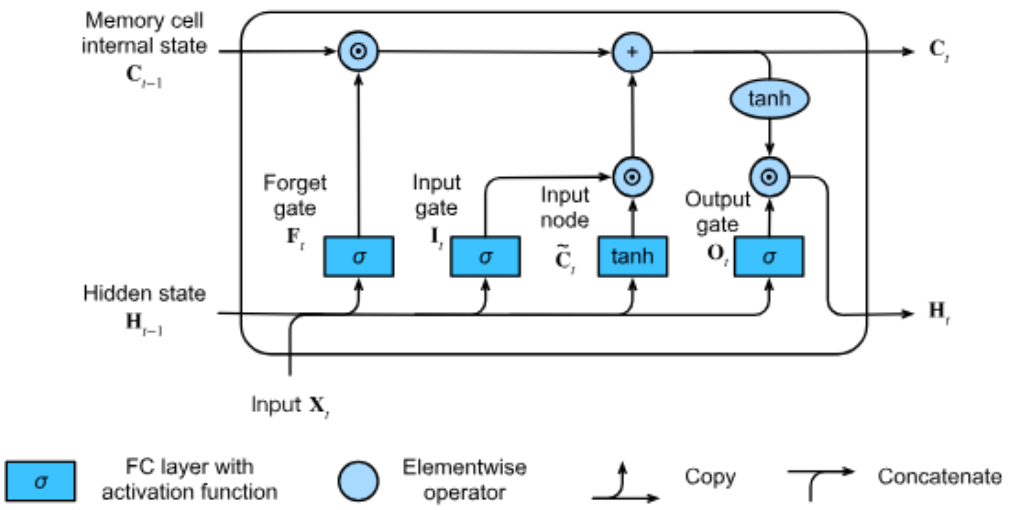

Figure-11: LSTM cell architecture [56].

4.2.1 Memory Cell

Within each memory cell, there exists an internal state alongside a collection of multiplicative gates, and their combined role is to oversee three pivotal functions. Firstly, the presence and influence of a particular input on the internal state are determined, a responsibility shouldered by the input gate. Secondly, the decision regarding the retention or clearing of the internal state is made under the purview of the forget gate. Lastly, the question of whether a specific neuron's internal state should contribute to the cell's output is resolved by the output gate. These three functions, orchestrated by the memory cell's components, are fundamental to its operation.

4.2.2 Gates

LSTMs use three types of gates to control the flow of information: the input gate, the forget gate, and the output gate. These gates regulate what information should be stored, forgotten, and passed to the output. The input gate determines which information from the current input should be stored in the memory cell. Forget gate decides what information from the previous state of the memory cell should be discarded. The output gate controls what information from the memory cell should be used to produce the output. A value approaching 1 signifies the retention of information, while a value nearing 0 implies the exclusion or discarding of said information. The equations for input gate (I_t), forget gate (F_t), and output gate (O_t):

$$\begin{aligned} I_t &= \sigma(W_{xi}X_t + W_{hi}H_{t-1} + W_{ci}C_{t-1} + b_i) \\ F_t &= \sigma(W_{xf}X_t + W_{hf}H_{t-1} + W_{cf}C_{t-1} + b_f) \\ O_t &= \sigma(W_{xo}X_t + W_{ho}H_{t-1} + W_{co}C_t + b_o) \end{aligned} \tag{3}$$

where W represents the weight parameter, b represents the bias parameter, H_{t-1} represents the previous hidden state, X represents the input value, C_t is the cell state, and C_{t-1} is the previous cell state. Also, sigmoid functions to map the input values to the interval form 0 to 1.

4.2.3 Input Node

The input node is also called the candidate cell state and represents the information that can potentially be stored in the cell at the current time step. It is calculated based on the input,

the previous hidden state (H_{t-1}), and a weight matrix. The sigmoid and hyperbolic tangent functions help in scaling and gating the information. This leads to the following equation at time step t :

$$\tilde{C}_t = \tanh(W_{xc}X_t + W_{hc}H_{t-1} + b_c) \tag{4}$$

where $tanh$ represents the hyperbolic tangent activation function, and returns from -1 to 1. In general, the candidate cell state also use G_t to represent.

4.2.4 Memory Cell Internal State

It represents the accumulated knowledge or memory of the LSTM cell at the current time step. The cell state (C_t) is updated by combining the previous cell state (C_{t-1}) with the candidate cell state ($\tilde{C}_t$) based on the input gate (I_t) and forget gate (F_t). It represents the accumulated knowledge or memory of the LSTM cell at the current time step. Using the Hadamard product operator $\odot$, the following update equation:

$$C_t = C_{t-1} \odot F_t + \tilde{C}_t \odot I_t \tag{5}$$

When the forget gate is consistently set to 1 and the input gate remains at 0, the memory cell's internal state remains static, persisting unchanged through each subsequent time step. However, the presence of input and forget gates grants the model the adaptability to decide when to maintain this value unaltered and when to modify it in response to incoming inputs. This

architectural choice effectively addresses the vanishing gradient problem in practice, making models considerably more trainable, particularly when dealing with datasets characterized by lengthy sequences.

4.2.5 Hidden State

The gated hidden state (H_t) is computed by applying the output gate (O_t) to the cell state (C_t) using the hyperbolic tangent function ($tanh$). This procedure guarantees that the values of consistently fall within the range of -1 to 1. This hidden state is the primary output of the LSTM cell and carries information that can be passed to subsequent layers or used for predictions. The equation for the hidden state:

$$H_t = O_t \odot \tanh(C_t) \tag{6}$$

When the output gate approaches a value of 1, it grants unrestricted influence to the memory cell's internal state on subsequent layers. Conversely, for output gate values nearing 0, it prevents the current memory from affecting other network layers during the current time step. It's worth noting that a memory cell can accumulate information over multiple time steps without influencing the remainder of the network (as long as the output gate maintains values close to 0). However, it can swiftly impact the network at a subsequent time step as soon as the output gate transitions from values near 0 to values close to 1.

4.2.6 Training

Training the LSTM-based model on the training set involves using a batch size of 128, running for 10 epochs, and employing a learning rate set at 0.001, as depicted in [Figure-12] and [Figure-13].

```
def create_compiled_model():
    model = Sequential([
        layers.LSTM(128, input_shape = (window_length, 15), return_sequences=True, activation = "tanh"),
        layers.LSTM(64, activation = "tanh", return_sequences = True),
        layers.LSTM(32, activation = "tanh"),
        layers.Dense(96, activation = "relu"),
        layers.Dense(128, activation = "relu"),
        layers.Dense(1)
    ])
    model.compile(loss = "mse", optimizer = tf.keras.optimizers.Adam(learning_rate=0.001))
    return model
```

[146] Python

Figure-12. LSTM model creation.

```
def scheduler(epoch):
    if epoch < 5:
        return 0.001
    else:
        return 0.0001
callback = tf.keras.callbacks.LearningRateScheduler(scheduler, verbose = 1)
model = create_compiled_model()
history = model.fit(processed_train_data, processed_train_targets, epochs = 10,
#history = model.fit(processed_train_data, processed_train_targets, epochs = 20,
#history = model.fit(processed_train_data, processed_train_targets, epochs = 30,
#history = model.fit(processed_train_data, processed_train_targets, epochs = 40,
#history = model.fit(processed_train_data, processed_train_targets, epochs = 100,
                    validation_data = (processed_val_data, processed_val_targets),
                    callbacks = callback,
                    batch_size = 128, verbose = 2)
```

Figure-13. Scheduler function.

