# Developing Deep Learning Models for System Remaining Useful Life Predictions: Application to Aircraft Engines

Timothy Darrah[1], Andreas Lövberg[2], Jeremy Frank[3], Marcos Quinones-Gruiero[4], Gautam Biswas[5]

[1,4,5] *Vanderbilt University, Institute for Software Integrated Systems, Nashville, TN, USA*
*timothy.s.darrah@vanderbilt.edu*
*marcos.quinones.gruiero@vanderbilt.edu*
*gautam.biswas@vanderbilt.edu*

[2] *RISE Research Institute of Sweden, System Integration Unit, Mölndal, Sweden*
*andreas.lovberg@ri.se*

[3] *NASA Ames Research Center, Intelligent Systems Division, Mountain View, CA, USA*
*andreas.lovberg@ri.se*

## Abstract

Prognostics and health management (PHM) is an important part of ensuring reliable operations of complex safety-critical systems. System-level remaining useful life (RUL) estimation is a much more complex problem than making estimations at the component level. Model-based approaches have traditionally worked in the past for components such as capacitors, MOSFETs, batteries, or hard-drives (to name a few examples), but developing high fidelity dynamics models of cyber physical systems that can be used to study the effects of multiple degrading components in the system remains a challenging task. Hybrid and pure data driven approaches have shown to be much more promising, and in this work, we propose an end-to-end data-driven framework for developing deep learning models to predict remaining useful life of turbofan jet engines operating under unknown faulty conditions. The raw data is organized with a data schema that improves the model development process and down stream data analysis tasks. The raw sensor data is transformed into signals that expose the underlying degradation processes, which are then used for model development. Bayesian Optimization is used to tune the model parameters prior to training and validation. We show that this approach results in accurate predictions within 3 cycles to end of life (EOL). We demonstrate the effectiveness of our approach by applying it to the N-CMAPSS turbofan engine dataset recently released by NASA, which includes high fidelity degradation modeling, real world operating conditions, and a large set of fault operating modes.

## 1. Introduction

Condition-Based Maintenance (CBM) is a method of scheduling and performing maintenance activities based on the operational history of the system and its current state. Many organizations have been successful with integrating this approach to maintenance within their concept of operations while maintaining time-based inspections and parts replacement. This approach does not account for the expected *future usage* of the system, and, therefore, accurate Remaining Useful Life (RUL) estimates cannot be derived. RUL is a measure of how long the system can remain in operation and meet all of its safety and performance goals until a threshold violation occurs, at which point we say the system has reached its End of Life (EOL) [Goebel et al., 2017]. Prognostics brings together the study of how systems fail with life-cycle management to ensure safe and proper operations of the system [Peng et al., 2010]. Prognostics-Based Maintenance (PBM) uses this information and goes one step beyond CBM to further improve safety, maintenance, and operation activities.

The primary barrier to developing and maturing prognostics technologies, such as PBM is the lack of real world run-to-failure data or high fidelity run-to-failure simulated data. The Commercial Modular Aero-Propulsion System Simulation (CMAPSS) dataset has been available for quite some time but lacked real-world flight characteristics and multi-source degradation. Recently, an updated version of this dataset was released that addresses these shortcomings and includes real-world operating conditions [Chao et al., 2021]. We use this dataset to define an end-to-end framework for the development of deep learning models that can be used for

PBM applications.

### 1.1. Problem Description

Data-driven methods for prognostics use probability-based [Si et al., 2011] and machine learning algorithms [Schwabacher and Goebel, 2007] to map component measurements to the degradation parameters of a component or the health state of a system, and need run-to-failure datasets for training. This entails many unique challenges that can result in poor performance or complete failure to predict without a systematic approach to data management and model development. Proper data management is necessary due to the large volumes of data and the complex relationships among data features. Such data management principles are lacking in the PHM community, and result in a higher degree of effort spent on data processing and validation tasks, redundant use of code, and difficulty in reproducing results. This means that often times the same processing pipeline used for one application cannot readily be used in another. There are many great publications on model architectures and developing models for various prognostics applications, however many of these are application-specific, and lack the necessary abstraction to generalize across domains. By utilizing the same software engineering principles found in enterprise software development, we have taken what was once an application-specific solution and show here how it is generalized with use in a different domain.

### 1.2. Approach

We take an end-to-end approach to model development starting with data organization *first*, then data preprocessing steps, followed by a two-stage hyperparameter search, and finally the training and validation steps. Our approach is motivated by the need for developing more formally defined *Automated Machine Learning* (AutoML) methods and data processing pipelines for prognostics. AutoML is defined as a *systematic* and *efficient* processes of developing machine learning models with minimal user input [Hutter et al., 2018]. The field of AutoML is quite large and we specifically focus on data management and hyperparameter optimization (as opposed to meta-learning and neural architecture search methods).

