Spring 4-29-2022

# Data-driven Models for Remaining Useful Life Estimation of Aircraft Engines and Hard Disk Drives

Austin Coursey
*Murray State University*

# Data-driven Models for Remaining Useful Life Estimation of Aircraft Engines and Hard Disk Drives

Submitted in partial fulfillment of the requirements for the Murray State University Honors Diploma

Austin Coursey

May 2022

# Abstract

Failure of physical devices can cause inconvenience, loss of money, and sometimes even deaths. To improve the reliability of these devices, we need to know the remaining useful life (RUL) of a device at a given point in time. Data-driven approaches use data from a physical device to build a model that can estimate the RUL. They have shown great performance and are often simpler than traditional model-based approaches. Typical statistical and machine learning approaches are often not suited for sequential data prediction. Recurrent Neural Networks are designed to work with sequential data but suffer from the vanishing gradient problem over time. Therefore, I explore the use of Long Short-Term Memory (LSTM) networks for RUL prediction. I perform two experiments. First, I train bidirectional LSTM networks on the Backblaze hard-disk drive dataset. I achieve an accuracy of 96.4% on a 60 day time window, state-of-the-art performance. Additionally, I use a unique standardization method that standardizes each hard drive instance independently and explore the benefits and downsides of this approach. Finally, I train LSTM models on the NASA N-CMAPSS dataset to predict aircraft engine remaining useful life. I train models on each of the eight sub-datasets, achieving a RMSE of 6.304 on one of the sub-datasets, the second-best in the current literature. I also compare an LSTM network's performance to the performance of a Random Forest and Temporal Convolutional Neural Network model, demonstrating the LSTM network's superior performance. I find that LSTM networks are capable predictors for device remaining useful life and show a thorough model development process that can be reproduced to develop LSTM models for various RUL prediction tasks. These models will be able to improve the reliability of devices such as aircraft engines and hard-disk drives.

# Contents

# List of Figures

# List of Tables



# List of Algorithms

# 1 Introduction

Physical devices can fail. Whether it is a car engine, medical device, television, or one of countless other devices, there is an aspect of unreliability to it. Some devices are components of a larger system. For example, consider a car engine. It is part of the larger system of a full car. When smaller components fail, an entire system can fail. This is clearly true for car engines. Vehicles will not run without a working engine.

The shocking aspect of device failure is not that it happens, but rather the frequency in which it happens. For example, Warranty Direct's 2013 car engine reliability report [2] found that MG Rover cars had a 1 in 13 chance of engine failure. Similar numbers can be seen in popular car brands still operating today, too. According to the same report, Audi cars have an engine failure rate of 1 in 27. With millions of vehicles produced, engine failure will impact an extremely large number of people. Frequent failure is not limited to car engines. According to a 2020 report by Mindfactory [54], a German retailer, around 493 of the 9,530 2080 Ti computer graphics processing units were returned. This is a return rate of around 5%. With a current manufacturer's suggested retail price of around $1,000 [51], hundreds of thousands of dollars are lost to faulty GPUs being returned.

Devices failures have many negative consequences. First, a device failure leads to an inconvenience for the device's user. GPUs are an essential computer component to many researchers, video game players, or anyone else needing

to process computationally-intensive graphics. If their GPU fails, and there is up to a 1 in 20 chance it will according to the above statistics, their work or lifestyle is interrupted while waiting for their return. Imagine any frequently-used device in someone's life. An air-conditioning unit, a refrigerator, or a smartphone camera. Every device would cause some sort of inconvenience in its user's life if it failed. Second, device failures lead to large amounts of money lost by manufacturers and users. As previously mentioned, hundreds of thousands of dollars are lost to GPU failures by manufacturers on just one model of a GPU. That only considers the GPUs that were returned by users. It is easy to imagine scenarios where users would choose not to return their failed GPU, causing consumers to lose thousands of dollars too. Another example of device failure causing monetary loss can be seen in smartphones. According to a 2017 report by Blancco [70], 26% of iPhone 6s had a reported failure. At the time, the iPhone 6 was the most commonly used model with 21% of the 728,000,000 active iPhones being iPhone 6s. This means that around 40,000,000 iPhone 6s would have had some form of reported failure. With smartphones being as expensive as they are, the failure of these iPhones undoubtedly cost consumers massive amounts of money in upgrade or repair fees. Finally, device failures can cause physical harm. Consider the 2018 engine failure of Southwest Airlines Flight 1380. The left engine of the aircraft failed, causing a window to be damaged. One passenger died, and eight others were injured [41]. It is clear that the reliability of physical devices serves an important purpose to those who interact with them.

To improve the reliability and maintenance of physical devices, it would be helpful to know how long a device has before it is unusable. The amount of time a device has before it fails or is unusable is known as the remaining useful life (RUL). "In engineering, prognostics can be defined as the process of RULs estimation of system/subsystem/component that is degrading due to either normal operation (no fault symptoms) or detected fault" [28]. By being able to estimate the remaining useful life of a device, we can predict when it will fail. This can help minimize the damages caused by a failure, as users or maintainers will have a warning before failure. The approaches taken towards prognostics can be split into two categories: model-based and data-driven approaches.

## 1.1 Model-Based Approaches

Model-based remaining useful life prediction uses models of physical systems to depict the lifetime of a device or system [7]. An accurate, mathematical model of the degradation towards failure of the device is required. Once such an equation is obtained, deviations from this model can be measured and checked against a threshold. If the sensor measurements exceed the threshold, the range of error in predictions becomes larger [48]. In these models, parameters are estimated using various algorithms, such as a Rao-Blackwellised particle filter [43]. An example of a model-based prognostics approach is in battery degradation [7]. The authors use the following equation to model the degradation of a battery.

$$\lambda = ae^{-bt} \tag{1}$$

In this equation, $\lambda$ is the battery's performance, $t$ is the time, and $a$ and $b$ are parameters that need to be estimated. They set a true $a$ and $b$ to generate data. They allow $a$ to be known as a value of 1. Then they use a particle filter to estimate $b$ using the data. Using this estimated value of $b$, they are essentially fine-tuning a degradation model to model battery performance with their assumed model parameters. They can then use the predicted value of $b$ for each value of $t$ until $\lambda = 0$ so that they have a model of the lifetime of the battery.

There are some advantages to using model-based remaining useful life prediction approaches. Model-based approaches can model the physical system where RUL prediction is needed. Model-based approaches use physics to predict how the lifetime of a device will degrade. If the models are perfect, then it would perfectly model the lifetime of the device in the real world. Additionally, model-based approaches can often determine which features led the device to degrade [48]. This would allow the engineers designing these devices to know where the devices can be improved.

An obvious disadvantage to model-based approaches is that they require an accurate model of the physical system. If one is not available, it would require a deep understanding of the device or system being modeled and the physics behind it to create such an equation. And, in the end, there

are assumptions and predictions being made in the model that cause it to be imperfect. In our battery example, this was the assumption of the value of $a$ and the estimation of $b$. This model may not depict the degradation of a battery perfectly, leading to incorrect RUL predictions or a high error window. For this reason, it is worth looking at other remaining useful life prediction methods. such as data-driven approaches.

## 1.2 Data-Driven Approaches

Often, physical devices have sensors that are constantly collecting data about the device. For example, hard-disk drives have S.M.A.R.T. features reported by the drive at all times [56]. With many devices, these features can be aggregated to form a dataset of historic device data. If failures are marked, the run-to-failure of each device in that dataset can be seen. Data-driven prognostics use this data to build a model of the device's remaining useful life [49].

The benefit of using a data-driven technique for remaining useful life prediction comes from the ease of building a model. No in-depth physics knowledge or accurate mathematical model is needed. All one has to do to use a data-driven approach is implement some method of data-driven model building. There are a variety of techniques to create such models, some of which will be discussed in the next section, and these models are not specific to one problem. Someone could develop a data-driven model that works on data

from smartphones and is easily transferred to another type of device, such as car engines, with little modification [28].

A downside to data-driven techniques is that they require large datasets for the devices in order to model failure [29]. It is easy to imagine an example where this is hard to obtain. For example, if someone wanted to predict the remaining useful life of a new type of aircraft, there would be no available data on this aircraft. In this case, it might be more appropriate to create a model-based representation of the aircraft to predict the RUL. However, in situations where large datasets are available, data-driven techniques for prognostics are very powerful. Additionally, these large datasets are becoming more feasible to work with and collect as computational power increases. Consequentially, data-driven approaches are becoming more popular [71]. Due to the increase in popularity, transferability, and relative simplicity of data-driven models, I choose to explore their use for RUL estimation throughout the rest of this thesis.

It is worth noting that hybrid data-driven and model-based approaches exist. These can be more powerful than either of the two on their own [20]. However, they still require physical modeling of the system or device, so I will not explore these techniques further.

## 1.3 Machine Learning

There are two main categories of data-driven approaches: statistical and machine learning approaches. Statistical techniques involve building a statistical model of the given data to predict RUL. These statistical models may be some kind of time series analysis model, like an ARIMA model. The discussion of such statistical models is kept brief and included as needed, as machine learning models have shown superior performance [64].

Machine learning is a rapidly-growing field that involves using computer algorithms that can "learn" from data. It is a subfield of artificial intelligence. Learning in this context can be thought of as an algorithm improving its performance by training on some task [38]. Once this algorithm has been trained on data, it should be able to make reasonable predictions on its given task. For example, if someone wanted to create a machine learning model that can learn to detect spam emails, they would likely start by collecting hundreds or thousands of emails. They would train some kind of machine learning model on these emails. The model would start its learning process incorrectly discriminating the spam emails from the real emails. Eventually, the model would start to improve its performance and the model user could implement this model in a spam-detection application. There is a large overlap between statistical and machine learning modeling. They both involve creating a model to learn trends from data. However, statistical models often have a focus on inference, being able to determine truths about the population from

the data. In contrast, machine learning models often focus on the predictive performance of models [18]. There are three main categories of machine learning: supervised, unsupervised, and reinforcement learning [38].

### 1.3.1 Categories of Machine Learning

Supervised learning is the most widely-used machine learning method [38]. It involves training a model on labeled data [45]. This means that the true value that one is trying to predict is already known. The model can see how far off its predictions are when learning. Supervised tasks, and tasks in other categories of machine learning, can usually be placed into the categories of regression and classification [42]. Regression tasks involve predicting a continuous number. An example of a supervised regression task may be predicting a stock price given historical stock data. The model will have access to the true values of the historical stock prices, making it supervised, and the model has the goal of predicting a continuous value, the future stock price. Therefore, this model would be a supervised regression model. The second major type of supervised prediction is classification. Classification is, as the name suggests, grouping data points into classes. For example, consider a dataset of labeled images of cats and dogs. The model has the true animals that the images represent, and is therefore supervised. The model also has the goal of placing the images into the categories of a cat or a dog, so it is a classifier. There are many popular supervised learning algorithms. Some include Logistic Regression, Decision Trees, Random Forests, Support

Vector Machines, and Neural Networks [65].

Unsupervised learning is another category of machine learning. Unsupervised learning involves learning patterns from unlabelled data. The algorithms often group data by patterns or analyze unlabelled data based on the structural properties of the data [38]. For example, consider a website for online shopping. An unsupervised model could group customers by purchase type so the website can recommend users products they may want to buy. The true label of the type of products a customer may buy could be unknown or unknowable. In either case, the label is not needed to group the customer data by similarity. Some common types of unsupervised learning include K-Nearest-Neighbors, Principal Components Analysis, and Singular Value Decomposition [27].

The final major category of machine learning is reinforcement learning. Reinforcement learning algorithms have an agent that is learning to interact with its environment. The agent performs actions in the environment to receive rewards. The better an agent is doing at its job, the more reward the agent gets [68]. This agent has the goal of maximizing its reward. This is very similar to how humans learn. As an example, consider a robot trying to learn to walk. The robot would have a constant stream of data from all of its sensors. When the robot first tries to walk, it will likely not succeed. It will receive a very small reward for taking no steps. This robot will experiment with different actions until it stumbles (pun intended) upon the correct series of actions it needs to take to make a step, gaining a large reward. As it

maximizes its rewards, it also learns to walk. We did not provide this robot with any labels, but we also did not leave it completely unsupervised; we defined what we would reward it for doing correctly.

With those categories in mind, let us consider the problem of remaining useful life prediction again. In remaining useful life prediction, we have data from physical devices. Hopefully, this data contains a mark for when that device fails. We can use this mark to calculate the true remaining useful life. This remaining useful life will probably be a discrete numerical value. Therefore, we should use a supervised algorithm that can perform classification. (It should be noted that regression predictions can be rounded to an integer to mock multiclass classification, so regression will work too.)

### 1.3.2 Deep Learning and Neural Networks

In supervised learning, a sub-field of machine learning has developed known as deep learning. Deep learning networks are models that contain multiple layers of neural networks that update their parameters with optimization algorithms [38]. Various deep learning architectures have shown great performance in the task of prognostics [76]. This is supported by the number of prognostics papers using deep learning models in comparison to those using traditional statistical or machine learning models. For example, in the problem of aircraft engine remaining useful life prediction, the top four categories of papers published working with the NASA Commercial Modular Aero-Propulsion System Simulation dataset used some form of neural-network-based deep

learning, totaling 64 out of the 81 papers [72]. To understand how deep learning models work, we need to start with understanding a neural network.

A neural network algorithm is a representation of the human brain. A basic artificial neural network contains neurons that are either active or inactive. An output is determined by adding weighted inputs. This output is put through an activation function to determine if the neuron is active or not. Many neurons are connected into a network, called an artificial neural network. The activation of neurons throughout a network can approximate nonlinear functions [50]. Figure 1 demonstrates a basic neuron in a neural network. In that figure, $w_{ij}$ is the weight of the connection between the current neuron, $i$, and the previous connected neurons, $j$. The state of a neuron is denoted as $s_j(t)$ and is computed with an activation function. In this diagram, the activation function is $g$, and the specific function used depends on the problem. The argument of the activation function is a weighted sum of the inputs that have a bias term, $\theta$ [50]. The equation below can be used to determine the state of a neuron at time $t+1$ using a given activation function $g$.

$$s_i(t+1) = g(\sum_{j=1}^{N} w_{ij} s_j(t) - \theta_i) \tag{2}$$

A neural network is the connection of these neurons. There is typically an input layer, a "hidden layer" of neurons, and an output layer that are all fully connected. There could be multiple hidden layers. These could be used to

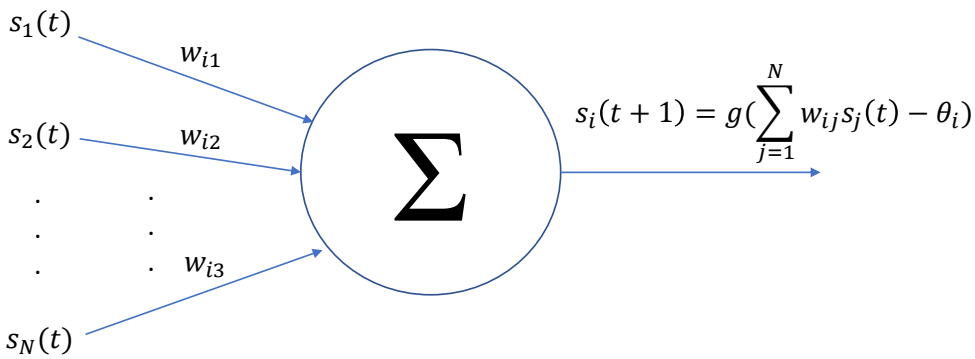



Figure 1: Diagram of basic neuron in a neural network.

learn more complex features. For example, layer one may learn what shapes are present in an array of pixels while layer two learns the relationship among those shapes. As the number of layers increases, the neural network can learn more complex functions. An example of a simple neural network can be seen Figure 2.

After a prediction is made, the neural network can calculate an error. The error function can be differentiated to calculate a gradient. This gradient is used to perform stochastic gradient descent and propagate weight updates backward through the network (backpropagation) [50]. This is what allows a neural network to learn. It makes a prediction and changes its weights to correct for its mistake. After doing this thousands or millions of times, it can approximate the universal function that represents the true relationship between the features and predicted variable.

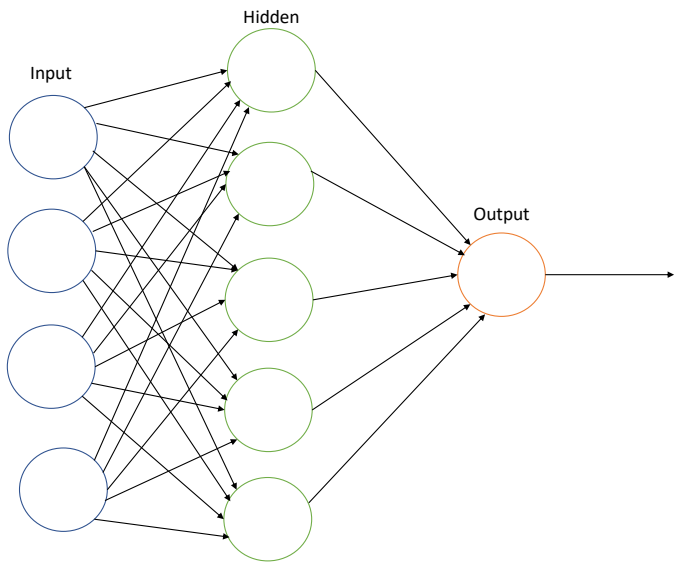



Figure 2: Artificial neural network. Each node is a neuron.

Neural networks are organized into different architectures to perform various tasks. Recall that in device health data, we have a collection of sequential, time-dependent data points. Each data point depends on the previous data point. There is a type of neural network made for making predictions on sequential data known as a Recurrent Neural Network.