4.3: Evaluation Metric

Assessment metrics are numerical criteria employed to gauge the effectiveness of machine learning models, algorithms, or systems. These metrics facilitate the comprehension of a model's performance in a particular task or dataset. In this study, three evaluation metrics are used to evaluate the model.

Firstly, the root mean square error (RMSE) stands as a frequently employed assessment metric in the realm of regression analysis and predictive modeling. Its purpose is to gauge the precision of numerical predictions by quantifying the average magnitude of errors between predicted values and actual values. The equation of RMSE is given below:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{8}$$

where n represents the number of data points in the dataset, y_i represents the actual (observed) value for the i^{th} data point, $\hat{y}_i$ represents the predicted value for the i^{th} data point and $\sum$ denotes the summation of the squared differences between actual and predicted values.

Secondly, the mean absolute error (MAE) is a widely used evaluation metric in regression analysis and predictive modeling to assess the accuracy of numerical predictions. It

measures the average absolute magnitude of errors between predicted values and actual values and treats all errors equally. The equation of MAE is given below:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{9}$$

where n represents the number of data points in the dataset, y_i represents the actual (observed) value for the i^{th} data point, $\hat{y}_i$ represents the predicted value for the i^{th} data point and $\sum$ denotes the summation of the squared differences between actual and predicted values.

Lastly, the scoring function within the C-MAPSS dataset is tailored to reflect real-world industrial scenarios [57]. A superior model performance is characterized by a lower score value. The scoring mechanism penalizes overestimations of RUL values more heavily, as these can potentially lead to asset failure. Conversely, it imposes milder penalties for underestimated RUL values, which correspond to earlier-than-expected asset maintenance. This approach aligns with the practical considerations of the industry [58]. The scoring function is computed as follows:

$$S = \begin{cases} \sum_{i=1}^{N}\left(e^{\frac{g_i - r_i}{13}} - 1\right), & r_i < g_i \\ \sum_{i=1}^{N}\left(e^{\frac{r_i - g_i}{10}} - 1\right), & r_i \geq g_i\,. \end{cases} \tag{10}$$

where r_i and g_i denote the actual RUL and estimated RUL of i^{th} engine, respectively.

4.4 AutoGluon

AutoGluon is an open-source AutoML framework that simplifies and automates the machine learning pipeline. It is designed to make machine learning accessible to users of various skill levels, from beginners to experienced data scientists as creating a prediction model requires just a concise three-line code snippet. AutoGluon provides a high-level API that abstracts away many of the complexities of machine learning, allowing users to build and deploy accurate machine learning models with ease. It automates tasks such as data preprocessing, feature engineering, model selection, and hyper-parameter tuning, enabling users to focus on their specific problem domains rather than the intricacies of machine learning algorithms. AutoGluon is highly versatile and supports a wide range of machine learning tasks, including classification, regression, object detection, and natural language processing. It also includes a user-friendly dashboard for model training and evaluation.

This platform offers several key advantages for machine learning tasks. AutoGluon enables quick prototyping, allowing users to construct machine learning solutions from raw data with just a few lines of code. It leverages state-of-the-art (SOTA) techniques by automatically incorporating cutting-edge models, even for users without specialized expertise. Transitioning from experimental models to production is made simple with cloud predictors and pre-built containers, ensuring seamless deployment. Additionally, the platform is highly customizable, offering extensibility through custom feature processing, models, and metrics to meet unique

project requirements. Its flexibility and extensive capabilities have made it a valuable tool for organizations and individuals seeking efficient and powerful machine learning solutions.

4.4.1 Data Preprocessing

AutoGluon starts with raw data, which can be in various formats, including tabular data, time series data, or even multimodal data combining text, images, and tabular data. It performs essential data preprocessing tasks such as handling missing values, encoding categorical variables, and normalizing or scaling features. Also, AutoGluon's data processing involves two consecutive stages: model-agnostic preprocessing, which applies transformations to all models' inputs, and model-specific preprocessing, exclusively utilized on a duplicate of the data specific to training a particular model. Initially, it automatically determines the type of prediction task based on the label column's values. Non-numeric string values signify a classification task with classes determined by unique values in that column, while numeric values with limited repetition imply a regression task. This feature is part of AutoGluon's efforts to simplify problem translation from raw data to accurate predictions.

Model-agnostic preprocessing categorizes features into numeric, categorical, text, or date/time categories. Uncategorized columns, which typically contain non-numeric and non-repeating fields with low predictive value (e.g., engineIDs), are removed. Text features undergo transformation into numeric vectors based on n-gram features, with only high-occurrence n-grams retained to reduce memory usage. Date or time features are also converted into

appropriate numeric representations. A copy of these processed features, containing both numeric and categorical components, is then provided to model-specific methods for further tailored preprocessing. To address missing discrete variables, AutoGluon introduces an "Unknown" category instead of imputing missing values. This approach ensures that AutoGluon can handle novel categories during testing while preserving the evidence of absence, particularly when observations are not missing at random [59].

4.4.2 Types of Models

AutoGluon provides three distinct prediction patterns: tabular, multimodal, and time series. For conventional tabular datasets represented as tables, AutoGluon offers a diverse range of algorithms, covering neural networks, traditional machine learning, and ensemble techniques. These algorithms encompass neural networks, traditional models like random forests and extreme random trees, the k-nearest neighbors algorithm, and two robust boosting tree algorithms, namely CatBoost [60] and LightGBM [61].

Additionally, AutoGluon's MultiModalPredictor presents a deep learning model zoo capable of automatically constructing cutting-edge deep learning models suitable for various input types, including images, text, and tabular data.

Finally, AutoGluon addresses time series data, which consists of measurements collected at regular intervals. Time series forecasting aims to predict future values based on past observations. AutoGluon offers a selection of 14 default time series forecasting models,

including the naive model, seasonal naive model, ARIMA model, ETS model, auto ARIMA model, auto ETS model, Theta model [62], dynamic optimized Theta model [63], direct tabular model, recursive tabular model, DeepAR model [64], DLinear model [65], patch TST model [66], simple feedforward model, and temporal fusion transformer model [67]. So far, AutoGluon has not yet incorporated the LSTM algorithm.

4.4.3 Multi-layer Stack Ensembling

Ensembling stands as a well-established method to enhance model accuracy by amalgamating their predictions [68], thereby refining generalization. A prominent ensemble learning technique for both classification and regression is the random forest. It employs bagging [69] (bootstrap aggregating) to construct complete decision trees in parallel, drawing from random bootstrap samples of the dataset and its features. Predictions are derived by aggregating outcomes from all these trees, curtailing variance and elevating predictive precision. The ultimate prediction is a majority class or mean regression gleaned from all the decision tree forecasts. The introduction of randomness plays a pivotal role in the forest's success, with bagging ensuring that no two decision trees are alike, mitigating the overfitting issues associated with individual trees. These fundamental principles underlie popular machine learning algorithms such as random forest, XGBoost [70], Catboost, and LightGBM, all of which are leveraged by AutoGluon.

Furthermore, ensembling techniques, which combine predictions from multiple models, are well-known for their effectiveness in improving predictive accuracy by reducing prediction variance. Various AutoML frameworks utilize ensembling methods such as bagging, boosting [71], stacking [72], or weighted combinations. Within the ML competition community, it's a rare occurrence for a single model to claim victory. Almost invariably, winning solutions involve the amalgamation of model ensembles. Shallow stack ensembling is common, where individual "base" models are trained separately, and their predictions are used as features for a "stacker" model. AutoGluon introduces a novel multi-layer stack ensembling approach, employing multiple layers of stacker models illustrated in Figure-14. These stackers use both the predictions from the previous layer and the original data features as inputs, allowing higher-layer stackers to revisit the original data. Additionally, AutoGluon uses ensemble selection in the final stacking layer to aggregate stacker model predictions in a weighted manner [73], enhancing resilience against overfitting.