The dataset is provided as a series of HDF5 files containing telemetry data, degradation data, virtual sensor data, flight conditions, and auxiliary information such as the unit number and flight class. As discussed in previous work [Darrah et al., 2021], employing best practices when it comes to data management for prognostics or machine learning tasks is critical to developing robust models and reproducible results. This data management framework simplifies the entire data processing and model development pipeline, which is discussed in Section 2. We generalize the same data management framework previously used for UAV operations to accommodate other applications and use it with the New Commercial Modular Aero-Propulsion System Simuation (N-CMAPSS) [Chao et al., 2021], which contains 90 run-to-failure simulations. A two-stage hyperparameter search is employed to search over regularizing parameters separately from network parameters due to the short search horizon and the effect of regularization on convergence speed. The network is then trained, validated, and tested with the 90 units in the dataset.

### 1.3. Contribution

The data management framework used in this work was originally developed for UAV simulations [Darrah et al., 2021, Darrah et al., ], and in those works an in-depth review of data management practices and the inefficiencies in current open-source frameworks for prognostics are discussed. Key patterns have been abstracted to generalize to other applications and here we show how such a framework can be applied to the N-CMAPSS dataset as an example of its generalization capabilities. This serves as the foundation for a model development pipeline in effort to streamline automation, data integrity, and reproducability of results. The primary steps in this pipeline include transforming raw telemetry data into features that expose the unobservable degradation, and then performing a two-stage hyperparameter search. This separates searching network parameters from regularization parameters due to the tendency for regularization to have a higher loss during the early stages of training, and, therefore, score lower than other parameter configurations. Finally we compare fixed length input sequences to variable length input sequences and show that using variable length input sequences result in more accurate predictions, however there were not enough validation units to be conclusive. The key contribution is a demonstration of the overall model development pipeline, which encapsulates these steps and is a repeatable framework which can be applied to any data-driven task for prognostics.

### 1.4. Paper Organization

The paper is organized as follows. Section 2 discusses the overall model development pipeline, from raw data to final results. Section 3 discusses the implementation of the data management framework with the N-CMAPSS dataset. Sections 4 through 6 discuss the subsequent steps in the pipeline, namely preprocessing, hyperparameter search, and training & validation. Section 7 presents the results and discussion, followed by the conclusion and future work in Section 8.

## 2. Methodology

Over the last several years, there has been a rise in the use of deep learning methods for prognostics due to the complexities inherent in many cyber physical systems and the in-

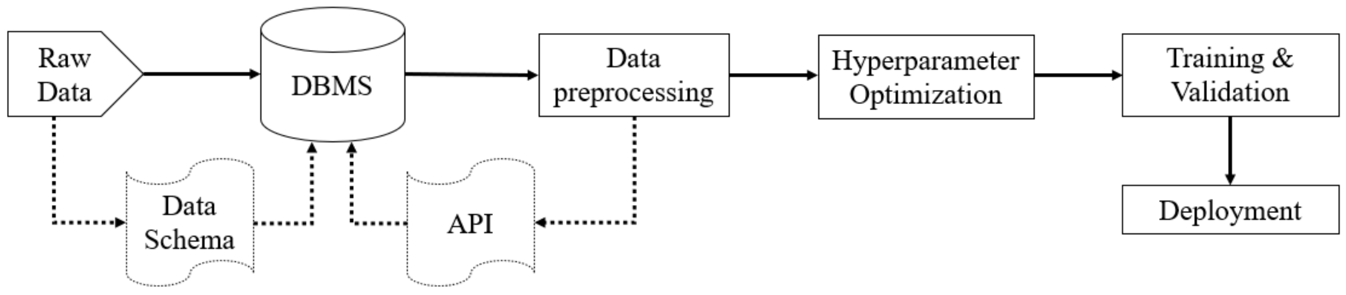

Figure 1. Model Development Framework

ability for model-based methods to achieve high quality results. However, there is a general lack of data management and attention to data provenance with these studies, which result in poor data descriptors, unverifiable origins, and other issues that make replication or validation tasks difficult or not possible. We argue that these issues persist due to inadequate data management standards and policies. The goal of data management is to produce *"self-describing datasets"* [Strasser et al., 2012] such that scientists and practitioners can discover, use, and interpret the data across multiple experiments. This fits well within the context of machine learning and prognostics applications for cyber physical systems. Machine learning models are highly dependent on the underlying data, and, therefore, consistency, accuracy, and completeness of the data is essential to train models that are capable of sufficient generalization and performance [Munappy et al., 2019]. Thus, good data management principles and practices need to be adopted throughout the entire development process. This is the cornerstone of the data-driven model development pipeline for prognostics, shown in Figure 1.

The three critical pieces to implementing this framework are the Database Management System (DBMS), the Data Schema, and the Application Programming Interface (API). When these are properly implemented, every subsequent task is simplified and collaborative efforts are more easily carried out. These benefits do not come without upfront cost, however, and the raw data must be thoroughly understood in order to define an appropriate data schema. Simply inserting data into a database without making careful decisions about data types, data constraints, relationships among data features, or how the dataset should be organized will inevitably lead to trouble down the road, and not provide any benefit. The point of all this is to organize the data such that it can indefinitely persist without structural changes, and be extensible. The data schema should be defined once, and not have to change on any substantial level. On the other side of the data schema is the API, which is used by the engineer to retrieve data from the database without writing complex queries or blocks of code to read and parse CSV files. This makes it easier for them to focus on the data processing and model development tasks. The details of the data schema, organization, and API are discussed in Section 3.