### 1.3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are neural networks that can use previous outputs as inputs. They are usually thought of as having different cells that are connected into a network. These cells can activate and produce an output. A diagram of a basic RNN is shown in Figure 3. In this diagram, $a^{[t]}$ is the activation of a cell at a given time $t$, $x^{[t]}$ is the input at a given time, and $y^{[t]}$

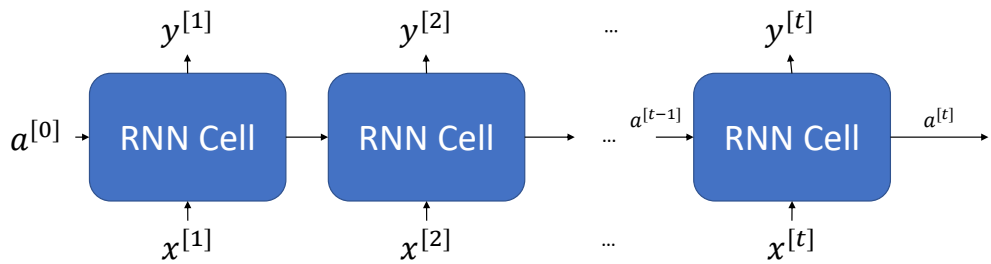



Figure 3: Traditional many-to-many RNN.

is the output at a given time. This RNN is many-to-many, meaning that it takes multiple inputs and produces multiple outputs [6]. For the problem of remaining useful life estimation, we only need to produce one output, the RUL.

The activation of a cell $a^{[t]}$ and the output of a cell $y^{[t]}$ can be defined by the following equations. This these equations, $g_1$ and $g_2$ are activation functions and $W_{ax}$, $W_{aa}$, $W_{ya}$, $b_a$, and $b_y$ are coefficients that are shared among cells temporally to allow the network to remember information [6].

$$a^{[t]} = g_1(W_{aa}a^{[t-1]} + W_{ax}x^{[t]} + b_a) \tag{3}$$

$$y^{[t]} = g_2(W_{ya}a^{[t]} + b_y) \tag{4}$$

The traditional backpropagation algorithm that was previously described cannot be used in a Recurrent Neural Network, so RNNs have a custom loss

function, $\mathcal{L}$, that is the sum of the loss at every time step and a backpropagation method (called backpropagation through time) that can be expressed by the following equations [6].

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{[t]}, y^{[t]}) \tag{5}$$

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T_y} \frac{\partial \mathcal{L}^{(T)}}{\partial W}\Big|_{(t)} \tag{6}$$

In these equations, $\hat{y}^{[t]}$ is the predicted value at a time point, $y^{[t]}$ is the true value at a time point, $T$ represents the number of time points, $t$ is an individual time point, and $W$ is the weight matrix. Backpropagation will be done at each time point using the partial derivative in the second equation above. Recurrent neural networks have shown success in tasks involving sequential such as text classification [44] and time series forecasting [69]. However, they suffer from something known as the vanishing gradient problem [33].

The gradient is calculated by taking a partial derivative of the loss function. This partial derivative can be followed to minimize the loss function. In a RNN, the loss function is calculated for each input in the data. The calculation of the loss function involves the multiplication of each previous weight. Therefore, if an activation function that scales values from 0-1 is used, the network will be multiplying numbers from 0-1 together. This causes the gradient to shrink in backpropagation through time as the number of nodes

or layers increases. In certain cases, a small gradient can keep a network from learning [33]. To solve this problem, RNNs were modified to introduce the Long Short-Term Memory Network [34].

### 1.3.4 Long Short-Term Memory Network

The LSTM introduces a cell state $c_t$ that contains a series of gates that have more control over the information that is retained between cells. The LSTM has a forget $f_t$, input $i_t$, and output $o_t$ gate. The combination of these allows the LSTM cells to extend their short-term memory, keeping any information they need through the learning process. Each gate contains activation functions like sigmoid. The forget gate takes information from the previous cell and current input to decide what to keep or forget. Any information that is kept goes through the input gate. This determines what values will be updated in the cell. The *tanh* function is applied to the cell state and current input for regulation. The cell state is then updated according to the combination of forget and input gates. Using the current cell gates and state, the output gate decides what to pass on to the next cell. A diagram to demonstrate a typical LSTM cell is shown in Figure 4. Equations [58] that describe this process are shown below.

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i) \tag{7}$$

$$f_t = \sigma(W_{fx}x_t + W_{fm}m_{t-1} + W_{fc}c_{t-1} + b_f) \tag{8}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + b_c) \tag{9}$$

$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_t + b_o) \tag{10}$$

$$m_t = o_t \odot h(c_t) \tag{11}$$

$$y_t = \phi(W_{ym}m_t + b_y) \tag{12}$$

These equations describe one iteration of the network unit activations. These are calculated $T$ times. In these equations, $x$ denotes an input vector of size $T$, $y$ denotes an output sequence of size $T$, $W$ denotes the weight matrices, $b$ denotes the bias vectors, $i$ is the input gate, $f$ is the forget gate, $o$ is the output gate, $c$ is the cell activation vector, $m$ is the cell output activation vector, $\odot$ is the element-wise product of two vectors, $g$ is the cell input activation function, $h$ is the cell output activation function, and $\phi$ is the softmax network activation function [58].

With a model that can learn from sequential data without suffering from

the vanishing gradient problem established, we can begin experimenting with LSTMs. The next two sections are dedicated to demonstrating the effectiveness of LSTMs for remaining useful life prediction of physical devices through two real-world applications. First, in hard-disk drives and second in aircraft engines. By showing these experiments, I provide reproducible model development methods that can be transferred to other RUL prediction problems or to similar tasks. I achieve state-of-the-art performance in problems in device RUL prediction, offering real-world solutions to the pressing problem of device failure.

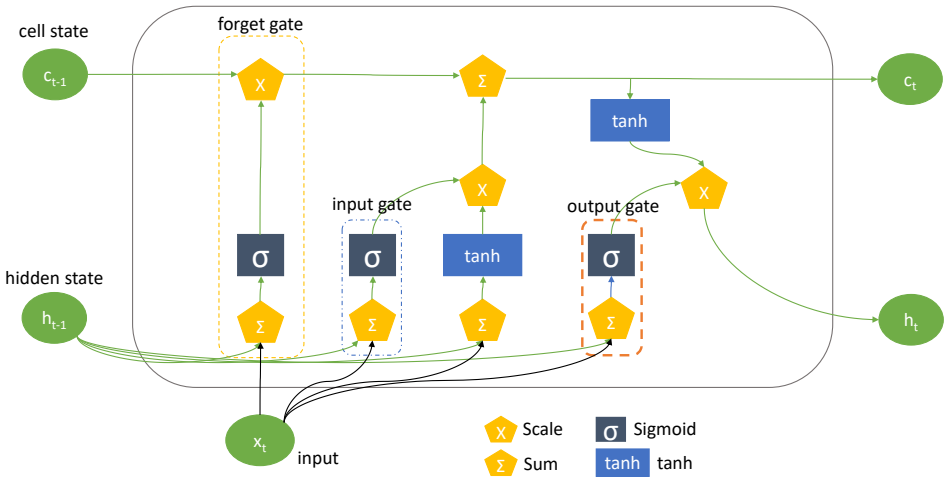



Figure 4: The Structure of an LSTM Cell

# 2 Hard-Disk Drive RUL Prediction

The first application of long short-term memory networks to predict remaining useful life that I performed was in hard-disk drives. This section contains work that was published in [25], in which I am the primary author.

## 2.1 Introduction

Hard disk drives or HDDs have been the primary storage systems in large-scale data centers. Like any electronic device, HDDs also have a limited lifespan. To monitor operational health of these devices, operators frequently rely on S.M.A.R.T. (Self-Monitoring Analysis and Reporting Technology). S.M.A.R.T. logs device health data such as power-on hours, temperature and sector error rates, head flying hours and so on. [56]. These are set by each drive manufacturer along with a threshold for each attribute which is traditionally used to detect failure. If a hard drive is operating as it should, each S.M.A.R.T. attribute should ideally not be outside this threshold.

Backblaze, a cloud storage and data backup company, has over 171,000 hard drives in their data center [1]. With an annualized failure rate of 0.93% in 2020, over a thousand of their hard drives failed [39]. It follows that it is necessary for hard drive manufacturers and companies like Backblaze to know the remaining useful life (RUL) of their hard drives. This would not only help reduce downtime at a large data center, but would help protect valuable user data.

S.M.A.R.T. attributes are a great way of detecting imminent failure of hard drives [14], but it would be ideal to predict the RUL of a hard drive. This would allow consumers to know that they need to start making backups and initiate planning to buy a new hard drive long before it fails. It would also allow companies like Backblaze to prepare to replace a drive ahead of time.

**Contributions:** In this work, we propose methods for data standardization, normalization and RUL prediction for working with the highly class-imbalanced Backblaze data using a Bidirectional Long Short Term Memory Network [30] with multiple days of look-back period. Our approach considers S.M.A.R.T. features which are highly correlated to failure and builds a prediction pipeline that takes into consideration the long-term temporal relations in the failure data. We employ a vanilla Long Short Term Memory Network under similar data preprocessing conditions to contend that the Bi-LSTM outperforms the standard LSTM implementation on all lookback periods. We also consider a Random Forest baseline which upon inspection is found to generate sub-par learning capability on the failure data thereby reinforcing the need for learning temporal sequence patterns as our Bi-LSTM model does. At an accuracy of 96.4% for a 15 day look-back, our model is among the state-of-the-art reported in the literature. An overview of our approach is shown in Figure 5.

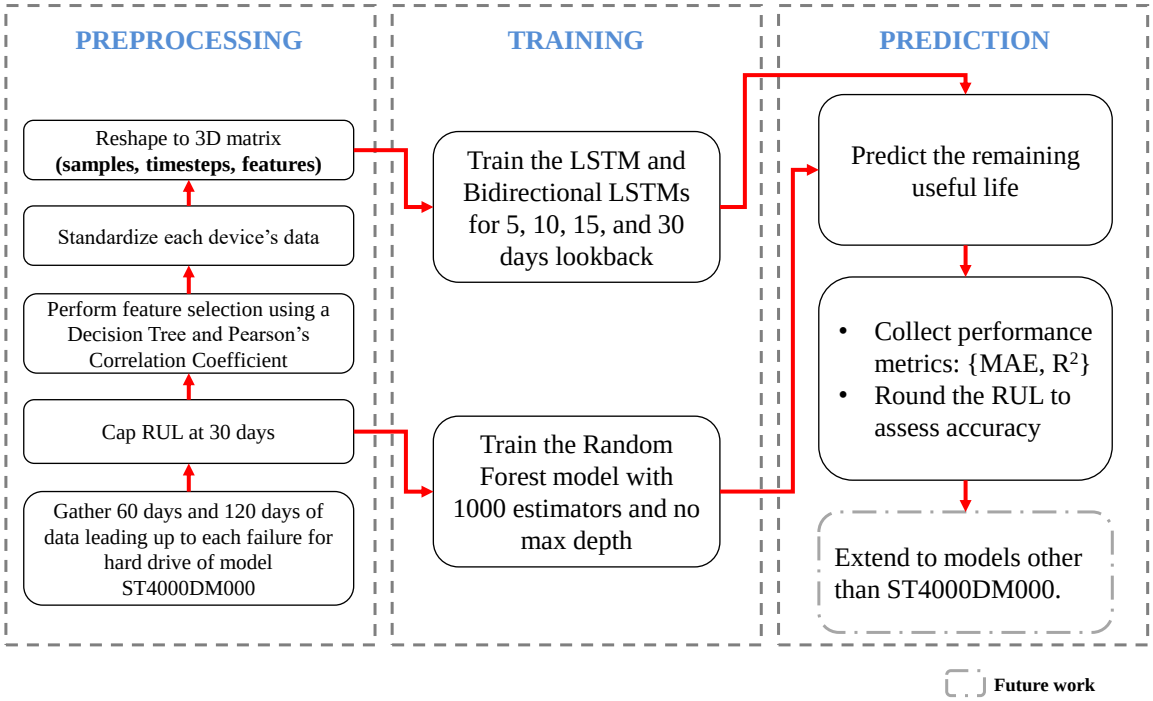



Figure 5: Overview of the workflow for our approach.

## 2.2 Previous Approaches

In this section, we review some of the previous works focused on remaining useful life prediction of HDDs. For ease of review, we have divided these into statistical and machine learning approaches.

### 2.2.1 Statistical Approaches

Remaining useful life prediction is inherently a time series problem. The S.M.A.R.T. features of each hard drive are recorded at a constant time interval. Each S.M.A.R.T. attribute value is dependant on the previous value.

This lack of independence makes this problem difficult to solve with simple statistical models such as linear regression. This section will briefly cover a few statistical approaches historically taken. For more details, the reader is encouraged to see the cited papers.

One such approach (Wang et al.) involved using a two-step parametric method. The first step was transforming the variables into Gaussian variables with the Box-Cox transformation, using Mahalanobis distance to get the variables into one index. This was used for anomaly detection. Second, a generalized likelihood ratio test with a sliding window was used to predict failure. This method resulted in around a 68% failure detection rate [74].

Another approach (Wang et al.) used an adaptive Rao-Blackwellized particle filter error tracking method. This gave a health status to the HDD. The failure was predicted by using a threshold that is placed by the adaptive error tracking. Using an accelerated degradation test, they were able to achieve a 97.44% failure detection rate [75]. It should be noted that this approach was used to predict whether a hard disk drive would fail, not the RUL of an HDD.

A third approach (He, Yang and Xie) proposes using the Weibull distribution instead of the exponential distribution to predict RUL. Using sectional Weibull modeling can better capture the nuances of the HDD time-to-failure distribution [32].

The above approaches rely heavily on a deep understanding of specific sta-

tistical techniques to predict impending failure. Model-driven approaches are highly pointed and they do not always end up learning the complex, underlying patterns in unstructured failure data with class imbalance.

### 2.2.2 Data-driven Approaches

In recent years, there has been an exponential rise in the adoption of data-driven approaches for fault prediction tasks. The solution frameworks involving the (a) prediction of RUL of an HDD instance and (b) classifying whether an HDD instance will fail within a certain number of days have both seen significant improvements with the adoption of machine learning and and deep learning approaches. This section will briefly survey some machine learning methods widely used in HDD RUL prediction.

A Bayesian Network is "a probabilistic graphical model that represents a set of random variables and their conditional dependencies" [22]. These use the S.M.A.R.T. attributes to predict the probability that a hard drive will fail at a particular time. Simple (Chaves et al.) [22, 21] and Combined Bayesian Networks (Jia et al.) [52] have been used to predict the RUL. The Combined Bayesian Network is able to combine multiple machine learning classifiers to give a model that can predict more accurately than an individual classifier.

Relatively-straightforward machine learning models have also had some success in solving this problem. Machine learning algorithms such as Support Vector Machines (SVM), Decision Trees (DT), and Random Forests (RF) are

frequently used. These algorithms are often used as a baseline or control to compare a proposed method to [35], as they are known to be well-performing. However, RF algorithms have shown to predict the RUL very well, and are commonly treated as more than a baseline [62, 8, 9]. Random Forest algorithms do not require a sequence of time data, so the input of current S.M.A.R.T. attribute values would be enough to predict the RUL [8]. This means that RF approaches need less data, offering an advantage over some of the time-dependant models discussed later.

By far, the most common method to predict the RUL of hard drives involves using Long Short-Term Memory networks (LSTMs). These are an improvement on Recurrent Neural Networks that can help solve the vanishing gradient problem [34]. LSTMs [9, 12, 61], Convolutional Neural Network LSTMs [47], Clustered LSTMs [13], and Attention LSTMs [23, 73] have all been used to predict hard drive failure or the RUL of a hard drive with a high degree of success.

A convolutional neural network LSTM (CNN-LSTM) is able to combine the benefits of both CNNs and LSTMs into one model. The CNN uses convolutional layers to reduce variation in the input. This filtered input is then passed to the LSTM, giving the LSTM better input than without the CNN. This combined model could give better results than a simple CNN or LSTM would alone [47] [57].

An encoder decoder LSTM, also known as a sequence-to-sequence model, uses

two models. The first model encodes the input sequence to a fixed length. The second model decodes that fixed length input and predicts a sequence of output [24]. Remaining useful life prediction does not necessarily call for sequence prediction. All we need to predict is one number, the RUL. However, encoder decoder models have shown great performance and are often the base to an attention LSTM, so it is worth mentioning.

## 2.3 Bidirectional Long Short-Term Memory Network

Numerous variants of LSTMs have been introduced to improve performance. We use one such variant - the Bidirectional LSTM or Bi-LSTM - in combination with specialized normalization strategies to predict and improve upon the HDD RUL as predicted by vanilla LSTMs.

A bidirectional LSTM is a variant of an LSTM that consists of two LSTMs. These LSTMs run at the same time. One runs on the input sequence and the other runs backwards on the input sequence [31]. In this way, the LSTM runs in both directions. In this problem, one could think of one direction of the LSTM running on the sequence of hard drive data leading up to failure and another running on the sequence as the hard drive gets further away from failure. This allows the LSTM to better learn the relationship between the features and the remaining useful life with a simple, low-cost architecture change. For this reason, we experiment on effectiveness of a bidirectional LSTM in determining the remaining useful life of hard-disk drives. Figure 6

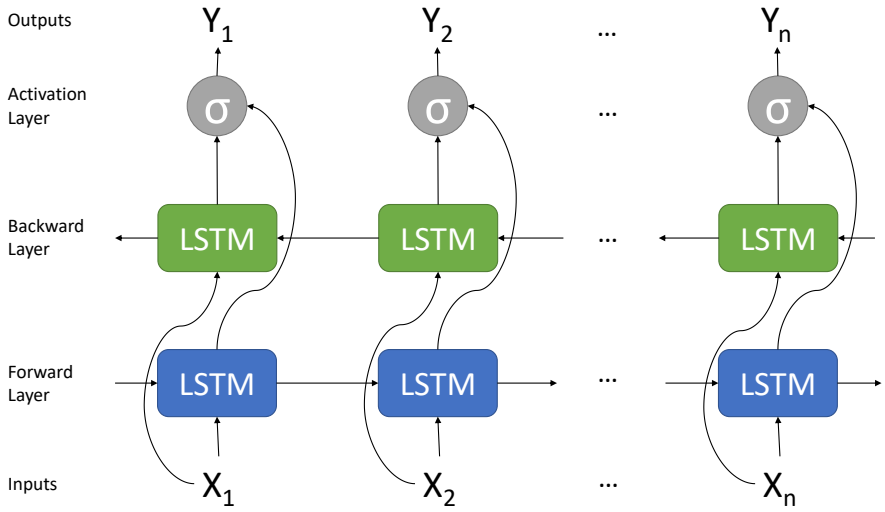



Figure 6: Bidirectional LSTM Architecture

shows the architecture of a bidirectional LSTM.