Stacking, a powerful technique in machine learning, involves utilizing the collective predictions of a set of "base" regression or classification models to create features used for training a higher-level meta-classifier or regressor known as the "stacker" model. Taking this concept a step further, multi-layer stacking leverages the predictions generated by stacker models and feeds them as inputs to additional higher-layer stacker models. This iterative approach has proven to be a successful strategy in various Kaggle competitions. This innovative multi-layer stack ensembling strategy improves predictive accuracy and is a unique feature of AutoGluon [73]

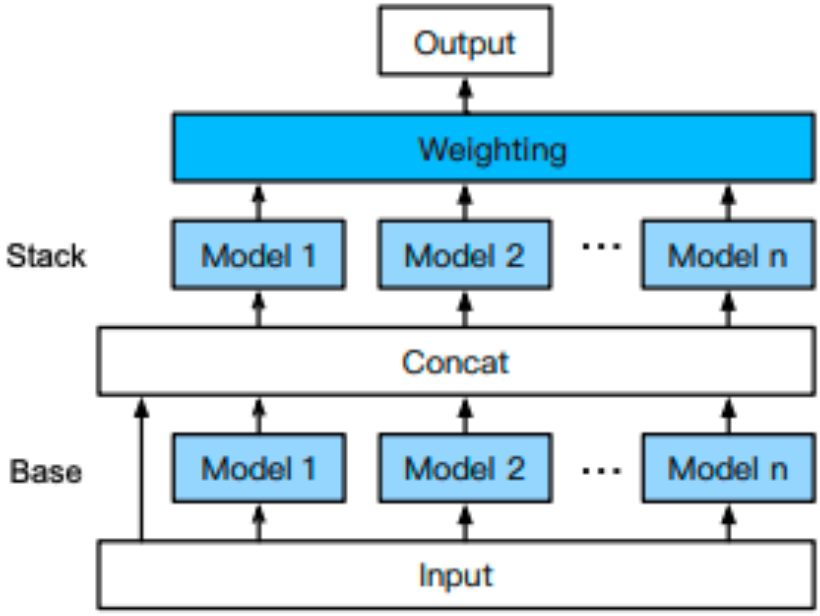

Figure-14. AutoGluon's multi-layer stacking strategy [9].

4.4.4 k-fold Ensemble Bagging

In AutoGluon, the technique of *k*-fold ensemble bagging is employed to enhance the stacking performance of its machine learning models. It operates by first randomly dividing the training data into *k* distinct subsets or folds taking care to maintain proportional class representation in classification tasks [74]. Then, AutoGluon trains *k* copies of each base machine learning model, with each copy being trained on *k*-1 of the folds while leaving one fold out. During this training process, each model generates predictions for the held-out fold, known as out-of-fold (OOF) predictions. Subsequently, AutoGluon constructs an ensemble by training a

stacker model on these OOF predictions, learning how to effectively combine the base models' predictions. To enhance stability and robustness, AutoGluon repeats this entire process n times, using different random data partitions for each repetition. The final predictions are then obtained by averaging the predictions of the stacker models across all n repetitions. This repeated *k*-fold bagging approach helps reduce variance, mitigate overfitting, and ultimately improves the accuracy and reliability of AutoGluon's machine learning models, particularly beneficial for smaller datasets where overfitting and data variance can be critical challenges.

The overall training strategy is outlined in Figure-15, where each stacking layer is assigned a time budget denoted as T_{total}/L. In Step 7, AutoGluon initiates the estimation of required training time, and if this estimation surpasses the remaining time allocated for the current layer, it proceeds to the subsequent stacking layer. Each newly trained model is promptly saved to disk to ensure fault tolerance, resulting in a framework that exhibits high predictability in its behavior.

This design guarantees the generation of predictions, as long as at least one model can be trained on one fold within the given time limit. AutoGluon also accounts for potential training failures and seamlessly moves on to the next model in such cases. Unlike many AutoML frameworks that attempt to train multiple models in parallel on a single instance, AutoGluon-Tabular opts for sequential model training, leveraging the efficiency of individual implementations to make the most of multiple cores. This sequential approach enables successful training even on larger datasets, where other frameworks may encounter out-of-memory errors without meticulous tuning.

Algorithm 1 AutoGluon-Tabular Training Strategy (multi-layer stack ensembling + n-repeated k-fold bagging).

Require: data (X, Y), family of models $\mathcal{M}$, # of layers L

Preprocess data to extract features
for $l = 1$ **to** L **do** {Stacking}
  for $i = 1$ **to** n **do** {n-repeated}
    Randomly split data into k chunks $\{X^j, Y^j\}_{j=1}^{k}$
    for $j = 1$ **to** k **do** {k-fold bagging}
      for each model type m in $\mathcal{M}$ **do**
        Train a type-m model on X^{-j}, Y^{-j}
        Make predictions $\hat{Y}_{m,i}^{j}$ on OOF data X^j
      end for
    end for
  end for
  Average OOF predictions $\hat{Y}_m = \{\frac{1}{n}\sum_i \hat{Y}_{m,i}^{j}\}_{j=1}^{k}$
  $X \leftarrow$ concatenate$(X, \{\hat{Y}_m\}_{m \in \mathcal{M}})$
end for

Figure-15. Tabular training strategy [9].

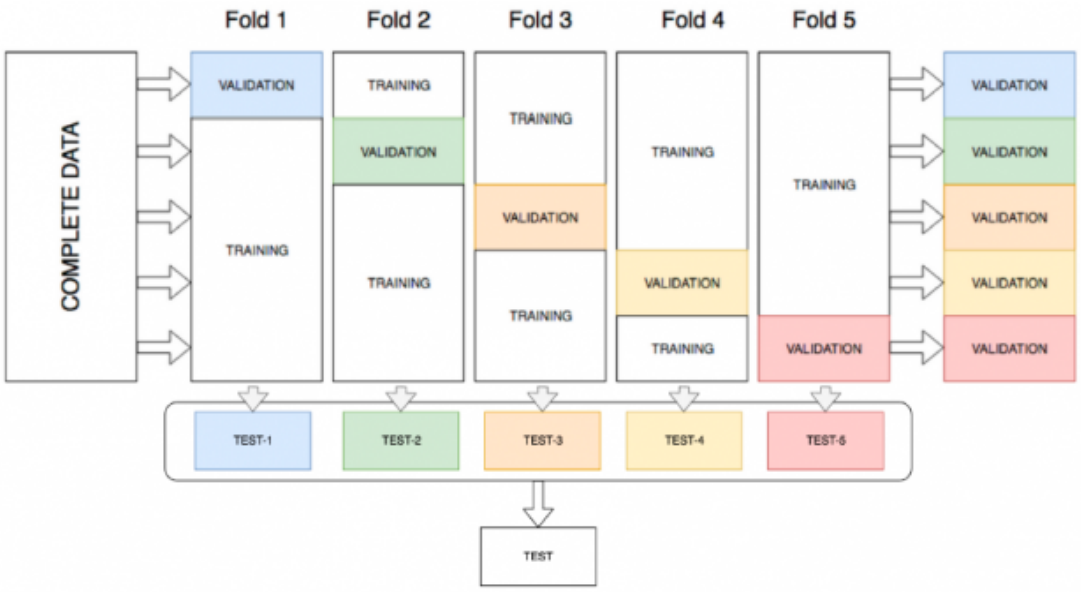

Figure-16. K-fold ensemble bagging [75].