The rest of the pipeline should be quite familiar to the machine learning engineer. Filling missing values, time-alignment, and normalization are all standard data preprocessing tasks. An additional task specific to prognostics is carried out as well, namely, transforming the raw signal data into residuals that expose the underlying degradation without actually knowing the degradation functions. This is further detailed in Section 4. The next step is to perform hyperparameter optimization. However, unlike the typical approach of searching over all parameters together, we search over network parameters and regularization parameters separately. This is due to the fact that regularization tends to increase the time to convergence and these configurations tend to score poorly when evaluating parameter configurations on short horizons (i.e. evaluate after the third epoch). This is discussed in more detail in Section 5. The next step consists of training and validation, and we highlight modifications to the training process such as stochastic weight averaging and learning weight decay in Section 6. The final step of the pipeline is deployment to a production environment. In this work we simulate a production environment with 10 test units.

This entire process starts with a well defined data schema, and understanding the structure of the dataset is a necessary requirement. The dataset and implementation of the data management framework are discussed next.

## 3. Data Management Framework

In this work, the N-CMAPSS [Chao et al., 2021] dataset is used to demonstrate the implementation of the data management framework and model development pipeline for prognostics. Previously, this framework was developed for UAV simulations [Darrah et al., 2021], but here we show how it can accommodate other applications and in this section walk the reader through implementing it with the N-CMAPSS dataset. The dataset provides simulated run-to-failure trajectories comprising of 90 commercial jet engines (units) with unknown degradation processes and initial health states. Each unit is assigned one of three flight classes ($Fc$) determined by

| Symbol | Variable | Description |
|---|---|---|
| $w$ | flight conditions | real flight data |
| $x_s$ | telemetry data | measured system signals |
| $x_v$ | virtual sensors | not used |
| $\theta$ | degradation parameters | unobservable |
| $\alpha$ | asset data | auxiliary data for each unit |

Table 1. Dataset Variables

For a more complete description of the dataset variables, see [Chao et al., 2021].

the flight length and all flights for that unit are of the same flight class. $Fc_1$ consists of flights under 3 hours, $Fc_2$ consists of flights between 3 and 5 hours, and $Fc_3$ is for flights with durations greater than 5 hours. Each unit contains five types of variables shown in Table 1.

The data is originally provided as a set of files in hierarchical data format (HDF5), which is an efficient way to store data for scientific and engineering purposes. However, it is not well suited to convey relationships among data features or enforce constraints, nor is it suited to create specific and complex data queries. There are also no mechanisms to streamline access to a common store of data for multiple people to use concurrently. Because relationships cannot be adequately captured, each data record contains redundant information that could be extracted and separately stored, reducing the memory footprint of the data. As an example, instead of including multiple columns of data that describe the unit with each record of sensor data, only one column is needed to *reference* the unit data, which is kept separately. Furthermore, data operations can be significantly improved with the use of a DBMS. As an example, retrieving sensor and other data for a set of 10 runs from raw files using the pandas library took 29.2 seconds, while using a database query took only 5.9 seconds. There are over 60M records in the dataset, and minimizing the number of columns will have a significant impact on the amount of disk space used, and subsequently the speed and efficiency of loading and manipulating the data. The implementation of the data management framework with this dataset is discussed next.

## 3.1. Framework Implementation

The framework*[1] [Darrah et al., 2021] centers around assets, processes, and data. Assets are abstractions of user defined components which act as *first class objects* and is the archetype model that all components inherit from. The asset itself is a container for user defined components that are used for linkage within the framework. Components are affected by processes, which themselves can be internal (degradation via damage accumulation) or external (environmental influences). Together, components and processes generate data which is linked to the system and other information pertaining to the usage of that system via the summary table. All usage-based data tables link to the *summary table*, which in turn links to the system, the assets installed on the system, and the processes affecting the system *at that time*. With this dataset, the degradation process coefficients are not known, and therefore no process models are stored. Also, different from previous work [Darrah et al., 2021], where the system was implemented as a container for several components, here the system and component are one and the same, the engine. The framework can be implemented in a multitude of ways due to the flexibility and generalization capabilities inherent in the asset-process-data paradigm. Figure 2 depicts the implementation of the framework with the N-CMAPSS dataset. In this work, we are not performing simulation, and therefore do not need to be concerned with the physics of turbofan jet engine operation. In a similar manner, we do not need to implement the degradation functions - these steps have already been completed in the data generation simulation process to create the dataset. Therefore, we can restrict our efforts to only defining the table schema for the data in the dataset.

[1]A pre-release version of the framework is available at https://github.com/darrahts/data_management_framework. We welcome those interested in contributing to this work.