## 2.4 Data Preprocessing

Backblaze is a cloud storage company that has over 170,000 HDDs used by customers across the globe. Each quarter, they release snapshots of all of their hard disk drives [1]. In these snapshots, there are daily statistics for each HDD including S.M.A.R.T. features and a few other features for each HDD. Among those is whether or not a HDD failed on a given day. If it failed, it it marked with a 1 and removed from the subsequent snapshots.

### 2.4.1 Data Collection

Using the Backblaze data from quarter 4 for 2019 and quarters 1 and 2 for 2020, we programmatically created a dataset containing the S.M.A.R.T. features leading up to failure for one hard drive model. Only one model was selected due to the inconsistency in meanings of S.M.A.R.T. features between models. The model selected was the Seagate ST4000DM000. We target this model because of the high prevalence of disk failures compared to other models.

This selection also allows for comparison with several existing state-of-the-art approaches in the literature such as Basak et al. [13] and Anantharaman et al. [8]. Basak et al. used a spatio-temporal approach to predict the RUL of HDDs combining LSTM and hierarchical clustering. They achieved a mean absolute error (MAE) of 2.4 days for the prediction on Seagate ST4000DM000. Anantharaman et al. used two approaches: (a) a Random Forest model using the current snapshot of S.M.A.R.T. readings and (b) an LSTM which models the historical temporal pattern of the S.M.A.R.T. features. Their results suggest the Random Forest predictions are comparable to or outperforms the LSTM on Seagate ST4000DM000.

In order to create our dataset, we first looked for a hard drive with the model ST4000DM000 that had failed. When one was found, the previous 60 and 120 days worth of data were gathered for that same serial number. This was then concatenated onto the dataset in the long format. For each day back,

a column for the remaining useful life was also added. For example, if a failed device was found, the column for RUL would have a value of 1 on the previous day, 2 on the day before that, and so on. This allowed us to keep from having to treat this as a binary classification problem. We could instead predict the RUL value using regression.

We chose to take a regression approach to this problem because we felt it remained true to the goal of the model. With regression, the model will be able to output a continuous number representing the predicted amount of days until the hard drive fails. Since the data is captured each day, this predicted number could be rounded or truncated. This will lead to a simple way to assess the performance of the model. One approach that was considered for binary classification was to iterate over each day, essentially asking the model the question "Will the hard drive fail on this day?" One could then use the first day where the model gives a value of 1 (hard drive failing) as the remaining useful life, capturing classification statistics along the way. We suspect this is how many previous approaches to RUL prediction handled this dataset, as most statistics provided in the literature are classification statistics (F Score, Precision, Recall, etc.).

### 2.4.2 Feature Selection

The Backblaze dataset provides dozens of S.M.A.R.T. statistics for each of its hard drives; however, not all of these are relevant. Some of the features are redundant, do not contain useful information, or are null. To remove

these and reduce the complexity of the model, we used two feature selection methods. These were a correlation score and a Decision Tree.

Before using any feature selection techniques, we removed all null features and features that were already normalized. Backblaze performs their own data normalization and provides raw and normalized versions of each feature reported. We chose to apply our own normalization techniques, discussed later, and therefore removed all of these features.

The first method of feature selection we chose was using a correlation score. We calculated the Pearson product-moment correlation coefficient between each S.M.A.R.T. feature and the feature representing the remaining life of the hard drive. This was calculated for each hard drive instance, averaged, and then the absolute value was taken. This left us with a correlation score for each S.M.A.R.T. feature, shown in Figure 7. As can be seen, 5 of the features are highlighted in green. These indicate the features that we selected as the final predictors. S.M.A.R.T. 7 was included over S.M.A.R.T. 193 because it was used as a predictor in [13] and we wanted to make sure a fair comparison was possible. We suspect that 193 had a higher correlation score in our analysis than S.M.A.R.T. 7 due to differences in data, but the scores of S.M.A.R.T. 7 and S.M.A.R.T. 193 were practically the same.

With concerns about nonlinearity of S.M.A.R.T. feature values, we conducted another method of feature selection. We trained a simple Decision Tree regressor on the dataset using all of the features. A graph demonstrating the

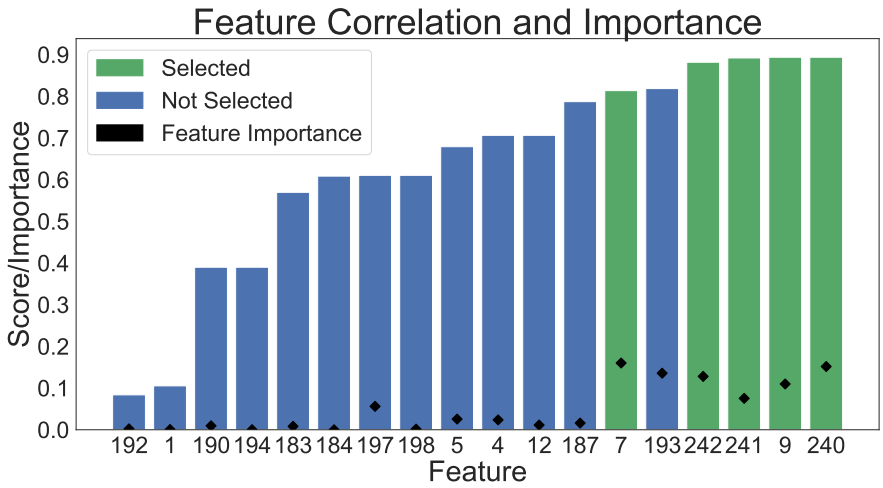



Figure 7: Pearson Correlation Score for S.M.A.R.T. Features as bars, Feature importance from Decision Tree as diamonds.

importance it gave to each feature is shown on top of the bar plots in Figure 7. The diamonds represent the importance percentage of the respective feature determined by the decision tree. The sum of each of these importances is equal to one. As can be seen, the decision tree feature importance agrees with the correlation scores. S.M.A.R.T. 7, 240, 193, 242, 9, and 241 are given the most importance and highest correlation scores. Because of this, we felt comfortable taking S.M.A.R.T. 7, 9, 240, 241, and 242 as the features for the LSTM models. As discussed, these are the same features reported in [13], which will allow for more comparability in model performance.

A brief description of what each selected feature represents [56] is provided below.

**S.M.A.R.T. 7**

**Seek Error Rate**. The rate of seek errors of the magnetic head.

**S.M.A.R.T. 9**

**Power-on Hours**. The total number of hours the hard drive has been powered on.

**S.M.A.R.T. 240**

**Head Flying Hours**. The amount of time a disk head has spent moving.

**S.M.A.R.T. 241**

**Total LBAs Written**. The total number of LBAs written by the hard drive. An LBA is a 512 byte section of memory.

**S.M.A.R.T. 242**

**Total LBAs Read**. The total number of LBAs read by the hard drive.

It seems to make intuitive sense that each of these features would be highly correlated with hard drive failure.

### 2.4.3 Data Standardization

A typical method of standardization would involve scaling all features of all of the data points to some standard range. When done on this data, this led to a few points being more extreme than others (see Figure 8). As can be seen, a few of the points have values above 10. The standardization makes the points have a mean of 0 and variance of 1. All of the points should ideally be somewhere between -2 and 2 to have an even distribution. The points that are extremely far outside of this range may inhibit the LSTM's training and reduce performance.

In order to fix this, a different approach to standardization was taken. Instead of standardizing each feature from every hard drive (the full dataset), each individual hard drive was standardized independently of the other. Each feature from each hard drive was scaled such that they would have a mean of 0 and variance of 1. This heavily reduced the impact of the extreme data points. However, this did come with a cost. We could no longer apply this same scaler to the test set since each scale was dependent on the hard drive. This has potential impacts on 'across-the-board' generalization of the model, a topic which will be discussed further in the paper.

Since applying the same scaler to the training and test sets was no longer an option, we took the same approach to scaling the test set and other validation sets as the training set. Each feature for each device was independently scaled to a mean of 0 and variance of 1.

This approach can be applied to any hard drive data of the same shape. This means that someone using this model could easily standardize their data and run it through our model without having to apply the exact transformation as us, something one would most likely have to do with typical normalization.

An alternative solution to this issue, and one that has been applied in the past [13], would have been to perform some kind of outlier detection and removal. However, this comes with the cost of losing potentially valuable data. Some points labled as outliers would be easy to justify as impossible. For example, consider a hard drive that reports its hours active as longer than the hard drive has been manufactured. Other points would not be as easy to justify removing, and there is the potential of removing something useful to the model.

It also raises the question of how this would be handled in an actual implementation of the model. Would an end user have to be okay with a model simply being wrong about their hard drive's RUL if it reported one of these extreme points? If a model is trained on a dataset without these extreme points, we cannot expect it to be accurate when it inevitably encounters feature values like this outside of training. For these reasons, we did not use this approach.

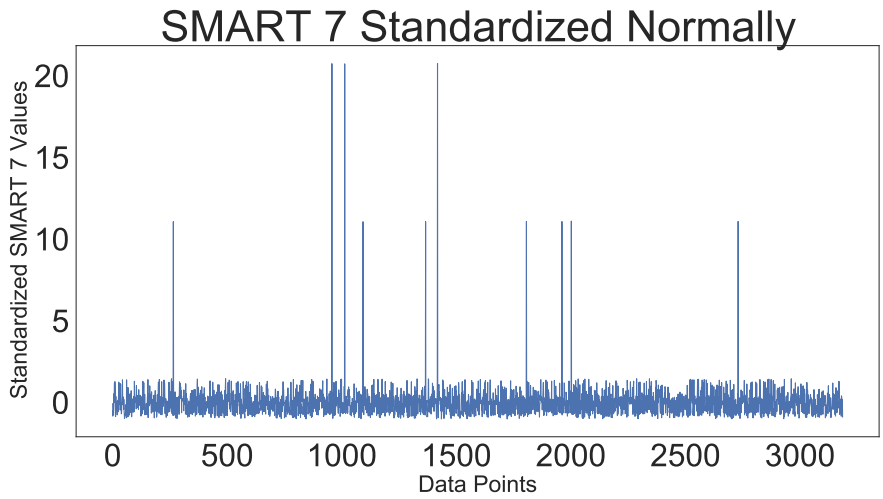



Figure 8: Values for S.M.A.R.T. 7 when standardized in a traditional way (Note the extreme points)

### 2.4.4 Reclassification of Data

At this point, we had a set of training data consisting of 78 hard drives with 5 S.M.A.R.T. features reported for each hard drive 60 days up to failure and the day of failure. We had 2 sets of test data. One consisted of 71 hard drives, different than the ones in the training set, with the same 5 S.M.A.R.T. features reported for each drive 60 days up to failure and the day of failure. The second test set consisted of 133 hard drives with the same 5 S.M.A.R.T. features reported 120 days up to failure and the day of failure.

After visualizing the data for each day leading up to failure, a pattern begins to emerge (see Figure 9). It appears that a few of the reported S.M.A.R.T. features have large jumps leading up to the day of failure. This is what would be expected. For example, consider S.M.A.R.T. 7, the rate of seek errors.

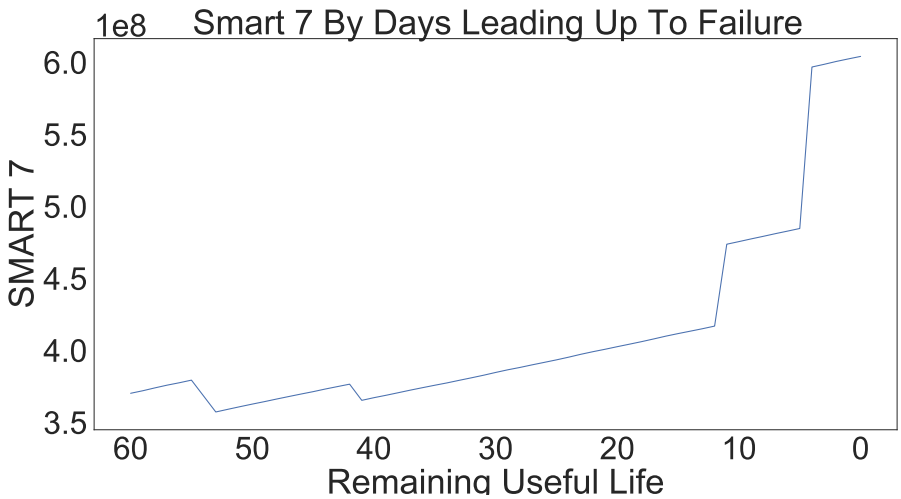



Figure 9: S.M.A.R.T. 7 values 60 days before failure (Notice the jump in values occurring about 15 days before failure)

One would expect these errors to increase dramatically as the hard drive begins to fail. Our model will likely be predicting a lower remaining useful life of the hard drive as each feature has a dramatic change. This will be a problem when trying to predict the RUL accurately using regression. How could the model say that a hard drive has 117 days of RUL instead of 116 when the feature values for these are virtually the same and these dramatic changes do not happen until right before failure?

To solve this problem, we reclassified every value for the remaining useful life that is over 30 days as 30 days. This created a category where the hard drive is in good working condition. This reclassification of the RUL is also done in [8]. Any day where a hard drive is reporting normal S.M.A.R.T. feature values can be treated as having 30+ days of remaining useful life. We believe

this is appropriate given the inspection of the feature values and would still maintain the practicality of the model. A consumer using this model would have about 30 days to back up and replace their hard drive before failure, much better than the last minute notification of failure usually given.

## 2.5 Model Training

### 2.5.1 LSTM Models

To train an LSTM model, the data has to be in the shape of [samples x timesteps x features]. The samples are the number of data points. (Given 60 days worth of data, including a 0th day of failure, for 78 hard drives, the number of samples is 4758.) The timesteps are the number of days of data leading up to each sample point. We trained these LSTM models with a variety of parameters for the timesteps. With these timesteps, our models do not have to make their prediction based on the current S.M.A.R.T. feature values. The models can also use the previous $n$ days' S.M.A.R.T. features to better understand the way the data is changing. The final dimension of the shape that LSTM data has to be in is the number of features. In this case, this is simply 5, as we chose 5 S.M.A.R.T. features in the previous section.

With the data in the correct shape, the LSTM models could be trained. To train the basic LSTM, we created a model with an LSTM layer consisting of 32 units followed by a densely-connected neural network layer. The Adam optimizer was chosen. The model was trained for 50 epochs with a batch size

of 64 and was shuffled during training. A very similar approach was taken to train the bidirectional LSTM. A Bidirectional wrapper was added to the LSTM layer that would return the last output.

All of the LSTM models were trained with 5, 10, 15, and 30 timesteps. This was done to see the impact on RUL prediction when the model is given varying amounts of data. An example loss function graph for the LSTM training is provided in Figure 10.

The LSTM models are trained on a specific hard-disk drive model with the five chosen features. In this, none of the LSTM models should have to be retrained unless major changes are made to the selected features, an unlikely event given that their meanings were determined by the model's manufacturer. So the performance on the test set should be comparable to the models' performance on new data if that data is in the correct shape.

### 2.5.2 Random Forest

To provide a baseline performance comparison, a Random Forest model was also trained. The random forest was trained on the same dataset as the LSTM models, split into 80% training and 20% testing. The data for the random forest was not standardized and each feature was given to it. Due to the flexibility of random forests, this would have been unnecessary and led to reduced performance when we tried it. Any data point more than 30 days away from failure was still capped at 30 days for comparability. It was

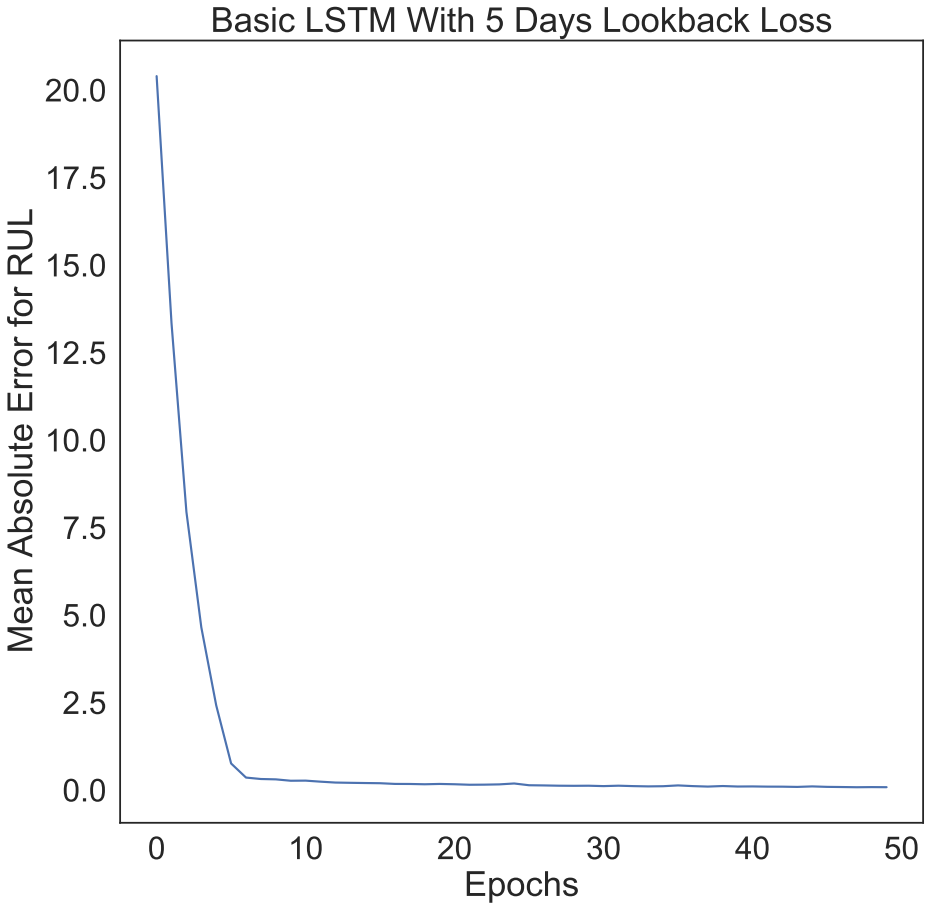



Figure 10: Loss function for a vanilla LSTM trained with 5 days look back over 50 epochs. Loss curves of other LSTM models follow a similar trend.

trained with 1000 estimators and no max depth.