4.4.5 Implementation

AutoGluon offers the ability to train models with minimal coding, and the main three lines of code includes loading the data or library, training the model, and making prediction as shown in Figure-17.

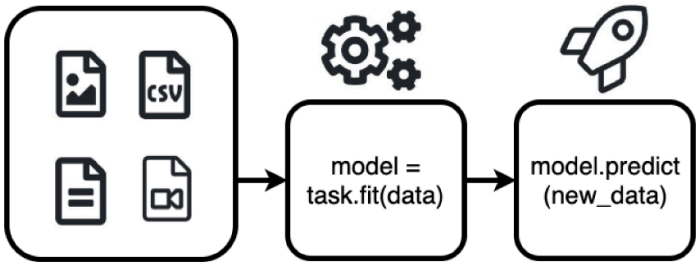

Figure-17.
A schematic illustrating 3 lines of code implementation [75].

1. Loading data and importing time series predictor:

```
>>> from autogluon.timeseries import TimeSeriesDataFrame, TimeSeriesPredictor
>>> import pandas as pd
>>> df = pd.read_csv('~/CMAPSSData/train_FD001.cols.update.csv')
>>> df.head()
   ID      Cycle  Sensor2  Sensor3  Sensor4  Sensor7  Sensor8  Sensor9  Sensor12  Sensor13
Sensor14  Sensor15  Sensor17  Sesnor20  Sensor21  RUL
0   1  1696704001   641.82  1589.70  1400.60   554.36  2388.06  9046.19    521.66   2388.02
8138.62    8.4195       392     39.06   23.4190  191
1   1  1696704002   642.15  1591.82  1403.14   553.75  2388.04  9044.07    522.28   2388.07
8131.49    8.4318       392     39.00   23.4236  190
```

```
2   1  1696704003   642.35  1587.99  1404.20   554.26  2388.08  9052.94    522.42   2388.03
8133.23    8.4178      390     38.95   23.3442  189
3   1  1696704004   642.35  1582.79  1401.87   554.45  2388.11  9049.48    522.86   2388.08
8133.83    8.3682      392     38.88   23.3739  188
4   1  1696704005   642.37  1582.85  1406.22   554.00  2388.06  9055.15    522.19   2388.04
8133.80    8.4294      393     38.90   23.4044  187

>>> train_data = TimeSeriesDataFrame.from_data_frame(df, id_column="ID",
timestamp_column="Cycle")
>>> train_data.head()
                              Sensor2  Sensor3  Sensor4  Sensor7  Sensor8  Sensor9  ...  Sensor14
Sensor15  Sensor17  Sesnor20  Sensor21  RUL
item_id timestamp                                                                    ...
1       1970-01-01 00:00:01.696704001   641.82  1589.70  1400.60   554.36  2388.06  9046.19  ...
8138.62    8.4195      392     39.06   23.4190  191
        1970-01-01 00:00:01.696704002   642.15  1591.82  1403.14   553.75  2388.04  9044.07  ...
8131.49    8.4318      392     39.00   23.4236  190
        1970-01-01 00:00:01.696704003   642.35  1587.99  1404.20   554.26  2388.08  9052.94  ...
8133.23    8.4178      390     38.95   23.3442  189
        1970-01-01 00:00:01.696704004   642.35  1582.79  1401.87   554.45  2388.11  9049.48  ...
8133.83    8.3682      392     38.88   23.3739  188
        1970-01-01 00:00:01.696704005   642.37  1582.85  1406.22   554.00  2388.06  9055.15  ...
8133.80    8.4294      393     38.90   23.4044  187
```

---

2. Training the model and use RMSE for evaluation metric:

```
[5 rows x 14 columns]
>>> predictor = TimeSeriesPredictor(target="RUL", eval_metric="RMSE")
>>> predictor.fit(train_data, presets="medium_quality", time_limit=600)
================ TimeSeriesPredictor ================
TimeSeriesPredictor.fit() called
Setting presets to: medium_quality
Fitting with arguments:
{'enable_ensemble': True,
 'evaluation_metric': 'RMSE',
 'excluded_model_types': None,
 'hyperparameter_tune_kwargs': None,
 'hyperparameters': 'medium_quality',
 'num_val_windows': 1,
```

```
 'prediction_length': 1,
 'random_seed': None,
 'target': 'RUL',
 'time_limit': 600,
 'verbosity': 2}
Provided training data set with 20631 rows, 100 items (item = single time series). Average time series length is 206.3. Data frequency is 'N'.
=====================================================
AutoGluon will save models to AutogluonModels/ag-20231009_163513/
AutoGluon will gauge predictive performance using evaluation metric: 'RMSE'
        This metric's sign has been flipped to adhere to being 'higher is better'. The reported score can be multiplied by -1 to get the metric value.

Provided dataset contains following columns:
        target:          'RUL'
        past covariates:  ['Sensor2', 'Sensor3', 'Sensor4', 'Sensor7', 'Sensor8', 'Sensor9', 'Sensor12', 'Sensor13', 'Sensor14', 'Sensor15', 'Sensor17', 'Sesnor20', 'Sensor21']

Starting training. Start time is 2023-10-09 09:35:25
Models that will be trained: ['Naive', 'SeasonalNaive', 'Theta', 'AutoETS']
Training timeseries model Naive. Training for up to 599.64s of the 599.64s of remaining time.
        -1.0000      = Validation score (-RMSE)
        0.01    s    = Training runtime
        4.49    s    = Validation (prediction) runtime
Training timeseries model SeasonalNaive. Training for up to 595.13s of the 595.13s of remaining time.
        -1.0000      = Validation score (-RMSE)
        0.01    s    = Training runtime
        0.05    s    = Validation (prediction) runtime
Training timeseries model Theta. Training for up to 595.07s of the 595.07s of remaining time.
        -0.5051      = Validation score (-RMSE)
        0.02    s    = Training runtime
        11.10   s    = Validation (prediction) runtime
Training timeseries model AutoETS. Training for up to 583.94s of the 583.94s of remaining time.
        Warning: AutoETS/W0 failed for 19 time series (19.0%). Fallback model SeasonalNaive was used for these time series.
        -0.4359      = Validation score (-RMSE)
        0.01    s    = Training runtime
        8.95    s    = Validation (prediction) runtime
Fitting simple weighted ensemble.
        -0.3938      = Validation score (-RMSE)
        0.47    s    = Training runtime
```

```
        20.06   s     = Validation (prediction) runtime
Training complete. Models trained: ['Naive', 'SeasonalNaive', 'Theta', 'AutoETS',
'WeightedEnsemble']
Total runtime: 25.51 s
Best model: WeightedEnsemble
Best model score: -0.3938 (39.38%)
```

As a result, the best model is WeightedEnsemble, and the best score of RMSE is 0.3938. Using RMSE as an evaluation metric will have their signs reversed, resulting in negative values. This approach is adopted to simplify the interpretation of the leaderboard, ensuring that users can intuitively discern that higher scores indicate better performance without the need to understand the intricacies of each specific metric. Also, setting up the time limit can accelerate model generation.

```
Fitting simple weighted ensemble.
        -0.3938       = Validation score (-RMSE)
        0.48    s     = Training runtime
        4.06    s     = Validation (prediction) runtime
Training complete. Models trained: ['Naive', 'SeasonalNaive', 'Theta', 'AutoETS', 'WeightedEnsemble']
Total runtime: 4.78 s
Best model: WeightedEnsemble
Best model score: -0.3938
<autogluon.timeseries.predictor.TimeSeriesPredictor object at 0x284ebfc70>
```

Figure-18. Finding the best model.