The first table to define is the *asset type* table (`asset_type_tb`), which is used in the underlying table schema as a means of organization and proper data linkage. There is only one entry in this table, shown below in Table 2. For every entry in this table, the `type` and `subtype` fields are used to dynamically create a new *user-defined* component table for that type. For the N-CMAPSS dataset, this is is the `engine_ncmapss_tb`, and shown in Table 3.

| id | type | subtype | description |
|---|---|---|---|
| 1 | engine | ncmapss | N-CMAPSS unit |

Table 2. Asset Type Table

All systems and components have an asset type, which is used as a means of organizing the data.

Each field in Table 3 must be defined by the user and can contain any type of data suited for any purpose. For example, if physics-based models are known, this table could contain the model parameters. Each record then would be a specific model or variation of the same model with different coefficients. However, the use case in this work is different as mentioned above, and therefore the definition of this table is specifically related to the existing data fields in the dataset. The `id` field is auto generated, and the `group_id` field maps the unit to the *dev* or *test* sets. The `unit` and `Fc` fields were previously in the auxiliary data partition, with redundant information being repeated with every single record entry. Now, the redundancy has been eliminated. This table is separate from the *asset* table (`asset_tb`), which is a *framework* table, but the two tables are automatically linked by the `id` field, and a record must exist in the asset table

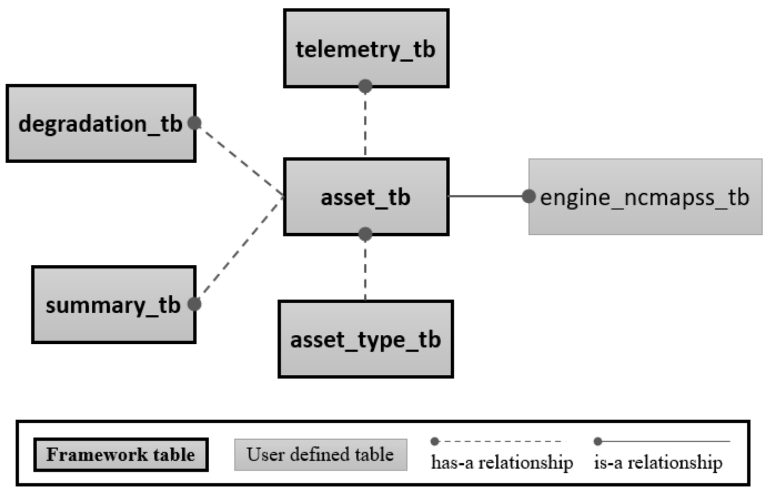

Figure 2. Table Schema for N-CMAPSS Dataset

first, shown in Table 4.

| id | group_id | Fc | unit | dataset |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | DS01-005 |
| 2 | 1 | 3 | 2 | DS01-005 |
| ... | | | | |
| 89 | 2 | 2 | 9 | DS08c-008 |
| 90 | 2 | 2 | 10 | DS08c-008 |

Table 3. User Defined Component Table

The component table is automatically created when an asset type is stored. Unlike other tables, which are predefined, this table is defined by the user after creation.

The primary purpose of asset table is for data organization and proper linkage of meta data and data records. There is a record for each of the 90 units in the *dev* and *test* set, encoded as the `group_id` in Table 3 (1 for dev, 2 for test). The unit numbers in the original dataset repeat across data files, meaning the unit numbers are not unique in the context of the entire dataset. With this organization, they now are, and there is a mapping in the component data table (Table 3) that holds this relationship.

| id | type_id | age | rul | units |
|---|---|---|---|---|
| 1 | 1 | 0.0 | 100.0 | cycles |
| 2 | 1 | 0.0 | 95.0 | cycles |
| ... | | | | |
| 89 | 1 | 0.0 | 59.0 | cycles |
| 90 | 1 | 0.0 | 54.0 | cycles |

Table 4. Asset Table

The asset table is a predefined table in the framework that contains fields necessary to maintain relationships among components and data, as well as fields specific for prognostics.

| Symbol | Description | Units |
|---|---|---|
| id | summary table foreign key | – |
| asset_id | asset table foreign key | – |
| Wf | fuel flow | pps |
| Nf | physical fan speed | rpm |
| Nc | physical core speed | rpm |
| T24 | LPC outlet temperature | °R |
| T30 | HPC outlet temperature | °R |
| T48 | HPT outlet temperature | °R |
| T50 | LPT outlet temperature | °R |
| P15 | bypass-duct pressure | psia |
| P2 | fan inlet pressure | psia |
| P21 | fan outlet pressure | psia |
| P24 | LPC outlet pressure | psia |
| Ps30 | HPC outlet pressure | psia |
| P40 | burner outlet pressure | psia |
| P50 | LPT outlet pressure | psia |

Table 5. System Telemetry Parameters

The flight data from these parameters are stored in the telemetry table.

The telemetry table ( `telemetry_tb` ) contains the *measurement ($x_s$)* data provided in the raw data files as well as two additional fields, `id` and `asset_id`, which are foreign key references to the summary table and asset table, respectively. The fields for the telemetry table are shown in Table 5.