## 2.6 Results

After each model was trained, each model was then evaluated on two test sets. The first included data similar to the data the models were trained on. This test set had the 5 S.M.A.R.T. feature values for the 60 days leading up to a hard drive failure for 71 hard drives. The second test set was included to see how to the trained models would perform when given a more difficult task. Instead of predicting the RUL for data at most 60 days away from hard drive failure, the second test set included data 120 days leading up to each hard drive's failure. This dataset included 133 hard drives. Both of these test sets were standardized in the same way the training sets were.

In order to evaluate the performance of these models, we needed some quantitative metrics. Because our models would output a continuous number representing the predicted remaining useful life, we chose $R^2$ as one of these metrics. We also collected the mean absolute error (MAE) in the predicted RUL. To concretely represent the usefulness of the models, we also collected accuracy. To determine how many correct predictions the model made, we rounded each prediction. So a prediction of 24.4 days RUL when the hard drive actually had 24 would be deemed a correct prediction.

The performance of each LSTM model with 5, 10, 15, and 30 days timesteps for 60 days of data is shown in Table 1. The basic LSTM network is denoted

as LSTM and the bidirectional LSTM is denoted as Bi-LSTM.

| Model | Timesteps | Accuracy | $R^2$ | MAE |
|---|---|---|---|---|
| LSTM | 5 | 0.910 | 0.992 | 0.287 |
| LSTM | 10 | 0.914 | 0.994 | 0.244 |
| LSTM | 15 | 0.916 | 0.992 | 0.247 |
| Bi-LSTM | 30 | 0.910 | 0.994 | 0.238 |
| Bi-LSTM | 5 | 0.939 | 0.999 | 0.131 |
| Bi-LSTM | 10 | 0.934 | 0.997 | 0.190 |
| **Bi-LSTM** | **15** | **0.964** | **0.998** | **0.120** |
| Bi-LSTM | 30 | 0.960 | 0.998 | 0.132 |
| RF | NA | 0.667 | 0.987 | 0.483 |

Table 1: Performance for trained models on test data 60 days before failure.

The performance of each LSTM model with the same timesteps for 120 days of data is shown in Table 2. The random forest model is also added for comparison at the bottom of both tables.

As can be seen, the LSTM models achieve high accuracy on data within the same range it is trained on. In Table 1 we see that the Bidirectional LSTM with a timestep of 15 days performs the best on the test set. This model achieves a MAE of 0.12, much lower than most of the state-of-the-art models, including Basak et al. [13] that achieved a MAE of 2.4 on the same model. Every LSTM model outperforms the Random Forest model. All of the LSTM models are able to generalize very well to the same time frame.

Figure 11, Figure 12, and Figure 13 demonstrate the difference in performance between the LSTM and random forest models. All of these graphs are sorted

| Model | Timesteps | Accuracy | $R^2$ | MAE |
|---|---|---|---|---|
| LSTM | 5 | 0.479 | 0.286 | 4.305 |
| LSTM | 10 | 0.471 | 0.219 | 4.589 |
| LSTM | 15 | 0.444 | 0.028 | 5.250 |
| LSTM | 30 | 0.358 | -0.359 | 6.433 |
| Bi-LSTM | 5 | **0.497** | 0.312 | 4.145 |
| Bi-LSTM | 10 | 0.496 | 0.216 | 4.448 |
| Bi-LSTM | 15 | 0.487 | 0.071 | 4.874 |
| Bi-LSTM | 30 | 0.369 | -0.565 | 6.792 |
| RF | NA | 0.378 | **0.333** | **3.647** |

Table 2: Performance for trained models when extrapolating to 120 days away from failure.

by the actual remaining useful life for demonstrative purposes.

Table **??** shows performance metrics for related data-driven approaches for hard drive health assessment. While the table is not exhaustive, it should provide some context to the performance of state-of-the-art models.

### 2.6.1 Observations

Some interesting observations can be gained from the graphs in Figure 11, Figure 12, and Figure 13. First, it is clear that the LSTM models outperform the Random Forest. While Figure 11 shows almost no separation between the True RUL and the Predicted RUL for the Bi-LSTM architecture, the difference is most clearly visible in Figure 13 for the Random Forest. It is also clear that the Bi-LSTM outperforms the regular LSTM model in Figure 12. Second, we see that the LSTM models incorrectly predict the RUL more often

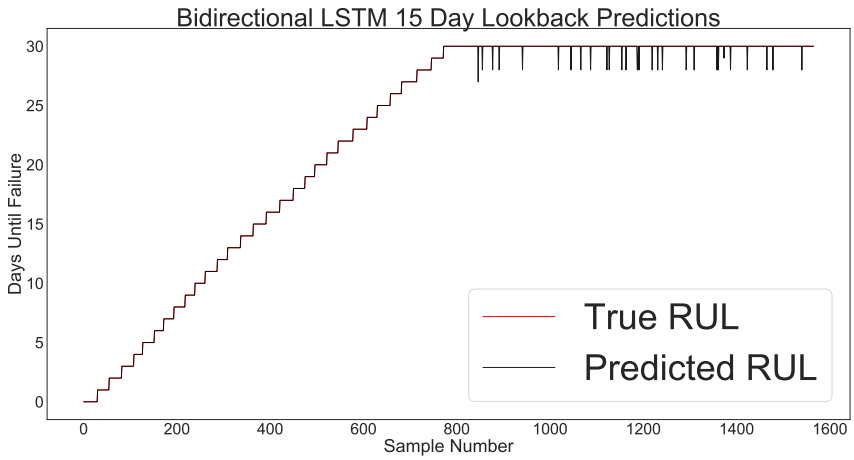



Figure 11: Bidirectional LSTM with 15 days lookback predictions. The points are sorted by the actual RUL.

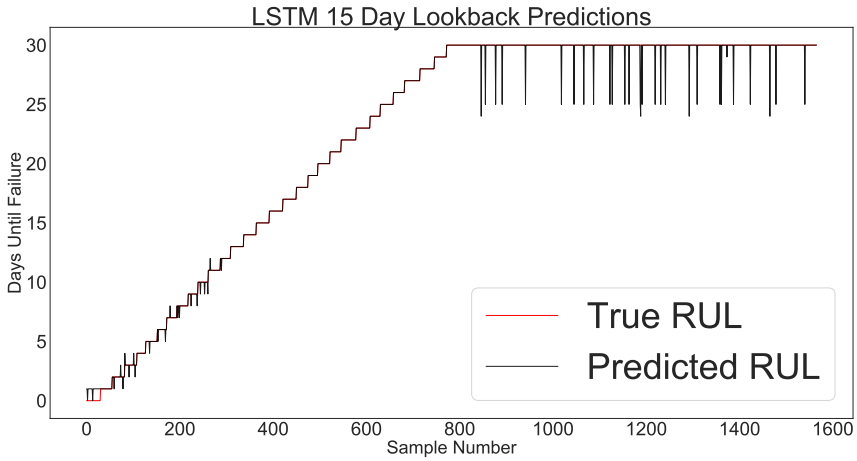



Figure 12: LSTM with 15 days lookback predictions. The points are sorted by the actual RUL.

| Model Type | Precision | Recall | MAE | Citation |
|---|---|---|---|---|
| Random Forest | 0.95 | 0.67 | - | Aussel et al.[9] |
| Attention LSTM | 0.93 | 0.96 | - | Wang et al.[73] |
| Random Forest | 0.66 | 0.94 | - | Lu et al.[47] |
| LSTM | 0.66 | 0.88 | - | Lu et al.[47] |
| CNN-LSTM | 0.93 | 0.94 | - | Lu et al.[47] |
| Random Forest | - | - | 6.4 | Anantharaman et al.[8] |
| LSTM | - | - | 8.15 | Anantharaman et al.[8] |
| Clustered LSTM | - | - | 2.4 | Basak et al.[13] |
| Bi-LSTM | - | - | 0.12 | **Our Approach** |

Table 3: Performance of previous approaches for RUL prediction by various data-driven approaches.

when the hard drive is further away from failure. Most of the deviations from the actual remaining useful life are when the hard drive is in the category of 30+ days RUL. The LSTMs are more accurate when the hard drive is closer to failure, reinforcing its practical importance. Third, we can see that the Random Forest appears to have an opposite relationship with the data. The Random Forest is much less accurate the closer the hard drive gets to the day of failure. There is not a spot on Figure 13 where the Random Forest appears to consistently predict the RUL, but it is certainly worse when the actual RUL gets below 10 days. It is also worth noting that the Random Forest is appearing to overestimate the remaining useful life. A real user would probably want the model to underestimate the RUL. It is better that they replace their hard drive early than have it fail on them before they can replace it.

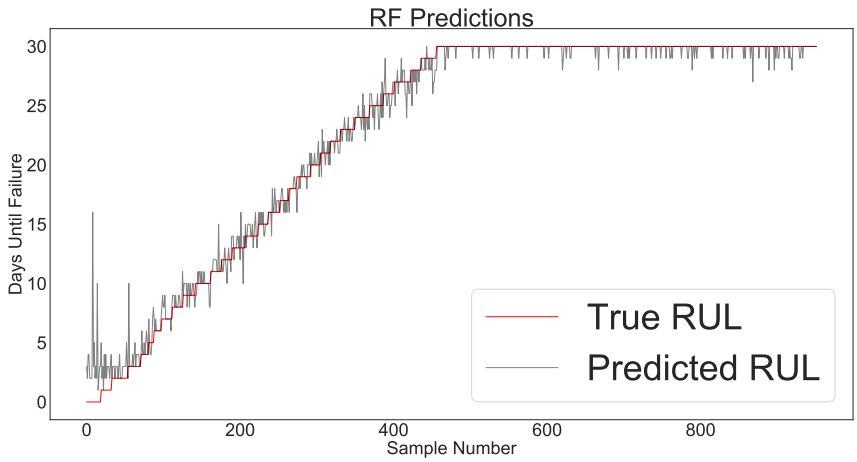



Figure 13: Random Forest predictions. The points are sorted by the actual RUL.

Every model performs poorly when attempting to extrapolate far into the future. This is a common problem with statistical and machine learning prediction methods in general. From Table 2, we see that the Bidirectional LSTM has the highest accuracy at predicting the remaining useful life of hard drives that are twice as far away from failing (120 days) as the models were trained on (60 days). This LSTM model gets nearly half of its predictions correct. In contrast, the random forest has a lower accuracy score, but also has a higher $R^2$ and lower MAE. This implies that the random forest predictions may actually be better than the LSTM models, but the random forest does not predict the exact RUL as well.

It would be hard to expect a model to be able to perform as well on data that is twice as far away as the data it was trained on, but being able to do

so would highly improve the real-world usefulness of the model.

## 2.7 Standardization and Generalization

This section is dedicated to more discussion of our novel standardization approach. The benefits and downsides of this method of standardization will be explained.

As mentioned in Section 2.4, the hard drive data needed to be normalized or standardized to be used by the LSTM models. A traditional standardization approach where the entire features would be standardized as a whole was considered, but not adopted due to the poor LSTM performance. We believe the cause of this to be partially due to extreme data points influencing the weights of the LSTM and difficulty of the problem.

To standardize the data, we standardized the data per sample. This means that each feature of each hard drive was essentially treated as its own dataset when applying the scaler.

To better explain this, consider two hard drives: hard drive A and hard drive B. Both hard drive A and B are of the same model, but hard drive A has an extreme value for its S.M.A.R.T. 7 feature on 2 of the 60 days of data captured. When all of the S.M.A.R.T. 7 feature values are standardized together, the weight of these extreme values are maintained. The standardization we used made each feature have a mean of 0 and variance of 1. It follows that the more data points that exist, the less the extreme values of S.M.A.R.T. 7 that

hard drive A contains will be pulled towards a reasonable number. For some models, such as a Random Forest, this may not be an issue, but because of the nature of recurrent neural networks we believe these extreme values will amplify the weights and reduce the performance of the model.

By normalizing the S.M.A.R.T. 7 values for hard drive A and B separately, the extreme values for hard drive A will be pulled more towards reasonable values while still maintaining the patterns in their feature values that the LSTM can learn. Figure 14 is included to visualize this process. The data included in this figure is entirely theoretical, but demonstrates the differences in the range of the data for both of the standardization techniques discussed.

This method of standardization deals with the extreme data points without risking a loss of data as would be done using an outlier removal. However, this does have the potential downside of not being able to reuse the same standardization function to new data. At first, we assumed that this would cause the model to lose any generalization, completely overfitting on the training data. From the results, we see that this was not the case. The LSTM models performed exceptionally well at predicting the RUL of hard drives in the test set after undergoing this standardization. It outperforms a model, the Random Forest, that was not standardized this way.

As was also demonstrated, this method of standardization has the potential of restricting the range of time that the model is useful. We saw that the models did not perform well on 120 days worth of data. Using this standardization

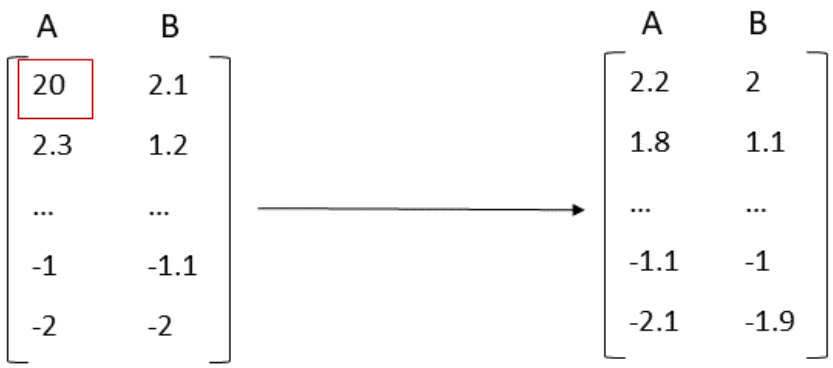



Figure 14: Visualizing a potential difference between the regular method of standardization (left) and the per sample standardization (right.)

technique, the data around day 120 should have around the same values as the data around day 60 for the other test sets. Instead of being able to apply a standardization function that would potentially assign different values to those past 60 days, they are all standardized to a similar range.

We believe that this should not discredit this technique of standardization. The performance of the LSTM models were greatly increased by this standardization technique over a traditional one. We attribute this to the complexity of the dataset. Many of the S.M.A.R.T. feature values remain relatively-unchanging until the hard drive is soon to fail. The typical method of standardization may assign a similar value for one of the S.M.A.R.T. features 60 days away from failure to one 120 days away from failure due to the nature of the data (see Figure 9). This, while still maintaining outliers, would probably make training an accurate LSTM very difficult.

## 2.8 Conclusion

In this section, we attempted to solve the problem of remaining useful life prediction for hard drives. Given a series of hard drive data 60 days before the hard drive fails, we were able to predict the remaining useful life of a hard drive with state-of-the-art accuracy, outperforming previous approaches in the literature [13] that worked on the same data and the same model, in particular.

To accomplish this, we used both Bidirectional and vanilla Long Short-Term Memory networks, or LSTMs, with highly customized pre-processing directives. We provided a cap on the number of days a hard drive could be away from failure, giving our model a category where the hard drive is treated as operating normally. This allowed the model to better handle data where the hard drive is not failing, improving the performance. We implemented a unique method of data standardization that standardized each hard drive independently. This maintained generalization on similar time windows, but came with the cost of reducing performance on time windows much different than trained on.

We also implemented a Random Forest model as a baseline to compare our LSTM models to. The random forest model was able to perform as well as the LSTM models on data in a different time window than trained on, but was severely outperformed on the test data from the same time window.

### 2.8.1 Future Work

While our work is able to predict the remaining useful life of hard drives well, there is still much room for future work to be done on this complex problem. Most notably, an ideal model would be able to achieve similar performance to our model while still being able to extrapolate well. There is no set number of days that one could select for training the model that would not run into extrapolation issues. Even if one were to train a model on 1,000 days of data, there is still a scenario where someone would have a hard drive that is more than 1,000 days away from failing. We attempted to solve this by implementing the cap on the remaining useful life, but the models still struggled to perform well when the time interval was doubled. Improving the extrapolation performance would be an important milestone towards true generalization and add to the applicability of the model in a real-world deployment scenario.