5. EXPERIMENTS

5.1 LSTM Model Result Analysis

After data preprocessing and training were conducted on an LSTM model, RUL prediction was performed. This study provides an interpretation of the results for the FD001 dataset. The RMSE value obtained is 13.78% [Figured-19], and the S-score is 394.62 [Figure-20]. This study focuses solely on using the LSTM model, which yields quite good accuracy, the true RUL and predicted RUL are very similar, as documented in Figure-21. Many studies have shown that combining models such as MLP, RNNs, CNNs, Bidirectional LSTM (BiLSTM), CNN-LSTM and deep layer-recurrent neural networks (DL-RNNs) can result in lower RMSE [76] [77] [78]. However, it is essential to note that this process requires a significant amount of time for tuning and testing, especially for the FD002 and FD004 databases, where feature correlations are very low during feature engineering. This poses a challenge when selecting evaluation methods. Data preprocessing and the algorithm of the model determine the quality of the model.

```
    rul_pred = model.predict(processed_test_data).reshape(-1)
    preds_for_each_engine = np.split(rul_pred, np.cumsum(num_test_windows_list)[:-1])
    mean_pred_for_each_engine = [np.average(ruls_for_each_engine, weights = np.repeat(1/num_windows, num_windows))
                                 for ruls_for_each_engine, num_windows in zip(preds_for_each_engine, num_test_windows_list)]
    RMSE = np.sqrt(mean_squared_error(true_rul, mean_pred_for_each_engine))
    print("RMSE: ", RMSE)

16/16 [==============================] - 2s 25ms/step
RMSE:  13.780660378275273
```

Figure-19. RMSE evaluation.

```
def compute_s_score(rul_true, rul_pred):
    """
    Both rul_true and rul_pred should be 1D numpy arrays.
    """
    diff = rul_pred - rul_true
    return np.sum(np.where(diff < 0, np.exp(-diff/13)-1, np.exp(diff/10)-1))
s_score = compute_s_score(true_rul, preds_for_last_example)
print("S-score: ", s_score)
```

```
S-score:  394.6238444270357
```

Figure-20. S-score evaluation.

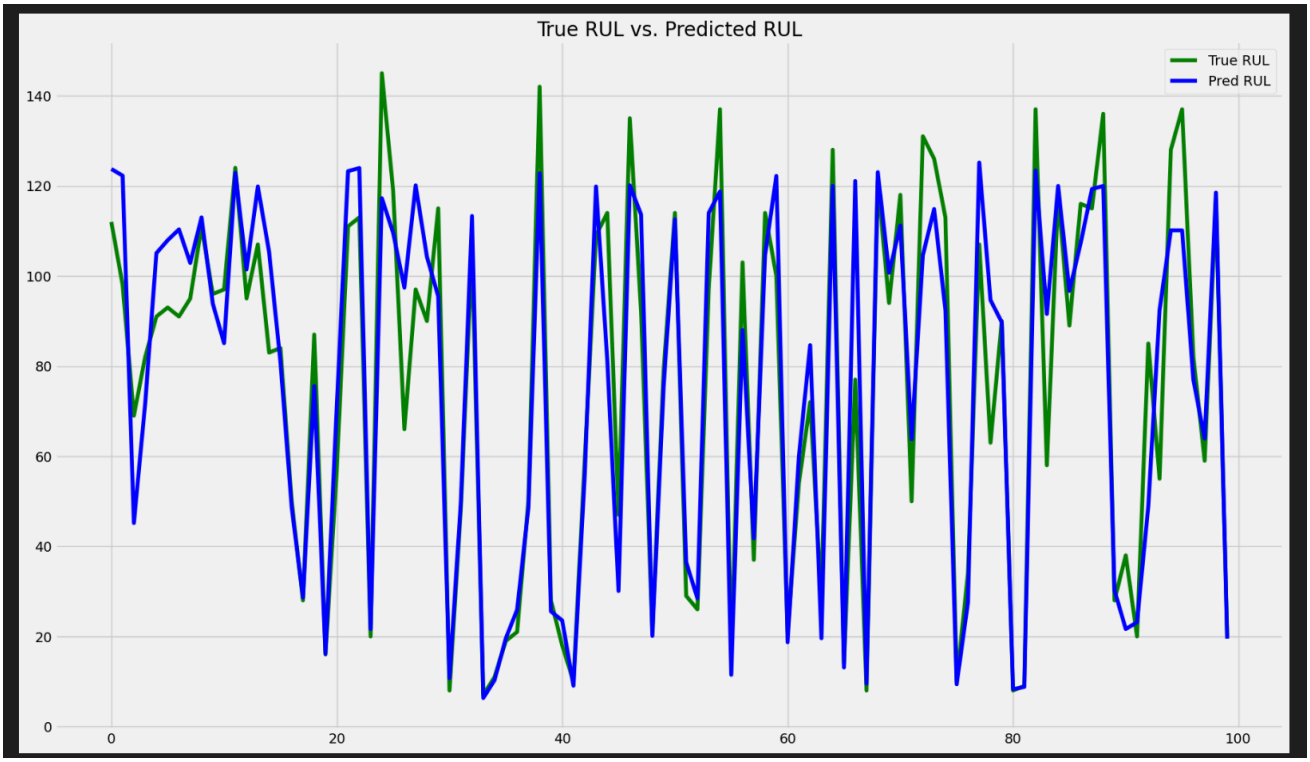

Figure-21. True RUL vs. Predicted RUL, where the x-axis represents Engine ID, and the y-axis represents RUL values.

5.2 AutoGluon Result Analysis

AutoGluon training and RUL prediction using timed series forecasting was conducted on Naive, SeasonalNaive, Theta, AutoETS, and Weighted Ensemble models. This study provides an interpretation of the results for the FD001 dataset. Comparing the final score of each model, the RMSE value obtained from the best model is 40.17% [Figure-22]. The results of testing data scores are sorted in 'score_test" and metrics scores always show in higher is better form. Although AutoGluon introduced support for time series forecasting in June 2022, it remains a relatively new technique with ample room for further enhancement and development. This result shows only the most basic function of AutoGluon. AutoGluon provides numerous general evaluation metrics. However, specific metrics like S-Score, which are exclusively designed for evaluating C-MAPSS, are not included. Advanced users can override the presets and manually specify what models should be trained by the predictor using the hyper-parameters argument. Also, AutoGluon offers multiple ways to configure the behavior of a TimeSeriesPredictor that are suitable for both beginners and expert users.

```
>>> predictor.leaderboard(test_data, silent=True)
/opt/homebrew/Caskroom/miniforge/base/lib/python3.10/site-packages/autogluon/timeseries/predictor.py:224: UserWarning: Detected frequency 'N' is not supp
orted by TimeSeriesPredictor. This may lead to some models not working as intended. Please convert the timestamps to one of the supported frequencies: {'
Y', 'M', 'H', 'D', 'W', 'S', 'T', 'A', 'Q', 'min'}. See https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases for detail
s.
  warnings.warn(
Additional data provided, testing on additional data. Resulting leaderboard will be sorted according to test score (`score_test`).
        Warning: AutoETS/W0 failed for 20 time series (20.0%). Fallback model SeasonalNaive was used for these time series.
              model  score_test  score_val  pred_time_test  pred_time_val  fit_time_marginal  fit_order
0  WeightedEnsemble   -0.401716  -0.393785        2.817217       4.055549           0.482289          5
1           AutoETS   -0.447214  -0.435890        2.716938       3.931814           0.010310          4
2             Theta   -0.505050  -0.505050        0.098889       0.123735           0.011065          3
3     SeasonalNaive   -1.000000  -1.000000        0.111611       0.060164           0.011416          2
4             Naive   -1.000000  -1.000000        0.076890       0.085990           0.016422          1
>>>
```

Figure-22. Leaderboard of models.