The degradation data is stored in a table named `degradation_tb`. Different combinations of degradation effects are applied to different units based on the datafile they originate from. The degradation parameters ($\theta$) are unobserved and cannot be directly used for RUL prediction, and therefore not used in this work (this is saved for future tasks). With this organization, complex data queries can be easily carried out, the memory footprint of the data is optimized, and it is easier to load data for further data processing tasks, discussed next.

## 4. Preprocessing

To reduce the size of the dataset to improve data handling speeds and reduce training time, the dataset features were reduced in size from their original datatypes of *int64* and *float64* to *int32*, *int16*, and *float32* depending on the value range of the feature. While the full dataset is stored in the database, we downsampled the dataset by a factor of 20 for model development. There are multiple ways to do this that come with their own tradeoffs, we simply kept every 20th data point. This method is much quicker than using a moving average and due to the high frequency of measurements relative to the number of measurements taken during the life of the system, did not result in noticeable information loss.

In the first version of the CMAPSS dataset [Saxena et al., 2008], normalization with respect to the flight settings were typically done by normalizing the sensor data depending on which of the six discrete operating conditions it belonged to. In the N-CMAPSS dataset however, the flight settings are continuous making such a normalization procedure infeasible. This is not to be confused with normalization to a $[0, 1]$, which is done in both cases. Due to this major difference and other improvements, a comparison of approaches applied to the old dataset and new dataset is not feasible, and at the time of this writing only a handful of papers have been published using the new dataset.

We made use of the health state variable ($h_s$) to train a model on the healthy subset of the data that learned the relationship between operating conditions and flight sensor data. The output of this model represents the expected sensor readings, given a flight setting, in its non-degraded state. The residual between the expected and the actual sensor is then interpreted as the degree of deviation from normal, i.e degradation. This procedure reveals the unobservable degradation trend which we argue reduces the prognostics model complexity. Figure 4 shows an example output of this transformation with the original (top) and the transformed (bottom) signal for the T30 sensor.

To achieve this, We used a model with 5 dense layers and ReLU activations, illustrated in figure 3. The Adam optimizer with the initial learning rate of 0.001 and a decay factor of .95 per epoch was used for training. An early stopping was used where the training would stop if the improvement in validation loss was less than .001 over 20 epochs.

Once the data types of the measurements have been converted, downsampled, normalized, and transformed to expose the degradation effects, the data is ready for use in the next step of the model development pipeline. This is hyperparameter optimization, and we implement a modified Bayesian Optimization method, discussed next.

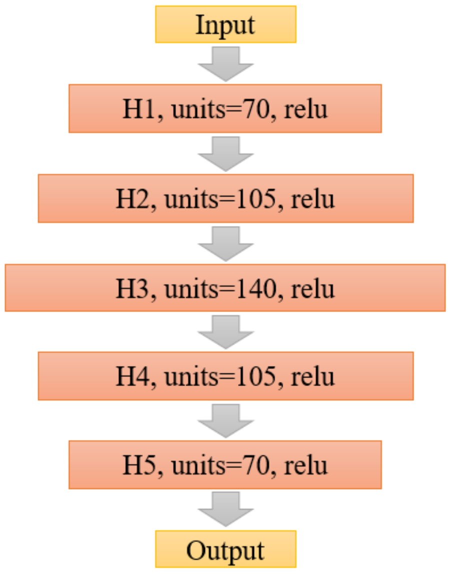

Figure 3. Feedforward Network Architecture for Preprocessing Telemetry Data

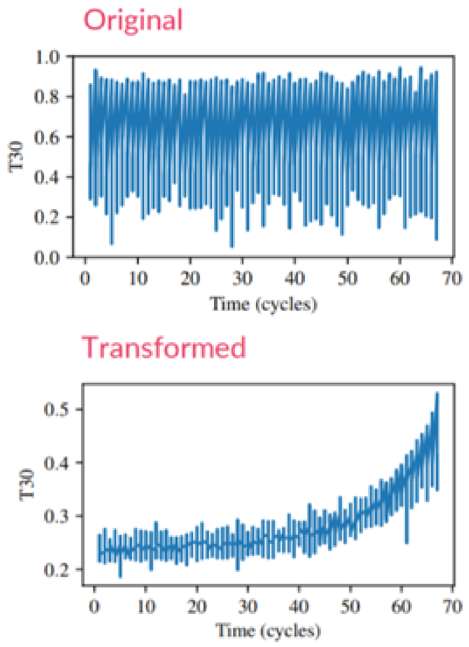

Figure 4. Transformation of Raw Data into Prognosable Signals

## 5. Hyperparameter Search

Proper hyperparameter configuration is essential to developing high performing models. Performing a hyperparameter search is a necessary part of an AutoML model development pipeline as it systematizes what would otherwise be conducted in an ad-hoc manual trial-and-error fashion. We do not want to create big models when smaller models would suffice, or manually select hyperparameter values that lead to sub-optimal configurations. Models tuned with hyperparameter optimization methods have shown to be superior than guess-and-check methods [Bergstra et al., 2011, Feurer and Hutter, 2019]. In our initial experiments, we found that configurations with non-zero regularizing parameters did not even make the list of top 5 best configurations. Figure 5 illustrates why this is the case. Regularization is a method (or set of methods) to improve a model's ability to generalize [Kukačka et al., 2017] and imparts a time-quality tradeoff during training [Goodfellow et al., 2016]. This means that the resultant model will have a smaller loss than its non-regularized counterpart, but will take longer to achieve a lower loss.