Second, future work could explore a model that is able to predict the remaining useful life for any hard drive model. While we believe our process could easily be reproduced for any hard drive model, it would be worth exploring the possibility of training a model that would work for more than one model.

Finally, future work could implement a hard drive RUL prediction model into a real system, testing the performance of the model on real-time hard drive data for hard drives that may not fail for years.

# 3 Aircraft Engine RUL Prediction

The next application of long short-term memory networks to predict remaining useful life that I performed was in aircraft engines. This section contains in-progress work.

## 3.1 Introduction

There are over 16,400,000 flights handled by the Federal Aviation Administration (FAA) in the United States every year [4]. With this many planes, hardware failures are bound to happen. Unexpected problems with an aircraft can cause flight delays and cancellations. A 2010 report sponsored by the FAA through its National Center of Excellence for Aviation Operations Research estimated "that the total cost of all US air transportation delays in 2007 was $32.9 billion." They say that "delays could be caused by mechanical problems." [11] In addition to the monetary cost of airplane hardware failures, an issue with an aircraft could lead to an in-flight disaster, potentially endangering the pilots or passengers on board.

To mitigate the effects of these failures, it is important to be able to predict the remaining useful life of aircraft components, such as the engine. This allows airlines to make appropriate maintenance to the engines before the inevitable engine failure happens. To help facilitate this process, NASA released the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset in 2010 [19].

The C-MAPSS dataset contains simulated turbofan engine data. It simulates a variety of realistic flight conditions and injects faults into the engine during one of the flights. It contains full recordings of the simulated flights with 30 parameters. Due to the realism of the simulated data, this dataset is often used for benchmarking of various prognostics approaches [55] and for remaining useful life prediction [72].

However, there was room for improvement with the C-MAPSS dataset. The engine fault could not be predicted before it would happen; all remaining useful life prediction would have to begin after the fault was detected. In order to provide a more realistic failure evolution, provide more data, and make models more usable, NASA released a New Commercial Modular Aero-Propulsion System Simulation (N-CMAPSS) dataset [5]. The N-CMAPSS dataset "contains eight sets of data from 128 units and seven different failure modes affecting the flow (F) and/or efficiency (E) of all the rotating sub-components" [5]. It contains 47 different variables taken from measurements, virtual sensors, model health parameters, and auxiliary data. It also includes a remaining useful life measurement, allowing for easy RUL prediction. This paper will use the N-CMAPSS data to predict the remaining useful life of turbofan engines.

### 3.1.1 Contributions

In this paper, we present a series of long short-term memory models to predict the remaining useful life of aircraft engines in the N-CMAPSS dataset.

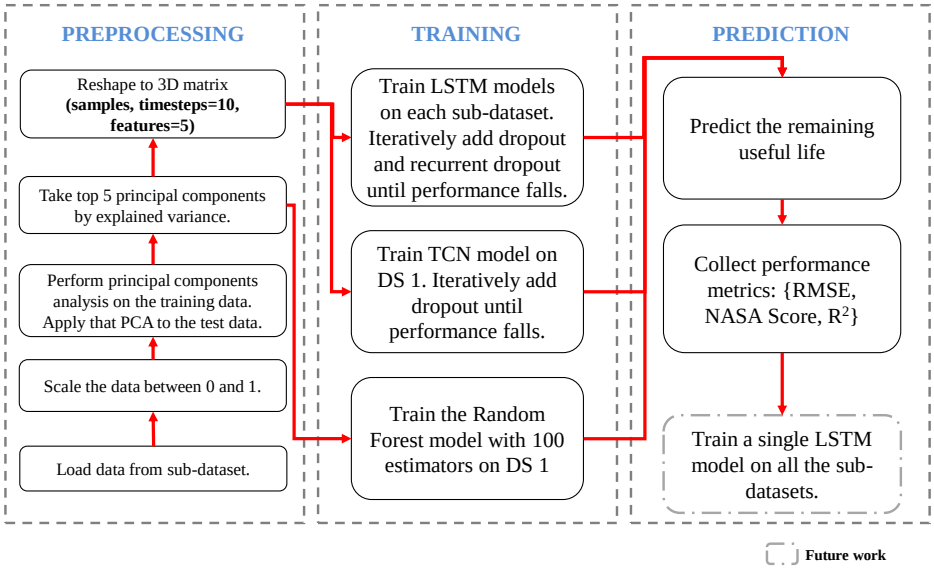



Figure 15: Overview of the model training and evaluation process of this paper.

These models show a high degree of success in this problem, with one model achieving a RMSE of 6.3 and a NASA score of 0.637. This is among the best in the literature. We also train a Random Forest and Temporal Convolutional Network model on one of the N-CMAPSS datasets. Our simple LSTM approach substantially outperforms these models, even when performing the same regularization method as the LSTM on the other models. Additionally, the approach we present is simple and could be easily replicated by anyone needing to predict the RUL of aircraft engines. Our approach could also be transferred to many other prognostics datasets. This is contrary to many of the previous approaches in the literature that use highly-specific and complex methods of predicting engine RUL.

We demonstrate the success of principal components analysis to reduce the complexity of this large dataset without sacrificing model performance. We highlight the issue of overfitting in the sub-datasets of N-CMAPSS and provide our solutions. Using recurrent dropout and a dropout layer, we are able to improve the test performance of an LSTM model trained on the second sub-dataset of N-CMAPSS from a RMSE of 10.38, a statistic which could already be considered good, to a RMSE of 8.627. We analyze the predictions of our models to show gaps in model performance that can be addressed in future works to lead to even better RUL prediction models.

## 3.2 Previous Approaches

### 3.2.1 C-MAPSS

Despite its limitations, understanding the approaches taken to predict RUL using the C-MAPSS dataset is essential to establishing a basis for approaching the N-CMAPSS dataset. While the fundamental differences between the two datasets make directly comparing the performance of solutions to the N-CMAPSS dataset to the best performing C-MAPSS models impossible, we can transfer techniques from the best models to this new dataset. According to a 2021 review by Vollert et. al [72], the top three most common machine learning models used to predict the RUL of the C-MAPSS dataset were the Long Short-Term Memory Network (LSTM) (28 papers), Convolutional Neural Network (CNN) (10 papers), and a CNN combined with an LSTM (10

papers). CNNs and LSTMs were the best performing models in terms of pure predictive performance, using root mean square error (RMSE) as a metric.

### 3.2.2 N-CMAPSS

As the N-CMAPSS dataset is a relatively-new dataset at the time of writing this paper, not much work has been done exploring its use for remaining useful life prediction. Perhaps the most notable usage of the N-CMAPSS dataset was in the 2021 Prognostics and Heath Maintenance Conference Data Challenge [3]. This was a challenge held by the 13th Annual Conference of the Prognostics and Health Management Society in which the contestants were tasked with predicting time to failure of aircraft engines using a subset of the N-CMAPSS dataset. This challenge is useful to look at, as the top three teams were asked to write papers on their approach. A similar competition was held in 2008 [36] that introduced the original C-MAPSS dataset and lead to the development of some powerful prognostics methods [55].

In the 2021 PHM competition [3], a subset of N-CMAPSS with 100 aircraft units across 7 different failure models is used. The scenario descriptors (flight data), measurements, and auxiliary data were provided. This can be contrasted with the 128 units and data from virtual sensors and engine health parameters that are present in the full N-CMAPSS dataset. The competition was scored using a combination of 50% of each of the two metrics, RMSE (13) and NASA's scoring function (14) [5] where late predictions are penalized heavier than early predictions. The formal definitions of these metrics are

shown in the equations below.

$$RMSE = \sqrt{\frac{1}{m_{v*}} \sum_{n=1}^{m_{v*}} \left(\Delta^{(n)}\right)^2} \tag{13}$$

$$s_c = \frac{1}{m_{v*}} \sum_{n=1}^{m_{v*}} exp(\alpha|\Delta^{(n)}|) - 1 \tag{14}$$

In these equations, $m_{v*}$ is the size of the validation dataset, $\Delta^{(n)}$ is the difference between the estimated and real remaining useful life for the nth sample, and $\alpha$ is $\frac{1}{13}$ when underestimating the RUL and $\frac{1}{10}$ if not.

Third place in the 2021 PHM contest [66] used two stacked deep convolutional neural networks. They carry out an encoding phase followed by an RUL prediction phase. They scored 3.651 in the competition. In their paper, they report a RMSE of 6.24, a $s_c$ of 0.64, and a combined score of 2.95 on their validation set, performing better than their competition score. Second place [26] used an inception-based deep convolutional network. They only trained their model on one flight but were able to achieve a competition score of 3.33. Surprisingly, the results on their validation set were slightly worse, achieving a RMSE of 12.5, $s_c$ of 2.53, and an overall score of 7.5. First place [46] used another deep convolutional network, utilizing a CNN that could take variable-length input sequences. They achieved a contest score of 3.006, not reporting their validation metrics. From these contest results, we can see that CNNs are a powerful tool for the task of RUL prediction. This is to be

expected, as 20 of the papers in the survey on the C-MAPSS dataset [72] mentioned in the above section used some form of CNN. However, the most popular model used on the C-MAPSS dataset was an LSTM model.

There are a couple of papers in the literature that use the N-CMAPSS dataset outside of this competition. One such example uses a Deep Gaussian Process to estimate RUL on the full N-CMAPSS dataset [15]. They introduce a new type of model, a DGP, that they validate on the N-CMAPSS dataset. Their proposed model achieves a RMSE of 7.37 with their best model, a Monte-Carlo Dropout model, achieving a RMSE of 7.31. These were evaluated on three test units. A second example uses a fusion of a physics-based and deep learning model for RUL prediction on the N-CMAPSS dataset [20]. They used hybrid vanilla neural networks, CNN, and LSTM models. On three test units, they achieved a RMSE of 4.22 and an $s_c$ of 0.43. Both of these approaches use very custom models that are powerful remaining useful life predictors. An aggregation of performance for all of the models in this subsection is shown in Table 4.

| Model Type | RMSE | $s_c$ | Contest | Reference |
|---|---|---|---|---|
| Stacked CNN | 6.24 | 0.64 | 3.651 | Solis-Martin et al. [66] |
| Inception CNN | 12.5 | 2.53 | 3.33 | Devol et al. [26] |
| Variable CNN | - | - | 3.006 | Lovberg et al. [46] |
| Deep Gaussian Process | 7.37 | - | - | Biggio et al. [15] |
| Monte-Carlo Dropout | 7.31 | - | - | Biggio et al. [15] |
| Hybrid | 4.22 | 0.43 | - | Chao et al. [20] |

Table 4: Performance on N-CMAPSS dataset

## 3.3 Machine Learning Model Details

Along with an LSTM model, we also consider other types of machine learning models.

### 3.3.1 Random Forest

To establish a performance baseline, we first consider a classical machine learning model. On the C-MAPSS dataset, many tree-based models were trained [72]. With this in mind, we choose to use a Random Forest baseline model.

Random forests use multiple Decision Trees to vote on a prediction. These Decision Trees are built with randomness so that each decision tree can be uncorrelated with another. With many uncorrelated trees, the Random Forest has better voting accuracy than with a single Decision Tree. [17]

### 3.3.2 Temporal Convolutional Neural Network

Recent results indicate that Convolutional Neural Networks can outperform Recurrent Neural Networks in sequence modeling [10]. Therefore, we also consider the use of Convolutional Neural Networks (CNNs) for predicting engine remaining useful life on the N-CMAPSS dataset.

A Convolutional Neural Network that is in an effective architecture for sequential prediction can be called a Temporal Convolutional Network (TCN). Temporal Convolutional Networks are Convolutional Neural Networks that

have causal convolutions. In other words, outputs at a specific time point can only be convolved with elements from the same or previous time points. This keeps information from the future from being used by the past. TCNs also predict an output that is the same size as the input, similar RNNs. As TCNs are not the main model type of this paper, an in-depth description of how CNNs work and how TCNs differ is omitted. However, we encourage the reader to see works such as papers by Lea et. al [40] and Bai et. al [10], as they focus on the abilities of TCNs for sequential prediction tasks.

## 3.4 Data Preprocessing

The N-CMAPSS dataset is split into eight different sub-datasets. In these, there are seven modes of failure that impact the flow or efficiency of each sub-component of the engine. The five sub-components consist of the "fan, low pressure compressor (LPC), high pressure compressor (HPC), low pressure turbine (LPT) and high pressure turbine (HPT)" [5]. A table displaying which sub-components are being degraded in which sub-datasets is shown in Table 2. Additionally, recommended training and testing sets are already included in each sub-dataset. These describe a number of simulated independent aircraft, usually about 7 aircraft in the training set and 4 in the testing, depending on the sub-dataset. Considering total data points alone, most sub up at about a 66/33 train/test split. We choose to use these recommended splits.

Each of these sub-datasets contains over two gigabytes of data. With 47

different variables and such a large file size, each of these datasets are very complex. This complexity is increased when considering preprocessing data for an LSTM. An LSTM needs data to be in the shape of [Samples, Timesteps, Features]. Samples and features correspond to a traditional wide data format's rows and columns. The timestep dimension introduces history to the data. For example, if we chose to use a timestep of 10, each data point would have the samples for each feature for the current and previous 10 data points. In the context of this problem, introducing the timesteps gives our model context for each data point. This allows the model to better view the progression of failure. While this is a powerful concept, it adds to the complexity of the dataset, nearly increasing the dataset to $n$ times its size for a timestep of size $n$. To bring the complexity of the dataset back down, we can lower the number of features.

| Dataset No. | Fan | | LPC | | HPC | | HPT | | LPT | |
|---|---|---|---|---|---|---|---|---|---|---|
| | E | F | E | F | E | F | E | F | E | F |
| 1 | | | | | | | • | | | |
| 2 | | | | | | | • | | • | • |
| 3 | | | | | | | • | | • | • |
| 4 | • | • | | | | | | | | |
| 5 | | | | | • | • | | | | |
| 6 | | | • | • | • | • | | | | |
| 7 | | | | | | | | | • | • |
| 8 | • | • | • | • | • | • | • | • | • | • |

Table 5: Degradation affecting either efficiency (E) or flow (F) of subcomponents in N-CMAPSS dataset.

### 3.4.1 Feature Reduction

There are around 47 features in the N-CMAPSS dataset. One of these features is the RUL itself, so we remove that from the dataset to use as a predicted label. Considering the remaining features, it follows that some of these features are less useful than others when predicting remaining useful life. It would be ideal to only have 4 or 5 key features that can explain engine failure well. To accomplish this, we can either perform feature selection or dimensionality reduction.

Feature selection can be thought of as the task of removing the least important features, determined by some metric so that we are left with only the most important features. This can be done in a variety of ways. Commonly, features are selected by choosing the features that correlate most with the predicted feature [16]. However, these techniques often require independence or linearity of data that our dataset does not satisfy. Recently, Random Forests have been used as a common form of feature selection [59]. However, this requires the training of a Random Forest model, a non-trivial task for large datasets. Additionally, in feature selection techniques, the solution often involves removing the features that are not needed. When removing features, we may take away a potential interaction between features that our machine learning model could pick up on. For these reasons, we choose to perform dimensionality reduction.

### 3.4.2 Principal Components Analysis

One of the most popular methods of dimensionality reduction is Principal Components Analysis (PCA). PCA simplifies the data to a number of orthogonal principal components. These uncorrelated principal components maximize the variance [37]. The principal components can be thought of as new variables that preserve the variability of the old data in a concise way. One can choose the first $k$ principal components that are needed to explain $m\%$ of the variability [63]. In many situations, the first 2 to 4 components can explain over 80% of the variance. This is often enough to train a well-performing machine learning or statistical model.

Since each of these principal components is a combination of many original features, a downside of using PCA comes from a loss in the interpretability of features. In many domains, it would be much less meaningful to say "the first principal component is the most important feature" than to leave the data in terms of its original features. Some analysis of feature importance can still be done, but it is usually much more difficult. In our case, the variance-preserving dimensionality reduction of this large dataset is worth the slight loss of interpretability, so we choose to apply PCA to the data.

Before applying PCA to the N-CMAPSS data, we normalized the data using a MinMaxScaler. This scales each data point to a value between zero and one. All of the aircraft engine features are on different scales, so normalizing the data will ensure that the model will not give higher weight to a feature

that is simply on a different scale.

After some experimentation, we found that 4 or 5 principal components were enough to explain over 90% of the variance in most of the N-CMAPSS sub-datasets. For consistency, we choose to keep the 5 principal components that explain the highest variance for each sub-dataset. We fit the PCA on the training values and then use this to transform the training and testing sets. An example graph displaying the percent of variance explained by each principal component, taken from sub-dataset 1 (DS1) can be seen in Figure 16. In this case, the first 5 components explain 94.89% of the variance. All of the other sub-datasets show similar patterns.

### 3.4.3 Data Reshaping

As previously mentioned, an LSTM requires data to be in the shape [Samples, Timesteps, Features]. So, as a final preprocessing step, we reshape the data to match this. We choose a timestep of 10. This is largely due to good experimental performance and computing limitations. Even with the data reduced to 5 features, the datasets are still extremely large, leading to long training time and high memory requirements. The preprocessed training split of the first dataset was in the shape of (4906627, 10, 5) and a test split with the shape (2735223, 10, 5). The other dataset files demonstrate similar numbers, most of them with slightly larger amounts of samples.