5.3 Manual Machine Learning vs. Automated Machine Learning

1. AutoGluon requires a much shorter period of time for coding and testing compared to the LSTM manual model coding and testing. LSTM manual model is time-consuming.

2. AutoGluon model RMSE value of 40.17% is greater than the LSTM model RMSE value of 13.78%. Thus the LSTM model is more accurate than the AutoGluon model.

3. AutoGluon automates various stages of the machine learning process, making it accessible to non-experts and reducing the need for manual intervention but building the LSTM model requires substantial expertise for data preprocessing, model selection, and coding.

4. Manual models are often more interpretable, making it easier to understand the underlying factors driving predictions. From the result, AutoGluon-generated models are less interpretable than LSTM models, making it challenging to explain predictions.

6. CONCLUSION

From the experimental results, it can be seen that manual machine learning has a higher accuracy compared to automated learning. However, manual machine learning is significantly more time-consuming, requiring a solid background knowledge, coding skills, and background research. In contrast, the AutoGluon platform is simpler and more user-friendly. Both approaches have their pros and cons, depending on the users' needs. Over the past decade, machine learning has made significant advancements, giving rise to innovative model architectures and methods for handling extensive datasets. For complex datasets like C-MAPSS, the process of training models, finding suitable algorithms, and repeatedly testing model performance is time-consuming. As state-of-the-art ML techniques continue to grow in complexity, even experienced ML experts face the challenge of keeping up with the latest best practices in modeling. The AutoGluon framework offers an appealing solution to this dilemma by automating model selection, ensembling, hyper-parameter tuning, feature engineering, data preprocessing, and data splitting, reducing the need for ongoing manual intervention. It lowers the entry threshold and enhances efficiency, but the crucial factor is enhancing precision. Adding a customization method to the AutoGluon model may reduce the RMSE value and improve accuracy. In summary, the contribution of this study lies in understanding the importance of RUL, basic knowledge of turbofan engines, introducing two different RUL prediction methods: manual machine learning using LSTM algorithms and automated machine learning using the AutoGluon’s time series model, and finally comparing the experimental results of these two approaches.

7. FUTURE WORK

In the future, the commitment is to train the existing models on the additional three datasets of C-MAPSS, including FD002, FD003, and FD004. The additional datasets containing different conditions and fault modes may require further data preprocessing. This will allow for a more comprehensive set of experimental results, enabling a more objective comparison between manual machine learning and automated machine learning. Subsequently, based on the experimental results, the model configurations will be improved to achieve a more accurate RUL prediction model, thus making a substantial contribution to the maintenance of turbofan engines and aviation safety. Furthermore, with the rapid advancement of technology, the simplification of complex workflows and automation to save time has always been a goal pursued by people. Self-driving cars and automated meal delivery robots have proven this. AutoML is expected to be a future trend. It functions like an encyclopedia, recording every ML algorithm, their strengths and weaknesses, and where they are best applied. For example, this study indicates that the LSTM model is suitable for evaluating time series data, and AutoGluon can add the LSTM model in time series modeling, making evaluations more convincing. Therefore, it is crucial to enhance the algorithms and models in the database, optimize recognition of input data, and automatically correct models with low accuracy to improve the evaluation accuracy in today's automated machine learning.

References

[1] A. Saxena and K. Goebel (2008). "Turbofan Engine Degradation Simulation Data Set", NASA Prognostics Data Repository, NASA Ames Research Center, Moffett Field, CA

[2] Benedettini, Ornella & Baines, Tim & Lightfoot, Howard & Greenough, Richard. (2009). State-of-the-art in integrated vehicle health management. Proceedings of The Institution of Mechanical Engineers Part G-journal of Aerospace Engineering - PROC INST MECHENG G-J A E. 223. 157-170. 10.1243/09544100JAERO446.

[3] L. D. Alford. (2000). *The problem with aviation COTS*. IEEE. https://ieeexplore.ieee.org/Xplore/guesthome.jsp. 2000, pp. 519-524, doi: 10.1109/AUTEST.2000.885634.

[4] (IATA), I. A. T. A. (2023). *IATA releases 2022 Airline Safety Performance*. IATA. https://www.iata.org/en/pressroom/2023-releases/2023-03-07-01/

[5] Cinar, Zeki & Nuhu, Abubakar & Zeeshan, Qasim & Korhan, Orhan & Asmael, Mohammed & Safaei, Babak. (2020). Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0. Sustainability. 12. 8211. 10.3390/su12198211.

[6] Contributor, M. H. I. (2021, June 8). *Mitsubishi Heavy Industries Brandvoice: 5 strategies that help airlines reduce the cost of maintenance*. Forbes. https://www.forbes.com/sites/mitsubishiheavyindustries/2019/09/18/5-strategies-that-help-airlines-reduce-the-cost-of-maintenance/?sh=2c67bfce2fce

[7] Zhang, Yang & Hutchinson, Paul & Lieven, Nicholas & Nunez-Yanez, Jose. (2020). Remaining Useful Life Estimation Using Long Short-Term Memory Neural Networks

and Deep Fusion. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2966827.

[8] Asif, O., Haider, S. A., Naqvi, S. R., Zaki, J. F., Kwak, K.-S., & Islam, S. M. (2022). A deep learning model for remaining useful life prediction of aircraft turbofan engine on C-MAPSS dataset. *IEEE Access*, *10*, 95425–95440. https://doi.org/10.1109/access.2022.3203406

[9] Erickson, Nick & Mueller, Jonas & Shirkov, Alexander & Zhang, Hang & Larroy, Pedro & Li, Mu & Smola, Alexander. (2020). AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data.

[10] Bolander, Nathan & Qiu, Hai & Eklund, N. & Hindle, E. & Rosenfeld, T.. (2009). Physics-based remaining useful life prediction for aircraft engine bearing prognosis.

[11] Liu, Liansheng & Guo, Qing & Liu, Datong & Peng, Yu. (2019). Data-Driven Remaining Useful Life Prediction Considering Sensor Anomaly Detection and Data Recovery. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2914236.

[12] Aykol, Muratahan & Gopal, Chirranjeevi & Anapolsky, Abraham & Herring, Patrick & Vlijmen, Bruis & Berliner, Marc & Bazant, Martin & Braatz, Richard & Chueh, William & Storey, Brian. (2021). Perspective—Combining Physics and Machine Learning to Predict Battery Lifetime. Journal of The Electrochemical Society. 168. 10.1149/1945-7111/abec55.

[13] Simon, Dan & Simon, Donald. (2003). Aircraft Turbofan Engine Health Estimation Using Constrained Kalman Filtering. Journal of Engineering for Gas Turbines and Power. 127. 10.1115/GT2003-38584.

[14] Chen, Shaowei & Wang, Meinan & Huang, Dengshan & Wen, Pengfei & Wang, Shengyue & Zhao, Shuai. (2020). Remaining Useful Life Prediction for Complex Systems With Multiple Indicators Based on Particle Filter and Parameter Correlation. IEEE Access. 8. 10.1109/ACCESS.2020.3041682.

[15] Oh, Hyunseok & Azarian, Michael & Pecht, Michael & White, Clifford & Sohaney, Richard & Rhem, Edward. (2010). Physics-of-failure approach for fan PHM in electronics applications. 2010 Prognostics and System Health Management Conference, PHM '10. 1 - 6. 10.1109/PHM.2010.5413501.

[16] Gouda, Osama & Dessouky, Sobhy & Kalas, Ahmed & Hamdy, Mohamed & Nawar, Amira. (2019). Corrosion Effects on the Grounding Resistance of Grounding System Buried in Two-Layers Soil. International Journal of Applied Energy Systems. 1. 10.21608/ijaes.2019.169958.