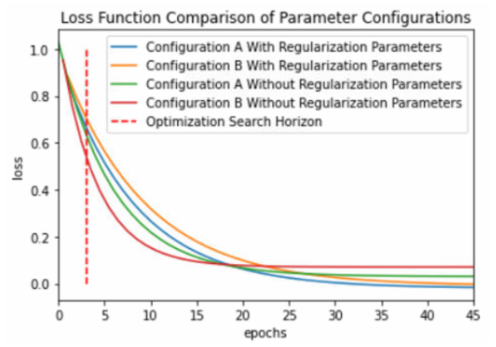

Figure 5. Comparing the effect of regularization on the loss function.

In Figure 5, four loss functions are depicted along with the search horizon ($\hbar$) for the optimization procedure. It is clear that at this stage in training, the network with non-zero regularization parameters is performing the worst. Informed searches such as Bayesian Optimization [bay, ] or Hyperband [Li et al., 2016] would either reduce the probability of improvement or expected improvement around this candidate solution or discard the configuration all together. However, later we see that this is actually the best solution, but only after the 25th epoch is this known. Therefore, we implemented a two-step hyperparameter search whereby the layers, units, and learning rate is searched first. The top $N_c$ candidates are selected from this step, and the search is then repeated with dropout and regularization. The results from this step are shown in Table 6. Initially we chose the top scoring configurations but later realized the results proved to be misleading. Instead of concluding after the search process is complete (by means of evaluating $N_t$ number of trials) that the top scoring networks are the best, a distribution of scores for each configuration should be calculated, and the mean of the distribution should be used to rank configurations. Therefore, a second modification was implemented to ensure that the top scoring configurations were evaluated a minimum number of times before the search procedure halts.

| # | Layers | Units | LR | Score | $N_t$* |
|---|---|---|---|---|---|
| 1 | 4 | 24 | .001 | 2.6 | 1 |
| 2 | 5 | 16 | .001 | 2.81 | 3 |
| ... | | | | | |
| 12 | 3 | 32 | .001 | 3.03 | 16 |
| ... | | | | | |
| 17 | 4 | 32 | .0005 | 0 3.07 | 4 |
| ... | | | | | |
| 21 | 3 | 32 | .0025 | 0 3.1 | 9 |
| ... | | | | | |

Table 6. Initial Hyperparameter Optimization Results

*number of trials evaluated.
Regularization parameters were searched separately after network architecture parameters.

| # | Layers | Units | LR | Score | $N_t$ |
|---|---|---|---|---|---|
| 1 | 3 | 28 | .005 | 3.0 | 9 |
| 2 | 3 | 32 | .0025 | 3.01 | 8 |
| 3 | 3 | 32 | .001 | 3.06 | 29 |
| 4 | 4 | 32 | .001 | 3.09 | 25 |
| 5 | 4 | 32 | .0005 | 3.09 | 40 |

Table 7. Top five configurations from step 1

These are the results of the search over network architecture parameters after ensuring a distribution of scores is captured for each configuration.

The process halts after a minimum number of unique configurations have been evaluated *and* the top $N_c$ configurations have been evaluated at least $min_t$ times each. This produces a distribution of top candidates and can be implemented with any search procedure. Once the top $N_c$ candidates from this step are identified, this process repeats over regularizing parameters (l2 and dropout were used in this work). The results of step 2 of the search process is shown in table 8. The smallest network that could have been chosen based on the range of allowed values is a 3-layer, 16-unit per layer network. The largest is a 6-layer 64-unit network. The final network configuration found was near the small end of this range, suggesting that a less complex network is better.

| # | Layers | Units | LR | Dropout | L2 | Score | $N_t$ |
|---|---|---|---|---|---|---|---|
| 1 | 3 | 32 | .0025 | .5 | $1e^{-4}$ | 2.8 | 25 |
| 2 | 3 | 32 | .001 | .5 | $1e^{-5}$ | 2.83 | 28 |
| 3 | 4 | 32 | .0005 | .2 | $1e^{-4}$ | 2.84 | 21 |

Table 8. Top 3 configurations with dropout and L2 regularization

These are the results after the regularization parameters have been searched over with the top $N_c$ candidates from step 1.