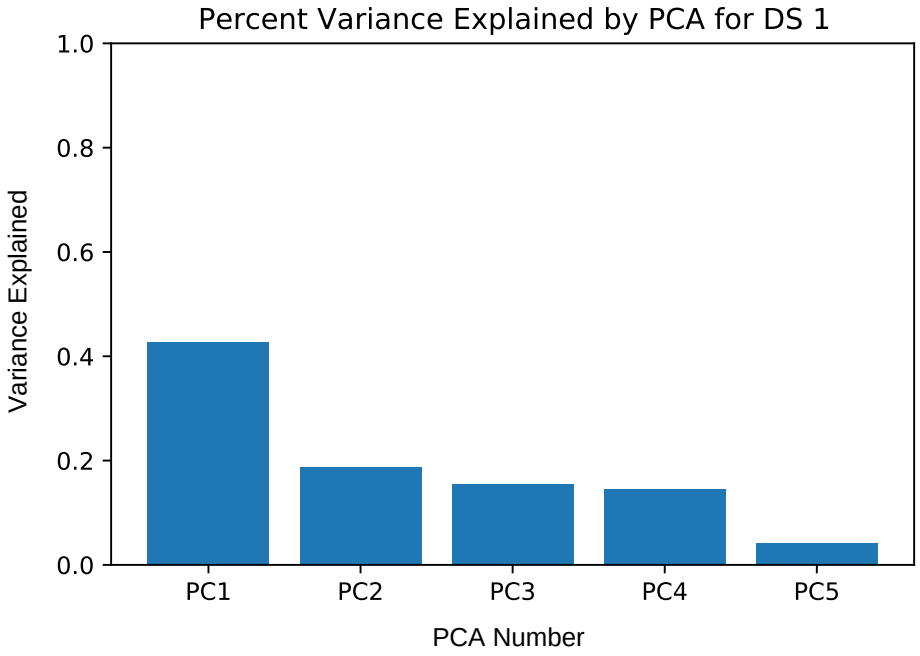



Figure 16: Percent of variance explained by the first five principal components for sub-dataset one of N-CMAPSS.

## 3.5 Model Training

### 3.5.1 The Problem of Overfitting

With the N-CMAPSS data properly preprocessed, we could begin the model training process. We began with a bidirectional LSTM architecture that worked well in our previous work [25]. However, it soon became apparent that this model was too powerful for these datasets. Even after simplifying the model to a vanilla LSTM, one epoch of training will learn the entire training set for most of the sub-datasets, causing a loss of generalization. Figure 17 shows the model's performance on the training split of the second sub-dataset

when trained for one epoch with a vanilla LSTM. It also shows the model's performance on the test set. The model achieves a RMSE of 0.43 on the training, but only a RMSE of 10.38 for the test. It is clear that the LSTM model performs well on the temporal N-CMAPSS dataset, but we need the model to generalize better.

To reduce a model's performance on its training set, regularization techniques are often used. A common regularization technique for neural networks is dropout [67]. A dropout layer is essentially a normal neural network that drops nodes and their connections with a random probability. This takes away the reliability of the presence of any given hidden unit in a neural network, reducing the ability for an architecture to overfit.

Along with dropout layers, there are a variety of variations on the dropout layer that can reduce overfitting in recurrent neural networks [53]. One such variation that we found useful for this problem was recurrent dropout [60]. Recurrent dropout is applied to the cell update vector, the arguments to the $g$ function in Equation 9. This is different than forward dropout, as it targets the recurrent connections of an LSTM. This, combined with dropout layers, allows us to target overfitting in multiple parts of an LSTM network, improving the generalization of our model.

Using these regularization techniques, we are able to improve the performance on the test set of sub-dataset two from a RMSE of 10.38 to a RMSE of 8.627. This is a drastic improvement that would have a large impact on the accuracy

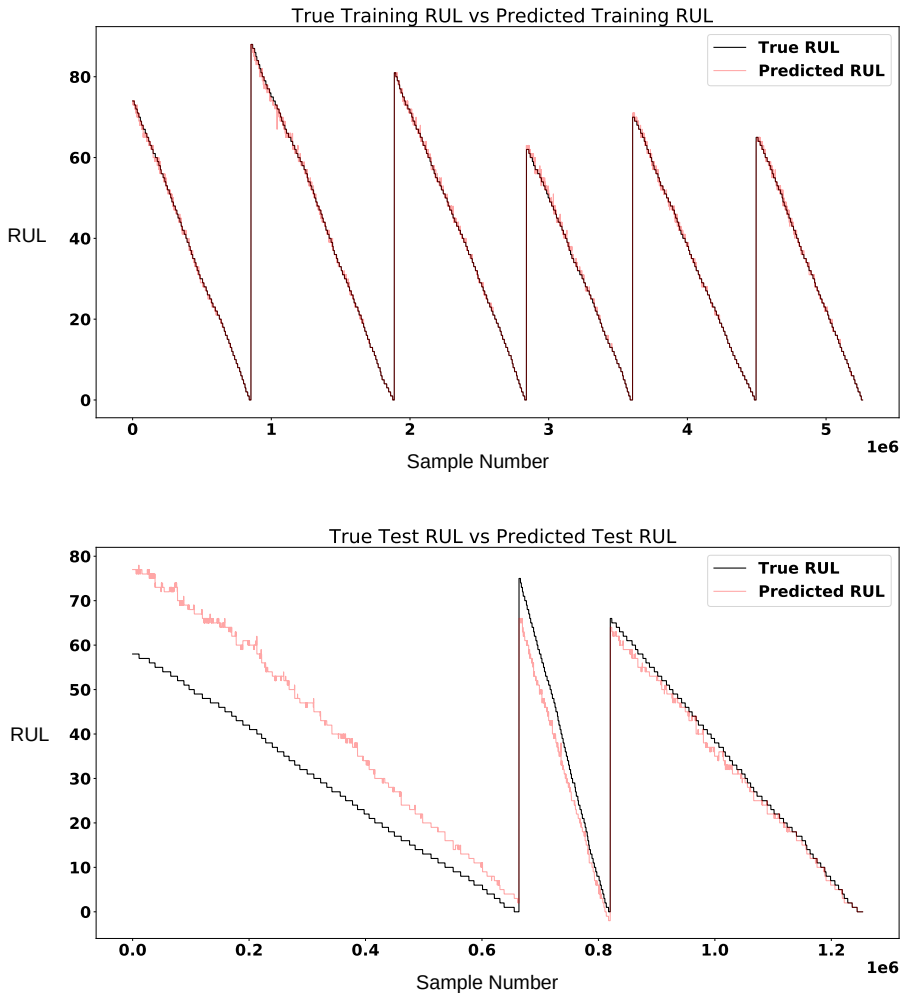



Figure 17: Evidence of overfitting on sub-dataset two. The top depicts the training performance, the bottom the test.

of this model if used in a real-world scenario.

### 3.5.2 Model Architecture

With methods of regularization, we began to train the LSTM models on each of the sub-datasets. Our hyperparameter selection method was as follows.

1. Train vanilla LSTM for one epoch
2. Add dropout layer with 10% dropout
3. Iteratively add 10% dropout until model performance drops
4. Train a model with 5% less dropout, continue with the best of this and the previous model
5. Perform the previous two steps for recurrent dropout if needed
6. Repeat for the next sub-dataset

On most of the sub-datasets, this simple training methodology led to great performance, as will be discussed in the next section. The final hyperparameters that were selected are shown in Table 6. It is possible that different hyperparameters could lead to slightly better performance, especially due to the stochastic nature of LSTM model training, but these are the ones that worked best for us. Pseudocode for our entire LSTM model training process can be seen in Algorithm 1.

**Algorithm 1** Preprocessing and hyperparameter selection for LSTM models

```
X_train, X_test ← splits from current sub-dataset
y_train, y_test ← X_train[RUL], X_test[RUL]
X_train.drop(RUL), X_test.drop(RUL)
MinMaxScaler.fit(X_train)
X_train ← MinMaxScaler.predict(X_train)
X_test ← MinMaxScaler.predict(X_test)
PCA.fit(X_train)
X_train, X_test ← PCA.predict(X_train)[0:4], PCA.predict(X_test)[0:4]
X_train ← X_train.reshape(X_train.n_rows, 10, 5)
X_test ← X_test.reshape(X_test.n_rows, 10, 5)
d_out ← 0.0
prev_model ← LSTM(nodes=16, epochs=1, Dropout=d_out, X_train, y_train)
curr_rmse, prev_rmse ← 9999
while curr_rmse ≤ prev_rmse do
    d_out ← d_out + 0.1
    prev_rmse ← RMSE(prev_model.predict(X_test), y_test)
    curr_model ← LSTM(nodes=16, epochs=1, Dropout=d_out, X_train, y_train)
    curr_rmse ← RMSE(curr_model.predict(X_test), y_test)
    if curr_rmse < prev_rmse then
        prev_model ← curr_model
    end if
end while
curr_model ← LSTM(nodes=16, epochs=1, Dropout=d_out-0.05, X_train, y_train)
curr_rmse ← RMSE(curr_model.predict(X_test), y_test)
if min(curr_rmse, prev_rmse) > 13 then
    repeat previous while with recurrent dropout
end if
```

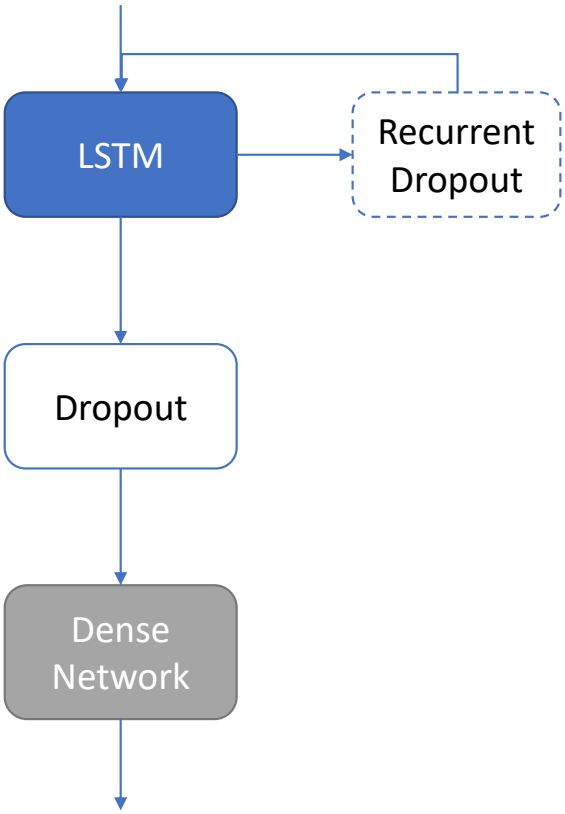



Figure 18: Our LSTM model architecture.

### 3.5.3 Training of Comparison Models

We also trained Random Forest (RF) and Temporal Convolutional Network models (TCN) to compare against our LSTM models. The RF model was trained to serve as a baseline classical machine learning model. The TCN models were trained to compare the LSTM's performance on this task with another state-of-the-art model.

| Dataset No. | Epochs | Hidden Units | Dropout | Recurrent Dropout |
|---|---|---|---|---|
| 1 | 1 | 16 | 0 | 0 |
| 2 | 1 | 16 | 0.25 | 0 |
| 3 | 1 | 16 | 0.35 | 0 |
| 4 | 1 | 16 | 0.3 | 0.35 |
| 5 | 1 | 16 | 0.1 | 0 |
| 6 | 1 | 16 | 0.35 | 0.55 |
| 7 | 1 | 16 | 0.15 | 0 |
| 8a | 1 | 16 | 0.25 | 0 |
| 8c | 1 | 16 | 0.25 | 0 |

Table 6: Selected hyperparameter values for N-CMAPSS sub-datasets.

We trained the Random Forest model on sub-dataset 1. We used 100 estimators with 6 cores working in parallel. We also trained the Temporal Convolutional Network on sub-dataset 1. We used 4 convolutional filters and followed the regularization hyperparameter selection algorithm described above for the LSTM to select the amount of dropout used. This led to a dropout layer with dropout of 0.3. The TCN was trained for 1 epoch.

We only train these models on the first sub-dataset to save computational time and resources. Training on one sub-dataset is enough to allow for comparison between models for at least one failure mode. We trained TCNs on other sub-datasets, but the performance followed the same pattern as on the first sub-dataset.

## 3.6 Results

After the models were trained, we used the models to predict RUL for the respective test sets. We tracked the metrics of RMSE and NASA's scoring function ($s_c$ in Equation 2). The results of each of the LSTM models are shown in Table 7. As can be seen, the best performance was on DS 1 with a RMSE of 6.304 and a NASA score of 0.637. This outperforms every model in the literature, except for one [20], using a simple LSTM network.

A visual comparison of the performance of each of the sub-datasets can be seen in Figure 19. As displayed, the LSTM model performs the worst on DS4 and DS6. Figure 20 displays the predicted values on top of the true RUL values for eight of the sub-datasets (dataset 8a is omitted as it is similar to 8c).

The performance of the Random Forest, Temporal Convolutional Network, and LSTM model trained on DS 1 can be seen in Table 8. As can be seen, the LSTM outperforms each of the other model types. The Random Forest performs the worst, obtaining a RMSE of 13.065 and NASA score of 1.77. The TCN performs slightly worse than the LSTM (but still notably better than the RF) with a RMSE of 8.83 and a NASA score of 0.8.

### 3.6.1 Observations and Analysis

Some interesting observations can be gained from analyzing the models' predictions.

| Dataset No. | $R^2$ | RMSE | NASA Score |
|---|---|---|---|
| **1** | **0.934** | **6.304** | **0.637** |
| 2 | 0.793 | 8.627 | 0.934 |
| 3 | 0.789 | 9.778 | 0.890 |
| 4 | 0.494 | 17.997 | 2.030 |
| 5 | 0.862 | 8.692 | 1.205 |
| 6 | 0.4864 | 16.405 | 2.554 |
| 7 | 0.831 | 10.735 | 0.847 |
| 8a | 0.534 | 12.338 | 1.130 |
| 8c | 0.595 | 10.934 | 1.129 |

Table 7: Performance of each LSTM model on its respective test set.

| Model | $R^2$ | RMSE | NASA Score | Improvement |
|---|---|---|---|---|
| RF | 0.715 | 13.065 | 1.770 | 51.7% |
| TCN | 0.870 | 8.830 | 0.8 | 28.6% |
| **LSTM** | **0.934** | **6.304** | **0.637** | |

Table 8: Performance of each model type on sub-dataset 1. The improvement is 1 - $\text{RMSE}_{\text{LSTM}}/\text{RMSE}_{\text{MODEL}}$, quantifying how much better the LSTM is.

First, we will consider the LSTM models. The LSTM models perform notably worse on DS4 and DS6. The models achieve a RMSE of 17.997 and 16.405 with a NASA score of 2.03 and 2.554 for DS4 and DS6, respectively. The model trained on DS4 has a worse RMSE but a better NASA score than the model trained on DS6. Both of these metrics seem substantially higher than the same metrics for the other sub-datasets. Usually, this would mean that the models are struggling to learn some pattern in the data. However, we observed that these two were the most overfit models. Without much effort, the LSTM would learn the training set. As a consequence, these models

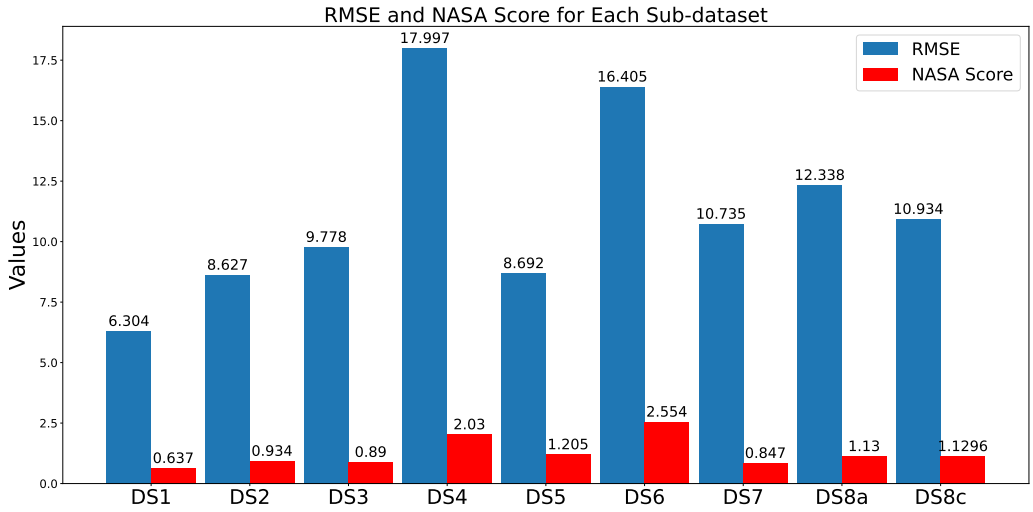
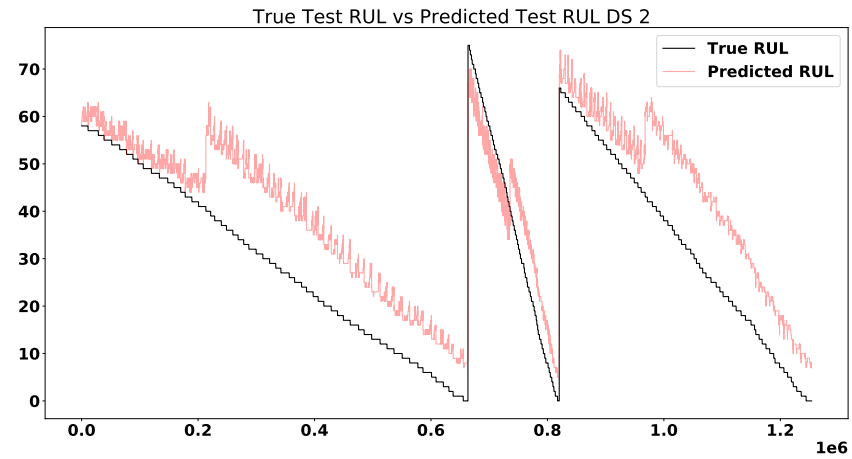


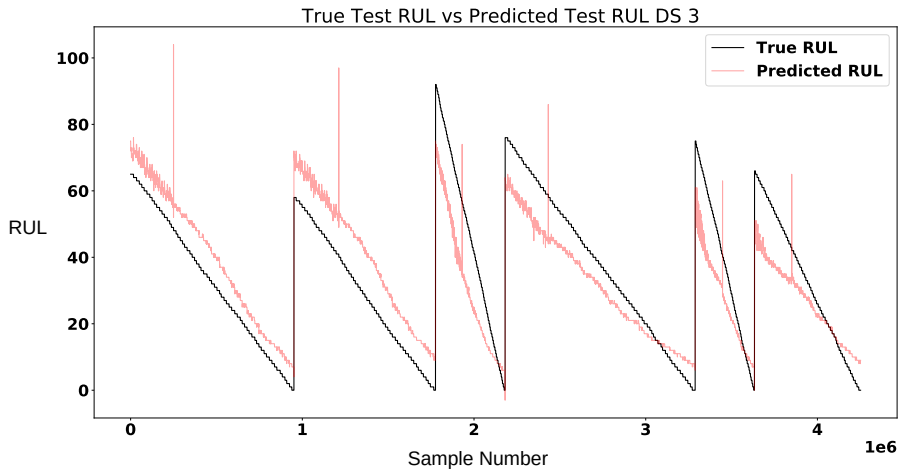

Figure 19: Visual comparison of LSTM model performance on each sub-dataset.