[17] Rajaraman, Dhanraj & Hertelé, Stijn & Fauconnier, Dieter. (2023). A novel calibration procedure of Johnson-Cook damage model parameters for simulation of scratch abrasion. Wear. 528-529. 10.1016/j.wear.2023.204977.

[18] Ye, Hong & Wan, Yi & Ren, Xiao & Zhanqiang, Liu. (2013). Validity Analysis of Generalized Taylor Regression Formula by Tool Wear Test for Difficult-to-Machine Materials. Key Engineering Materials. 589-590. 342-350. 10.4028/www.scientific.net/KEM.589-590.342.

[19] Byington, Carl & Watson, M. & Edwards, D. & Stoelting, P.. (2004). A model-based approach to prognostics and health management for flight control actuators. IEEE Aerospace Conference Proceedings. 6. 3551 - 3562 Vol.6. 10.1109/AERO.2004.1368172.

[20] Nguyen, Hoang-Phuong & Liu, Jie & Zio, Enrico. (2019). Ensemble of Models for Fatigue Crack Growth Prognostics. IEEE Access. 7. 49527-49537. 10.1109/ACCESS.2019.2910611.

[21] Luo, J., Namburu, M., Pattipati, K., Qiao, L., Kawamoto , M., & Chigusa, S. (2003, September). *Model-based prognostic techniques [maintenance applications] - IEEE xplore*. Model-based prognostic techniques [maintenance applications]. https://

ieeexplore.ieee.org/abstract/document/1243596/

[22] Dong, Guangzhong & Yang, Fangfang & Wei, Zhongbao & Wei, Jingwen & Tsui, Kwok-Leung. (2019). Data-Driven Battery Health Prognosis Using Adaptive Brownian Motion Model. IEEE Transactions on Industrial Informatics. PP. 1-1. 10.1109/TII.2019.2948018.

[23] Chaoui, Hicham & Ibe-Ekeocha, Chris. (2017). State of Charge and State of Health Estimation for Lithium Batteries Using Recurrent Neural Networks. IEEE Transactions on Vehicular Technology. PP. 1-1. 10.1109/TVT.2017.2715333.

[24] Dhiman, Harsh & Deb, Dipankar & Carroll, James & Muresan, Vlad & Unguresan, M.L.. (2020). Wind Turbine Gearbox Condition Monitoring Based on Class of Support Vector Regression Models and Residual Analysis. Sensors. 20. 10.3390/s20236742.

[25] Wang, Biao & Lei, Yaguo & Li, Naipeng & Li, Ningbo. (2018). A Hybrid Prognostics Approach for Estimating Remaining Useful Life of Rolling Element Bearings. IEEE Transactions on Reliability. PP. 1-12. 10.1109/TR.2018.2882682.

[26] Sun, Huibin & Cao, Dali & Zhao, Zidong & Kang, Xia. (2018). A Hybrid Approach to Cutting Tool Remaining Useful Life Prediction Based on the Wiener Process. IEEE Transactions on Reliability. PP. 1-10. 10.1109/TR.2018.2831256.

[27] Zhao, Li, Y.-G., & Sampath, S. (2023). A hierarchical structure built on physical and data-based information for intelligent aero-engine gas path diagnostics. *Applied Energy*, *332*, 120520–. https://doi.org/10.1016/j.apenergy.2022.120520

[28] El Mejdoubi, Asmae & Oukaour, Amrane & CHAOUI, Hicham & Slamani, Y. & Sabor, Jalal & Gualous, Hamid. (2015). Online Supercapacitor Diagnosis for Electric Vehicle Applications. IEEE Transactions on Vehicular Technology. 65. 10.1109/TVT.2015.2454520.

[29] Rodriguez-Guerrero, M.A.; Jaen-Cuellar, A.Y.; Carranza-Lopez-Padilla, R.D. (2018); A

Osornio-Rios, R.; Herrera-Ruiz, G.; Romero-Troncoso, R. Hybrid Approach Based on GA and PSO for Parameter Estimation of a Full Power Quality Disturbance Parameterized Model. *IEEE Trans. Ind. Inform. 14*, 1016–1028.

[30] Qasim, Mohammed & Khadkikar, Vinod. (2014). Application of Artificial Neural Networks for Shunt APF Control. IEEE Transactions on Industrial Informatics. 10. 10.1109/TII.2014.2322580.

[31] Babu, Giduthuri & Zhao, Peilin & li, Xiaoli. (2016). Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life. 214-228. 10.1007/978-3-319-32025-0_14.

[32] Li, Xiang & Ding, Qian & Sun, J. Q.. (2017). Remaining Useful Life Estimation in Prognostics Using Deep Convolution Neural Networks. Reliability Engineering & System Safety. 172. 10.1016/j.ress.2017.11.021.

[33] Zheng, Shuai & Ristovski, Kosta & Farahat, Ahmed & Gupta, Chetan. (2017). Long Short-Term Memory Network for Remaining Useful Life estimation. 88-95. 10.1109/ICPHM.2017.7998311.

[34] Liao, Yuan & Zhang, Linxun & Liu, Chongdang. (2018). Uncertainty Prediction of Remaining Useful Life Using Long Short-Term Memory Network Based on Bootstrap Method. 1-8. 10.1109/ICPHM.2018.8448804.

[35] Agarwal, Sarthak & Saxena, Vaibhav & Singal, Vaibhav & Aggarwal, Swati. (2018). LSTM based Music Generation with Dataset Preprocessing and Reconstruction Techniques. 455-462. 10.1109/SSCI.2018.8628712.

[36] Kefalas, Marios & Baratchi, Mitra & Apostolidis, Asteris & Herik, Dirk & Bäck, Thomas. (2021). Automated Machine Learning for Remaining Useful Life Estimation of Aircraft Engines. 10.1109/ICPHM51084.2021.9486549.

[37] Lin, Sun, Y., Jiao, W., Zheng, J., Li, Z., & Zhang, S. (2023). Prediction of Compressive

Strength and Elastic Modulus for Recycled Aggregate Concrete Based on AutoGluon. *Sustainability (Basel, Switzerland)*, *15*(16), 12345–. https://doi.org/10.3390/su151612345

[38] Lan, Haibing & Yang, Linlin & Zheng, Fengjie & Zong, Chaoyong & Wu, Si & Song 宋学官, Xueguan. (2020). Analysis and optimization of high temperature proton exchange membrane (HT-PEM) fuel cell based on surrogate model. International Journal of Hydrogen Energy. 45. 10.1016/j.ijhydene.2020.02.150.

[39] Bezrukavnikov, O., & Linder, R. (2021b, January 14). A neophyte with automl: Evaluating the promises of Automatic Machine Learning Tools. arXiv.org. https://arxiv.org/abs/2101.05840

[40] BRILL, THANGIRALA, A., APHINYANAPHONGS, Y., CHEN, J., HU, E., KELLEHER, A. C., MARTIN, J., OEDING, J. F., OSTBERG, N., KATZ, G., BREJT, S., RAMANATHAN, R., & KAN, K. (2022). AUTOMATED MACHINE LEARNING WITH AUTOGLUON TO PREDICT POSTOPERATIVE PNEUMONIA USING THE AMERICAN COLLEGE OF SURGEONS' NATIONAL SURGICAL QUALITY IMPROVEMENT PROGRAM DATABASE. Chest, 162(4), A2595–A2596. https://doi.org/10.1016/j.chest.2022.08.2121

[41] Saxena, Abhinav & Goebel, Kai & Simon, Don & Eklund, Neil. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. International Conference on Prognostics and Health Management. 10.1109/PHM.2008.4711414.