The search parameters for this two-stage process are given in Table 9. Alpha ($\alpha$) governs the expected noise in the observed performance of the underlying surrogate model and beta ($\beta$) governs how far to draw samples from the current best solution. The search horizon ($\hbar$) is the number of epochs to evaluate each configuration and $N_c$ is the number of top-scoring configurations to keep for step 2, which is searching over the regularization parameters with a given set of architecture configurations.

| Parameter | Symbol | Value |
|---|---|---|
| number of trials | $N_t$ | 128 |
| alpha | $\alpha$ | .0001 |
| beta | $\beta$ | 4.8 |
| search horizon | $\hbar$ | 3 |
| number of candidates | $N_c$ | 5 |

Table 9. Bayesian Optimization Parameters

## 6. Training & Validation

The Bi-LSTM network (Figure 6) was trained with two different data ingestion methods. First, we used a fixed sequence length input, the method traditionally used when training models. To generate the training samples, a sliding window approach was utilised with a window size of 100 time steps and a step size of 10. From the 90 trajectories in the data set, 10 units spread across the 8 data files were used for validation. Total training time was 50 epochs. The model hyperparameters are that of the model #1 in Table 8.

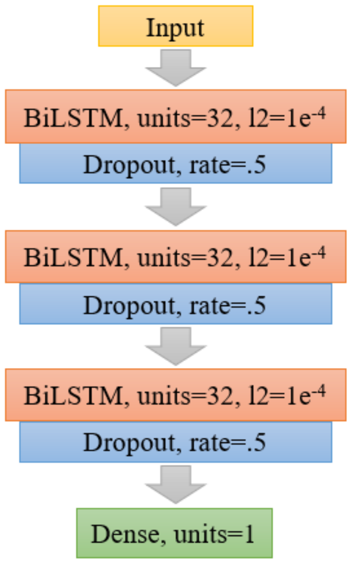
Input
BiLSTM, units=32, l2=1e-4
Dropout, rate=.5
BiLSTM, units=32, l2=1e-4
Dropout, rate=.5
BiLSTM, units=32, l2=1e-4
Dropout, rate=.5
Dense, units=1

Figure 6. Bi-LSTM Network for RUL Estimation

Since recurrent networks are input size agnostic, we also trained the same model configuration with a variable length input data loader. In a variable sequence length data loading procedure, the notion of an epoch is not clear. Therefore, we specified the training time in terms of gradient updates. In total, the network was trained for 10000 gradient steps. For each gradient step, a random sequence length between 4000 and 15000 data points is picked. A minimum sequence length of 4000 was chosen due to it being the shortest trajectory in the 2021 PHM Society Data Challenge and 15000 being the longest trajectory in the training set. Using the selected sequence length, one sequence from each of the engines in the training set was extracted, where the end point of the sequence had to be in the non-healthy state. This is the same training procedure used our previous work and described more thoroughly in [Lövberg, 2021].

To evaluate the performance of the model using both fixed and variable length sequences, we used NASA's scoring function [Saxena et al., 2008], shown in Equation 1. This function penalizes over estimations more than under estimations due to the nature of the intended use. It is better to be conservative with RUL estimates as over estimates can lead a system to operate in an unsafe state.

$$\begin{aligned}
\Delta_j &= \hat{y_j} - y_j, \\
\alpha &= \begin{cases} \frac{-1}{13} & \text{if } \Delta_j < 0 \\ \frac{1}{10} & \text{if } \Delta_j \geq 0 \end{cases} \\
s &= \sum_{j=1}^{m_*} exp(\alpha|\Delta_j|), \\
RMSE &= \sqrt{\frac{1}{m_*}\sum_{j=1}^{m_*}\Delta_j^2} \\
score &= (s + RMSE)/2.0
\end{aligned} \tag{1}$$

Eq 1: Scoring Function Calculation

The end result of the model when developed with this methodology, and evaluated with two different forms of input are discussed in the next section.

## 7. Results & Discussion

The results of both fixed and variable input sequence methods for the trained model are shown in table 10 and Figure 7. These results validate the hypothesis previously proposed in [Lövberg, 2021] that models trained with variable length input sequences perform better than fixed length sequences. With such a small validation set, however, the significance of the result is brought into question. When removing the worst scoring unit from both sets, the improvement of the variable length sequence over the fixed length is more noticeable. Furthermore, the hyperparameter search was conducted with a fixed length input sequence, so it can be proposed that a different set of hyperparameters would have been found if the search was conducted with a variable length input.

| **Unit** | **Score** | |
|---|---|---|
| | **Variable Length** | **Fixed Length** |
| 1-DS01 | 3.29 | 2.89 |
| 4-DS03 | 2.85 | 4.97 |
| 10-DS03 | 1.51 | 4.29 |
| 8-DS04 | 6.7 | 5.27 |
| 10-DS05 | 1.89 | 2.0 |
| 5-DS06 | 4.81 | 2.39 |
| 6-DS07 | 1.98 | 5.88 |
| 4-DS08a | 3.38 | 3.86 |
| 11-DS08a | 1.41 | 2.24 |
| 5-DS08c | 2.71 | 1.80 |
| **mean** | 3.05 | 3.56 |
| **std dev** | 1.56 | 1.41 |