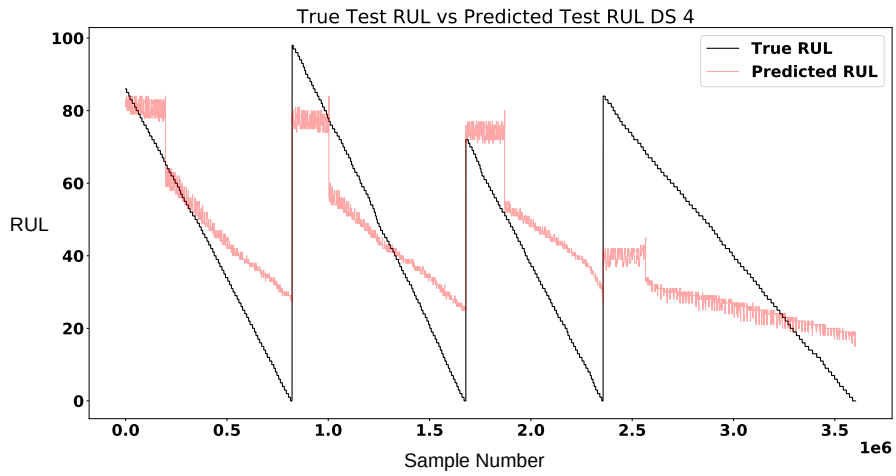
needed recurrent dropout. After applying recurrent dropout to their training, we were able to get decent performance from them but not as good as the other models.

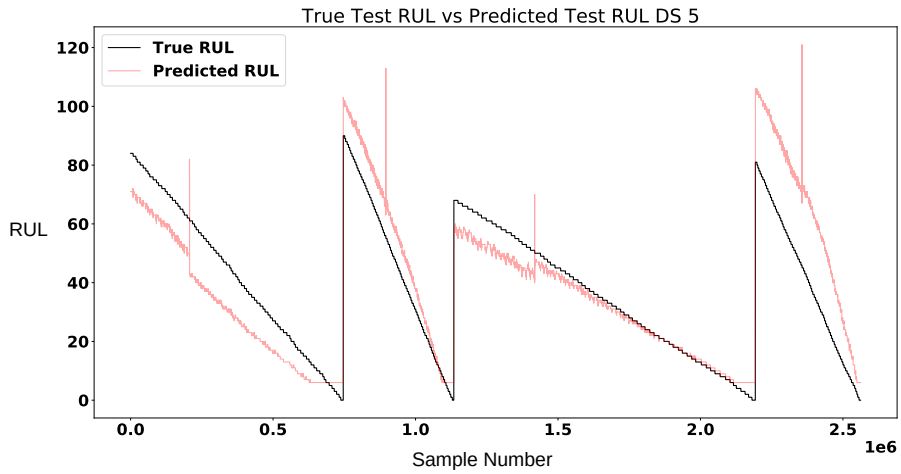
By looking at their prediction graphs in Figure 20, we can see that the model trained on DS6 is making reasonable predictions, it just overestimates the RUL (hence the high NASA score). Conversely, the model trained on DS4 makes the least-reasonable predictions of the eight models shown. It is clearly struggling to model the linear trend towards failure as well as the other models. This leads to the question, why are the models trained on DS4 and DS6 performing worse? There are a couple of possible explanations. The first explanation comes from the differences in the content of the sub-datasets. As shown in Table 2, DS4 only contains failure of the fan. DS6 contains

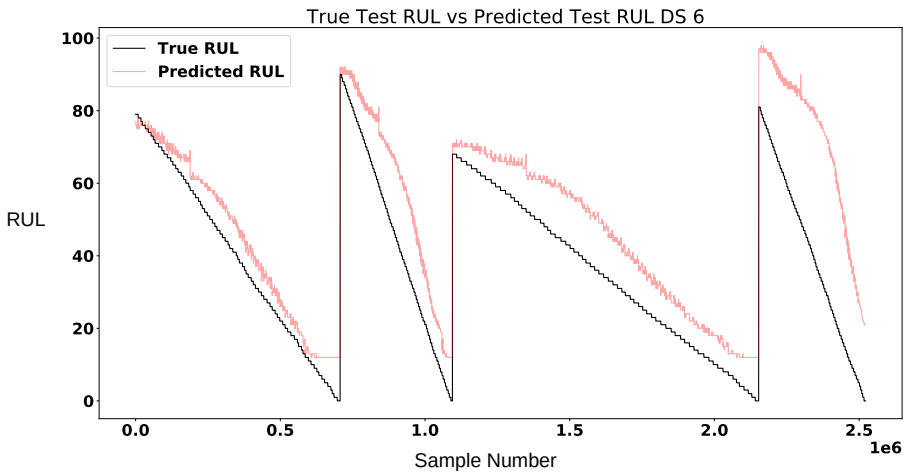

True Test RUL vs Predicted Test RUL DS 1
True RUL
Predicted RUL
175
150
125
100
75
50
25
0
0.0
0.5
1.0
1.5
2.0
2.5
1e6


True Test RUL vs Predicted Test RUL DS 2
True RUL
Predicted RUL
70
60
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e6


True Test RUL vs Predicted Test RUL DS 3
True RUL
Predicted RUL
RUL
100
80
60
40
20
0
0
1
2
3
4
1e6
Sample Number


True Test RUL vs Predicted Test RUL DS 4
True RUL
Predicted RUL
RUL
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e6
Sample Number


True Test RUL vs Predicted Test RUL DS 5
True RUL
Predicted RUL
RUL
120
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
1e6
Sample Number


True Test RUL vs Predicted Test RUL DS 6
True RUL
Predicted RUL
RUL
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
1e6
Sample Number

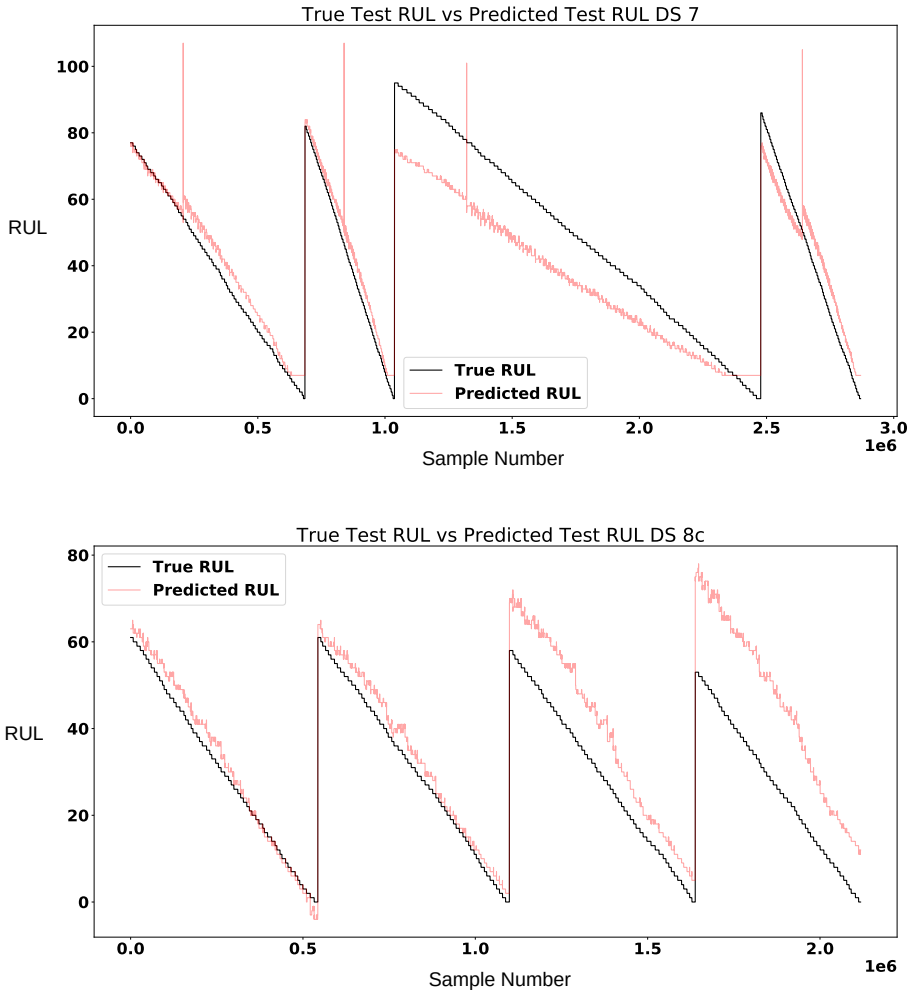



Figure 20: True vs Predicted RUL for each sub-dataset.

failure of the LPC and HPC. DS5 also contains failure of the HPC and DS5 has great performance, so maybe there is something about the fan or HPC being degraded that makes accurately generalizing to test engines difficult. A second possible explanation could lie in the distributions of the data. We chose to use the pre-split N-CMAPSS dataset, but maybe these splits do not contain the same difficulties. In this case, maybe the training sets are easier for the model to predict while the test sets contain some unseen or complex issue for one or more of the engines modeled. This would lead to the overfitting issues that we see in these two sub-datasets.

While it is interesting to look at what the model did not predict well, equally interesting observations can be gained from looking at the best models. The best model was the model trained on DS1, with the models trained on DS2, DS3, and DS7 close behind. The first model achieved a RMSE of 6.304 and a NASA score of 0.637. This is among the best in the literature, only behind the physics/deep learning hybrid model. (See Table 1.) This showcases the power of the LSTM for this problem. Using a simple LSTM model with 10 cycles of timesteps, we are able to achieve performance better than or comparable to much more complex or specialized machine learning models. Analyzing the prediction graph for DS1 from Figure 20, we see that the model appears to model the RUL of the engines very well. The predictions almost completely cover the true RUL values. When the prediction is off, it appears to be underpredicting, which is better than overpredicting in this problem. We believe that this same performance could be achieved on all of the other

sub-datasets with an LSTM model with the perfect hyperparameters and even more regularization.

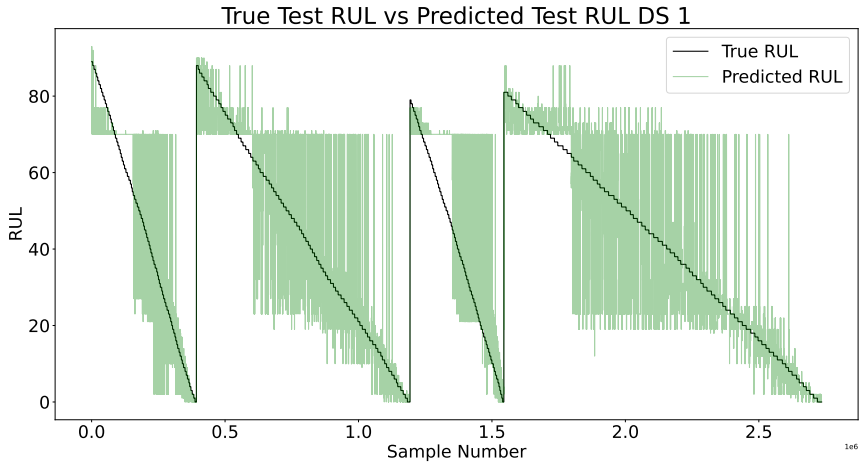
In all of the figures in the left-hand column of Figure 20, there is a prediction spike about a third of the way through each test engine's data. There is clearly some characteristic about the data that is causing this heightened prediction. Despite this, the models appear to be resilient, immediately jumping back down to reasonable predictions. This data that caused the prediction spike was included in the models' next 10 inputs, assuming that it is one observation that led to it, but the LSTM has learned to ignore it. This gives a good potential for a real-world application of these models. They appear to be strong predictors that are not impacted by an inconsistent prediction.

A final LSTM observation can be seen in nearly every graph in Figure 20. When these LSTM models are making incorrect predictions, they appear to be off by a constant. The models are consistently predicting a few cycles fewer or more than the actual RUL. However, most of the models capture the trend towards failure extremely well. This could be a generic issue that is solved with a better-generalizing model, or it could be a potential limitation of the vanilla LSTM. If the models could learn that they are simply off by a constant, some of the models would be nearly perfect. Future work could explore this idea, modifying the LSTM to deal with this issue or introducing a second model that predicts the constant that the LSTM is off by. These two models could be combined to potentially improve the performance of the

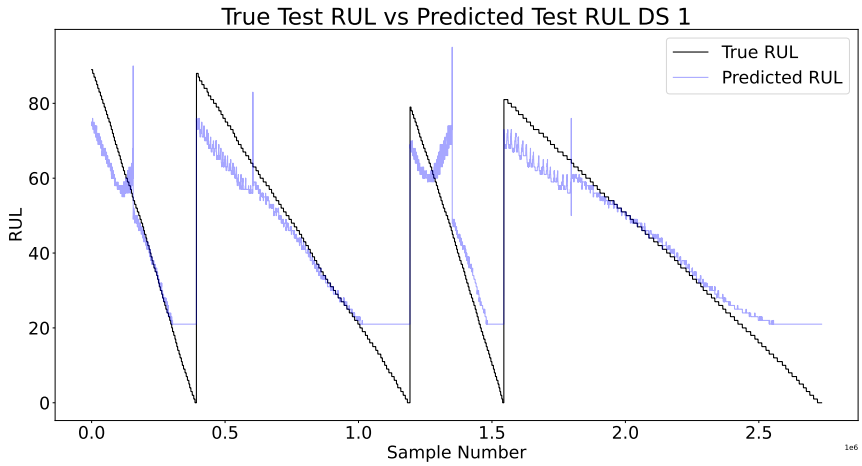
already-powerful LSTM.

Next, we will consider the predictions of the two comparison models, the Random Forest and Temporal Convolutional Network. These can be seen in Figure 21. At a glance, the Random Forest model has clearly not learned to model the remaining useful life well. It is making nonsensical predictions. In a way, this could have been expected. The Random Forest model was not given the same regularization as the LSTM and TCN models (applying dropout does not really make sense for a Random Forest). Additionally, Random Forest models are not designed to work with sequential data. They are often decent predictors on sequential data, but they do not contain the ability to capture time dependence like the LSTM and TCN models do. A Random Forest model appears to be a bad predictor of aircraft engine remaining useful life.

Conversely, the Temporal Convolutional Network shows a promising prediction graph. For the most part, it seems to model failure well. Interestingly, we still see the same spikes about a third of the way through each of the test aircraft engines as we did for the LSTM predictions for DS 1 in Figure 20. The TCN appears to lead up to these spikes more than the LSTM, temporarily predicting a higher RUL than the engine has until the prediction spike happens. The largest flaw of the TCN model can be seen as the remaining useful life drops. The TCN model predicts a constant value for the remaining useful life (20) instead of predicting below it. This renders the model useless




(a) RF on DS 1




(b) TCN on DS 1

Figure 21: True vs Predicted RUL for RF and TCN models for DS 1.

past a RUL of 20. This phenomenon only appeared after adding dropout to the Temporal Convolutional Network. A TCN without regularization performed substantially worse in terms of pure RMSE and NASA score, but

the predictions seemed much more realistic. Perhaps a usable TCN model would have to sacrifice some in terms of performance metrics in order to maintain a model that realistically models RUL.

## 3.7 Conclusion

In this paper, we attempted to solve the problem of aircraft engine remaining useful life prediction using a data-driven approach. We used the newly-released N-CMAPSS dataset of simulated aircraft flights to train and test our machine learning model. Following the suggestion of a previous paper that worked on this problem [26], we explored the use of LSTMs for this problem.

We started by preprocessing the data. Our preprocessing began with normalizing the data. Next, we performed principal components analysis to reduce the complexity of the data. We selected the first five principal components to be the factors we would train our model on, as they explained over 90% of the variance. Finally, we reshaped the data to include a timestep of 10, giving each data point the context of the previous 10 data points.

When beginning model training, we noticed that the LSTM models were overfitting the training sets extremely quickly. To combat this, we used dropout in the form of a dropout layer and recurrent dropout within the LSTM. We iteratively trained 9 LSTM models on the sub-datasets of N-CMAPSS. We also trained Random Forest and Temporal Convolutional Network models on the first sub-dataset. All of our LSTM models achieved

good results, with our first model performing better than almost every model in the literature, only being outperformed by a physics/deep-learning hybrid model that is tailored to this problem [20]. Our LSTM model outperformed the RF and TCN models. We analyzed the performance of our models on each sub-dataset to highlight some potential challenges within the N-CMAPSS dataset. Additionally, we looked at the predictions of each model on their test sets to see what kind of predictions the models are making on unseen data. This demonstrated the success of many of our models and points directly to what gaps in performance future models will need to fill.