[42] NASA (2007, January) . Prognostics Center of Excellence Data Repository. http://ti.arc.nasa.gov/projects/ data_prognostics, last accessed on January, 2007.

[43] Memon, Dr. O. (2022, November 15). *How much do jet engines cost?*. Simple Flying. https://simpleflying.com/how-much-do-jet-engines-cost/

[44] Kjelgaard, C., & Saathoff, M. (2023). *Aircraft Engine Maintenance Hub*. AvBuyer. https://www.avbuyer.com/articles/engine-maintenance-hub

[45] Memon, Dr. O. (2023, September 17). *Who are the High Flyers & How Do They Impact Private Aviation?*. Simple Flying. https://simpleflying.com/high-flyers-private-aviation-impact-analysis/

[46] Memon, Dr. O. (2023, March 29). *A Million-dollar industry: HOW MUCH DO major aircraft engine parts cost?*. Simple Flying. https://simpleflying.com/major-engine-parts-cost-guide/#:~:text=While%20numerous%20Life%20Limited%20Parts,to%20maintain%20optimal%20engine%20performance.&text=Each%20high%2Dpressure%20turbine%20disk,between%20%24200%2C000%20and%20%242%20million.

[47] Ellefsen, A. L., Bjørlykhaug, E., Æsøy, V., Ushakov, S., & Zhang, H. (2018, November 26). *Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture*. Reliability Engineering & System Safety. Retrieved April 7, 2023, from https://www.sciencedirect.com/science/article/pii/S0951832018307506

[48] Hong, Chang Woo & Lee, Changmin & Lee, Kwangsuk & Ko, Minseung & Kim, Dae & Hur, Kyeon. (2020). Remaining Useful Life Prognosis for Turbofan Engine Using Explainable Deep Neural Networks with Dimensionality Reduction. Sensors (Switzerland). 20. 1-19. 10.3390/s20226626.

[49] Hong, Chang Woo & Lee, Kwangsuk & Ko, Minseung & Kim, Jae-Kyeong & Oh, Kyungwon & Hur, Kyeon. (2020). Multivariate Time Series Forecasting for Remaining Useful Life of Turbofan Engine Using Deep-Stacked Neural Network and Correlation Analysis. 63-70. 10.1109/BigComp48618.2020.00-98.

[50] Benesty, Jacob & Chen, Jingdong & Huang, Yiteng & Cohen, Israel. (2009). Pearson Correlation Coefficient. 10.1007/978-3-642-00296-0_5.

[51] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211.

[52] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

[53] Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press.

[54] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

[55] Miguel, T. (2021, January 28). *How the LSTM improves the RNN*. Medium. https://towardsdatascience.com/how-the-lstm-improves-the-rnn-1ef156b75121

[56] Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023, August 22). *Dive into deep learning*. arXiv.org. https://arxiv.org/abs/2106.11342

[57] Ramasso, E. ., & Saxena, A. . (2014). Review and Analysis of Algorithmic Approaches Developed for Prognostics on CMAPSS Dataset. *Annual Conference of the PHM Society*, *6*(1). https://doi.org/10.36001/phmconf.2014.v6i1.2512

[58] Jin, Ruibing & Chen, Zhenghua & Wu, Keyu & Wu, Min & Li, Xiaoli & Yan, Ruqiang. (2022). Bi-LSTM based Two-Stream Network for Machine Remaining Useful Life Prediction. IEEE Transactions on Instrumentation and Measurement. 71. 1-1. 10.1109/TIM.2022.3167778.

[59] Qi, Xu, C., & Xu, X. (2021). AutoGluon: A revolutionary framework for landslide hazard analysis. *Natural Hazards Research*, *1*(3), 103–108. https://doi.org/10.1016/j.nhres.2021.07.002

[60] Dorogush, Anna & Ershov, Vasily & Gulin, Andrey. (2018). CatBoost: gradient boosting with categorical features support.

[61] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. (2017) LightGBM: a highly efficient gradient boosting decision tree. In

Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 3149–3157.

[62] Assimakopoulos, Vassilis & Nikolopoulos, K.. (2000). The theta model: A decomposition approach to forecasting. International Journal of Forecasting. 16. 521-530. 10.1016/S0169-2070(00)00066-2.

[63] Fiorucci, José & Pellegrini, Tiago & Louzada, Francisco & Petropoulos, Fotios & Koehler, Anne. (2016). Models for optimising the theta method and their relationship to state space models. International Journal of Forecasting. 32. 10.1016/j.ijforecast.2016.02.005.

[64] Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2019, October 19). *Deepar: Probabilistic forecasting with autoregressive recurrent networks*. International Journal of Forecasting. https://www.sciencedirect.com/science/article/pii/S0169207019301888

[65] Zeng, Ailing & Chen, Muxi & Zhang, Lei & Xu, Qiang. (2022). Are Transformers Effective for Time Series Forecasting?. 10.48550/arXiv.2205.13504.

[66] Nie, Y., Nguyen, N. H., Sinthong, P., & Kalagnanam, J. (2023, March 5). *A time series is worth 64 words: Long-term forecasting with Transformers*. arXiv.org. https://arxiv.org/abs/2211.14730

[67] Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2020, September 27). Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. arXiv.org. https://arxiv.org/abs/1912.09363

[68] Dietterich, T. G.(2000) Ensemble methods in machine learning. In International Workshop on Multiple Classifier Systems, pp. 1–15. Springer

[69] Breiman, L. (1996) Bagging predictors. *Mach Learn* **24**, 123–140. https://doi.org/10.1007/BF00058655

[70] Morde, V. (2019, April 8). *XGBoost algorithm: Long may she reign!*. Medium. https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-

may-rein-edd9f99be63d

[71] Yoav Freund and Robert E. Schapire. (1996) Experiments with a new boosting algorithm. In Proceedings of the Thirteenth International Conference on International Conference on Machine Learning (ICML'96). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 148–156.

[72] Ting, Kai & Witten, Ian. (1997). Stacking Bagged and Dagged Models. Fourteenth International Conference on Machine Learning.

[73] Caruana, Rich & Niculescu-Mizil, Alexandru & Crew, Geoff & Ksikes, Alex. (2004). Ensemble Selection from Libraries of Models. 10.1145/1015330.1015432.

[74] Feurer, Matthias & Springenberg, J.T. & Hutter, F.. (2014). Using meta-learning to initialize Bayesian optimization of hyperparameters. CEUR Workshop Proceedings. 1201. 3-10.

[75] McDonald, C., Erickson, N., & Becker, N. (2022, August 21). *Advancing the state of the art in Automl, now 10x faster with Nvidia gpus and Rapids*. NVIDIA Technical Blog. https://developer.nvidia.com/blog/advancing-the-state-of-the-art-in-automl-now-10x-faster-with-nvidia-gpus-and-rapids/

[76] BAMBANG PARMANTO , PAUL W MUNRO & HOWARD R DOYLE (1996) Reducing Variance of Committee Prediction with Resampling Techniques, Connection Science, 8:3-4, 405-426, DOI: 10.1080/095400996116848

[77] Thakkar U, Chaoui H. Remaining Useful Life Prediction of an Aircraft Turbofan Engine Using Deep Layer Recurrent Neural Networks. *Actuators*. 2022; 11(3):67. https://doi.org/10.3390/act11030067

[78] Li H, Wang Z, Li Z. An enhanced CNN-LSTM remaining useful life prediction model for aircraft engine with attention mechanism. PeerJ Comput Sci. 2022;8:e1084. Published 2022 Aug 30. doi:10.7717/peerj-cs.1084