### 3.7.1 Future Work

Future work should further explore additional methods of reducing overfitting on the sub-datasets. Despite using recurrent dropout and a dropout layer, we were unable to achieve even performance among each model. The model trained on sub-dataset four is the most notable example. The model trained on that dataset was easily overfitting, but our methods of reducing the training performance did not improve its generalizability to the same degree as on the other sub-datasets.

Future work should also consider methods of shifting predictions for RUL prediction problems. Many of our models appeared to be off by a constant for individual test aircraft engines. Finding a way to correctly reduce this constant while still modeling the trend of failure would greatly improve the performance on this and many more prognostics datasets.

Finally, future work should consider the training of an LSTM on the entirety of the N-CMAPSS dataset. This could lead to a model that can predict the remaining useful life of aircraft engines for any failure mode.

# 4 Conclusion

In this thesis, I began by introducing the problem of device failure and the theory behind RNNs and LSTMs. I followed by demonstrating two applications of LSTMs for RUL prediction. First, I predicted the RUL of hard-disk drives on a time window of 60 and 120 days away from failure. By training a bidirectional LSTM model 60 days from failure, I was able to achieve state-of-the-art performance with an accuracy of 96.4%. In this problem, I was also able to demonstrate the effectiveness of standardizing data per device as well as its impacts on extrapolation. Finally, I predicted the RUL of aircraft engines using NASA's N-CMAPSS dataset. I trained 8 LSTM models on each of the sub-datasets. I was able to achieve a RMSE of 6.3 which is the second-best of all the models in the literature. Additionally, I compared the performance of an LSTM model to the performance of Random Forest and Temporal Convolutional Neural Network models, demonstrating the LSTMs superior predictive capabilities. By applying LSTMs to these two problems, I have demonstrated the predictive performance of LSTMs on the problem of remaining useful life prediction.

This thesis provides evidence for the effectiveness of LSTM models for RUL prediction. I also provided an in-depth look at the model development process for these types of problems. The experimentation I performed is reproducible and should be able to be transferred to other RUL prediction and sequential data prediction problems. In these experiments, I encountered problems such

as overfitting and lack of generalization. I explored potential solutions to these general machine learning problems that were able to improve the performance of the models on their respective tasks. These solutions could be applied to a variety of models across the field of machine learning. I hope that models like the ones I have presented can be implemented in these real-world devices and systems to improve their reliability and maintenance, eventually helping to prevent the pressing problem of device failure.

# References


[1] URL: https://www.backblaze.com/b2/hard-drive-test-data.html%5C#how-you-can-use-the-data.

[2] *10 least reliable car brands - Douglas D. Law, esq.* Nov. 2019. URL: https://californialemonlawguide.com/warranty-directs-10-least-reliable-car-brands/.

[3] *2021 PHM Conference Data Challenge.* Oct. 2021. URL: https://data.phmsociety.org/2021-phm-conference-data-challenge/.

[4] *Air traffic by the numbers.* Nov. 2021. URL: https://www.faa.gov/air%5C_traffic/by%5C_the%5C_numbers/.

[5] "Aircraft engine run-to-failure dataset under real flight conditions for prognostics and diagnostics". In: *Data* 6 (1 Jan. 2021). ¡br/¿, pp. 1–14. ISSN: 23065729. DOI: 10.3390/data6010005.

[6] Afshine Amidi and Shervine Amidi. *Recurrent neural networks cheatsheet.* URL: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[7] Dawn An, Joo-Ho Choi, and Nam Kim. "A tutorial for model-based prognostics algorithms based on Matlab code". In: *Proceedings of the Annual Conference of the Prognostics and Health Management Society 2012, PHM 2012* (Jan. 2012), pp. 224–232.

[8] Preethi Anantharaman, Mu Qiao, and Divyesh Jadav. "Large Scale Predictive Analytics for Hard Disk Remaining Useful Life Estimation". In: *2018 IEEE International Congress on Big Data (BigData Congress)* (2018). DOI: 10.1109/bigdatacongress.2018.00044.

[9] Nicolas Aussel et al. "Predictive Models of Hard Drive Failures Based on Operational Data". In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)* (2017). DOI: 10.1109/icmla.2017.00-92.

[10] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence*

*Modeling*. 2018. DOI: 10.48550/ARXIV.1803.01271. URL: https://arxiv.org/abs/1803.01271.

[11] Michael Ball et al. *Total Delay Impact Study A Comprehensive Assessment of the Costs and Impacts of Flight Delay in the United States*.

[12] Sanchita Basak, Saptarshi Sengupta, and Abhishek Dubey. "Mechanisms for Integrated Feature Normalization and Remaining Useful Life Estimation Using LSTMs Applied to Hard-Disks". In: *2019 IEEE International Conference on Smart Computing (SMARTCOMP)* (2019). DOI: 10.1109/smartcomp.2019.00055.

[13] Sanchita Basak et al. "Spatio-temporal AI inference engine for estimating hard disk reliability". In: *Pervasive and Mobile Computing* 70 (2021), p. 101283. DOI: 10.1016/j.pmcj.2020.101283.

[14] Brian Beach. *Hard Drive SMART Stats*. Apr. 2021. URL: https://www.backblaze.com/blog/hard-drive-smart-stats/.

[15] Luca Biggio et al. "Uncertainty-Aware Prognosis via Deep Gaussian Process". In: *IEEE Access* 9 (2021), pp. 123517–123527. ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3110049.

[16] E Chandra Blessie and E Karthikeyan. "Sigmis: A Feature Selection Algorithm Using Correlation Based Method". In: *Journal of Algorithms & Computational Technology* 6 (3 2012).

[17] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/a:1010933404324.

[18] Danilo Bzdok, Naomi Altman, and Martin Krzywinski. "Statistics versus Machine Learning". In: *Nature Methods* 15 (Apr. 2018). DOI: 10.1038/nmeth.4642.

[19] *C-MAPSS aircraft engine simulator data*. URL: https://data.nasa.gov/dataset/C-MAPSS-Aircraft-Engine-Simulator-Data/xaut-bemq.

[20] Manuel Arias Chao et al. "Fusing physics-based and deep learning models for prognostics". In: *Reliability Engineering and System Safety* 217 (Jan. 2022). ISSN: 09518320. DOI: 10.1016/j.ress.2021.107961.

[21] Iago C. Chaves et al. "BaNHFaP: A Bayesian Network Based Failure Prediction Approach for Hard Disk Drives". In: *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)* (2016). DOI: 10.1109/bracis.2016.083.

[22] Iago C. Chaves et al. "Hard Disk Drive Failure Prediction Method Based On A Bayesian Network". In: *2018 International Joint Conference on Neural Networks (IJCNN)* (2018). DOI: 10.1109/ijcnn.2018.8489097.

[23] Zhenghua Chen et al. "Machine Remaining Useful Life Prediction via an Attention-Based Deep Learning Approach". In: *IEEE Transactions on Industrial Electronics* 68.3 (2021), pp. 2521–2531. DOI: 10.1109/tie.2020.2972443.

[24] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014). DOI: 10.3115/v1/d14-1179.

[25] Austin Coursey et al. "Remaining Useful Life Estimation of Hard Disk Drives using Bidirectional LSTM Networks". In: *2021 IEEE International Conference on Big Data (Big Data)*. 2021, pp. 4832–4841. DOI: 10.1109/BigData52589.2021.9671605.

[26] Nathaniel Devol et al. "Inception Based Deep Convolutional Neural Network for Remaining Useful Life Estimation of Turbofan Engines". In: *Proceedings of the Annual Conference of the PHM Society 2021* (Dec. 2021).

[27] By: IBM Cloud Education. *What is unsupervised learning?* URL: https://www.ibm.com/cloud/learn/unsupervised-learning.

[28] Hatem M. Elattar, Hamdy K. Elminir, and A. M. Riad. "Prognostics: A literature review". In: *Complex &; Intelligent Systems* 2.2 (2016), pp. 125–154. DOI: 10.1007/s40747-016-0019-3.

[29] Kai Goebel, Bhaskar Saha, and Abhinav Saxena. "A comparison of three data-driven techniques for prognostics". In: *62nd Meeting of the Society For Machinery Failure Prevention Technology (MFPT)* (Jan. 2008), pp. 119–131.

[30] A. Graves and J. Schmidhuber. "Framewise phoneme classification with bidirectional LSTM networks". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 4. 2005, 2047–2052 vol. 4. DOI: 10.1109/IJCNN.2005.1556215.

[31] Gaoyang Guo et al. "Who is answering whom? Finding "Reply-To" relations in group chats with deep bidirectional LSTM networks". In: *Cluster Computing* 22.S1 (2018), pp. 2089–2100. DOI: 10.1007/s10586-018-2031-4.

[32] Zhimin He, Hao Yang, and Min Xie. "Statistical modeling and analysis of hard disk drives (HDDs) failure". In: *2012 Digest APMRC.* 2012, pp. 1–2.

[33] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (Apr. 1998), pp. 107–116. DOI: 10.1142/S0218488598000094.

[34] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[35] Lihan Hu et al. "A disk failure prediction method based on LSTM network due to its individual specificity". In: *Procedia Computer Science* 176 (2020), pp. 791–799. DOI: 10.1016/j.procs.2020.09.074.

[36] Xiaodong Jia et al. *Review of PHM Data Competitions from 2008 to 2017: Methodologies and Analytics*.

[37] Ian T. Jolliffe and Jorge Cadima. "Principal component analysis: A review and recent developments". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), p. 20150202. DOI: 10.1098/rsta.2015.0202.

[38] M. I. Jordan and T. M. Mitchell. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245 (2015), pp. 255–260. DOI: 10.1126/science.aaa8415. URL: https://www.science.org/doi/abs/10.1126/science.aaa8415.

[39] Andy Klein. *2020 Hard Drive Reliability Report by Make and Model*. May 2021. URL: https://www.backblaze.com/blog/backblaze-hard-drive-stats-for-2020/.

[40] Colin Lea et al. "Temporal Convolutional Networks for Action Segmentation and Detection". In: *CoRR* abs/1611.05267 (2016). arXiv: 1611.05267. URL: http://arxiv.org/abs/1611.05267.

[41] *Left Engine Failure and Subsequent Depressurization Southwest Airlines Flight 1380*. Apr. 2018. URL: https://www.ntsb.gov/investigations/AccidentReports/Reports/AAR1903.pdf.

[42] Jiachong Li. "Regression and Classification in Supervised Learning". In: *Proceedings of the 2nd International Conference on Computing and Big Data*. ICCBD 2019. Taichung, Taiwan: Association for Computing Machinery, 2019, pp. 99–104. ISBN: 9781450372909. DOI: 10.1145/3366650.3366675. URL: https://doi.org/10.1145/3366650.3366675.

[43] Ping Li, Roger Goodall, and Visakan Kadirkamanathan. "Parameter estimation of railway vehicle dynamic model using Rao-Blackwellised particle filter". In: (Jan. 2003).

[44] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. "Recurrent Neural Network for Text Classification with Multi-Task Learning". In: *Proceed-*

*ings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (July 2016).

[45] Qiong Liu and Ying Wu. "Supervised Learning". In: (Jan. 2012). DOI: 10.1007/978-1-4419-1428-6_451.

[46] Andreas Lövberg. "Remaining Useful Life Prediction of Aircraft Engines with Variable Length Input Sequences". In: *Proceedings of the Annual Conference of the PHM Society 2021* ().

[47] Sidi Lu et al. "Making Disk Failure Predictions SMARTer!" In: *18th USENIX Conference on File and Storage Technologies (FAST 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 151–167. ISBN: 978-1-939133-12-0. URL: https://www.usenix.org/conference/fast20/presentation/lu.

[48] Jianhui Luo et al. "Model-based prognostic techniques [maintenance applications]". In: *Proceedings AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference.* 2003, pp. 330–340. DOI: 10.1109/AUTEST.2003.1243596.

[49] Kamal Medjaher, Noureddine Zerhouni, and J. Baklouti. "Data-driven prognostics based on health indicator construction: Application to PRONOSTIA's data". In: *2013 European Control Conference (ECC)* (2013), pp. 1451–1456.

[50] Bernhard Mehlig. *Machine Learning with Neural Networks*. Cambridge University Press, Oct. 2021. DOI: 10.1017/9781108860604. URL: https://doi.org/10.1017%2F9781108860604.

[51] *Nvidia GeForce RTX 2080 TI review*. Apr. 2022. URL: https://benchmarks.ul.com/hardware/gpu/NVIDIA+GeForce+RTX+2080+Ti+review.

[52] Shuai Pang et al. "A combined Bayesian network method for predicting drive failure times from SMART attributes". In: *2016 International Joint Conference on Neural Networks (IJCNN)* (2016). DOI: 10.1109/ijcnn.2016.7727837.

[53] Vu Pham et al. "Dropout improves recurrent neural networks for handwriting recognition". In: *2014 14th International Conference on Frontiers in Handwriting Recognition* (2014). DOI: 10.1109/icfhr.2014.55.

[54] Usman Pirzada. *German retailer mindfactory reports higher failure rate for Radeon 5000 series gpus than Nvidia Turing*. Aug. 2020. URL: https://wccftech.com/mindfactory-report-amd-gpus-fail-more-often-than-those-from-nvidia/.

[55] Emmanuel Ramasso, Abhinav Saxena, and Analysis Performance Benchmarking. *Performance Benchmarking and Analysis of Prognostic Methods for CMAPSS Datasets. Performance Benchmarking and Analysis of Prognostic Methods for CMAPSS Datasets*. 2014, pp. 1–15. URL: https://hal.archives-ouvertes.fr/hal-01324587.

[56] *S.M.A.R.T. Attributes*. URL: https://www.ntfs.com/disk-monitor-smart-attributes.htm.

[57] Tara N. Sainath et al. "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2015). DOI: 10.1109/icassp.2015.7178838.

[58] Haşim Sak, Andrew Senior, and Françoise Beaufays. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*. 2014. arXiv: 1402.1128 [cs.NE].

[59] "Selecting critical features for data classification based on machine learning methods". In: *Journal of Big Data* 7 (1 Dec. 2020). ISSN: 21961115. DOI: 10.1186/s40537-020-00327-4.

[60] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. "Recurrent Dropout without Memory Loss". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 1757–1766. URL: https://aclanthology.org/C16-1165.

[61] Jing Shen et al. "Hard Disk Drive Failure Prediction for Mobile Edge Computing Based on an LSTM Recurrent Neural Network". In: *Mobile Information Systems* 2021 (2021), pp. 1–12. DOI: 10.1155/2021/8878364.

[62] Jing Shen et al. "Random-forest-based failure prediction for hard disk drives". In: *International Journal of Distributed Sensor Networks* 14.11 (2018). DOI: 10.1177/1550147718806480.

[63] Jonathon Shlens. *A tutorial on principal component analysis*. Apr. 2014. URL: https://arxiv.org/abs/1404.1100.

[64] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. "A comparison of Arima and LSTM in forecasting time series". In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)* (2018). DOI: 10.1109/icmla.2018.00227.

[65] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. "A review of supervised machine learning algorithms". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2016, pp. 1310–1315.

[66] David Solis-Martin, Juan Galan-Paez, and Joaquin Borrego-Diaz. "A stacked deep convolutional neural network to predict the remaining useful life of a turbofan engine". In: *Proceedings of the Annual Conference of the PHM Society 2021* (Dec. 2021). URL: http://arxiv.org/abs/2111.12689%20http://dx.doi.org/10.36001/phmconf.2021.v13i1.3110.

[67] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[68] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2020.

[69] Alper Tokgöz and Gözde Ünal. "A RNN based time series approach for forecasting turkish electricity load". In: *2018 26th Signal Processing and Communications Applications Conference (SIU)*. 2018, pp. 1–4. DOI: 10.1109/SIU.2018.8404313.

[70] *Trend report: Q4 2017 State of Mobile Device Repair &; Security*. URL: https://freebit.cz/wp-content/uploads/2018/08/en-rs-q4-2017-state-of-mobile-device-repair-and-security.pdf.

[71] Vishnu TV et al. *Data-driven Prognostics with Predictive Uncertainty Estimation using Ensemble of Deep Ordinal Regression Models*. 2019. DOI: 10.48550/ARXIV.1903.09795. URL: https://arxiv.org/abs/1903.09795.

[72] Simon Vollert and Andreas Theissler. "Challenges of machine learning-based RUL prognosis: A review on NASA's C-MAPSS data set". In: IEEE, Sept. 2021, pp. 1–8. ISBN: 978-1-7281-2989-1. DOI: 10.1109/ETFA45728.2021.9613682. URL: https://ieeexplore.ieee.org/document/9613682/.

[73] Guochao Wang, Yu Wang, and Xiaojie Sun. "Multi-Instance Deep Learning Based on Attention Mechanism for Failure Prediction of Unlabeled Hard Disk Drives". In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–9. DOI: 10.1109/tim.2021.3068180.

[74] Yu Wang et al. "A Two-Step Parametric Method for Failure Prediction in Hard Disk Drives". In: *IEEE Transactions on Industrial Informatics* 10.1 (2014), pp. 419–430. DOI: 10.1109/tii.2013.2264060.

[75] Yu Wang et al. "Failure Prediction of Hard Disk Drives Based on Adaptive Rao–Blackwellized Particle Filter Error Tracking Method". In: *IEEE Transactions on Industrial Informatics* 17.2 (2021), pp. 913–921. DOI: 10.1109/tii.2020.3016121.

[76] Liangwei Zhang et al. "A Review on Deep Learning Applications in Prognostics and Health Management". In: *IEEE Access* 7 (2019), pp. 162415–162438. DOI: 10.1109/ACCESS.2019.2950985.