Table 10. Model results for fixed and variable length input sequences

The implementation of a data management scheme within a model development pipeline improves the efficiency of data storage, retrieval, and code reuse. By addressing data management concerns upfront, the rest of the processes can be more easily streamlined, and the process of developing data-driven models becomes a repeatable pattern that can be used regardless of the application. The overall framework offers three primary results specific to data-driven prognostics applications:

- Transforming the raw telemetry data into prognosable signals as shown in Figure 4 is required for a model to learn the underlying degradation trends without observing the actual degradation variables. Oftentimes, the degradation processes are unobservable or unknown, and therefore such information is unavailable for training.
- Ranking hyperparameter configurations based on their score distribution rather than their single-sample score provides a more reliable means of selecting the best configuration for a given task. The tables in Section 5 show that initial sample scores are not reflective of the models actual performance, and only later after several samples have been collected are the top scoring configurations known.
- variable sequence length inputs result in models that perform better than when trained with fixed length sequences. Recurrent neural networks are input length agnostic, and allowing for the sequence length to be randomized impoves the models ability to generalize.

## 8. Conclusion & Future Work

The most important aspect of the model development pipeline is in data management. The data management framework used in previous work for UAV simulation data [Darrah et al., 2021, Darrah et al., ] can generalize across domains and is flexible to suite a variety of use cases. It facilitates data handling with code reuse with a standard API (under active development) and allows data to persist in an organized manner that retains the relationships among different data features. Using this to systematically design, develop, and train models for data-driven prognostics makes these tasks for future work much easier to accomplish. It also improves collaboration among researchers and provides a higher degree of confidence in the results we obtain through our efforts or results we seek to validate.

We implement methods for data preprocessing, hyperparameter optimization, and model training that are specifically tailored to data-driven prognostics. However there is plenty of room for future work. For example, we downsampled the raw data by collecting every $n^{th}$ signal, but aggregating over a window of size $n$ is also a valid downsampling method. The modified hyperparameter search turned Bayesian Optimization into a pseudo-random search by means of random restart. Comparing the efficiency in terms of time, number of trials, number of candidate configurations, and results of this method with a purely Random Search method should be done. Furthermore, the hyperparameter search was carried out with a fixed-length input, but the resultant configuration was used for both fixed-length and variable length. It could be the case that a different set of parameters would be found that would produce a model which results in even lower scores for variable-length inputs. Such tasks will be the focus of research papers to come. Finally, a larger validation set would allow for higher confidence in the comparison results between fixed and variable length input sequences, which shows that variable length input sequences offer better results, but the variance is too high to be conclusive.

## Acknowledgements

This work is supported by NASA OSTEM Fellowship 20-0154 and the Swedish Governmental Agency for Innovation Systems (VINNOVA) within the project ”Applied AI for Prognostics” (Grant Number 2022-00870)”. The Authors are grateful for the support received from these agencies.

## References

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

Chao, M. A., Kulkarni, C., Goebel, K., and Fink, O. (2021). Aircraft engine run-to-failure dataset under real flight conditions for prognostics and diagnostics. *Data, 6, 5.*

Darrah, T., Biswas, G., Frank, J., Quiñones Grueiro, M., and Teubert, C. A data-centric approach to the study ofsystem-level prognostics for cyber physical systems:

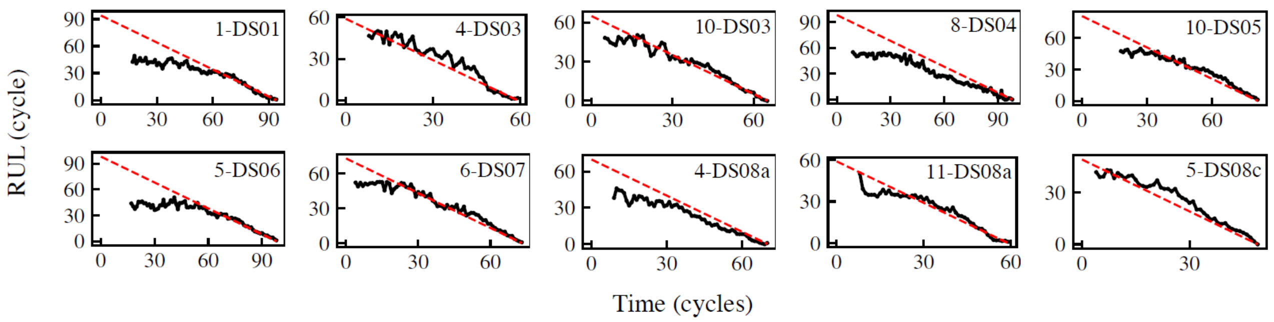

Figure 7. Remaining Useful Life Estimation Test Results

application to safe uav operations.

Darrah, T., Frank, J., Quiñones Grueiro, M., and Biswas, G. (2021). A data management framework & uav simulation testbed for the study of system-level prognostics technologies. In *Annual Conference of the Prognostics and Health Management Society*.

Feurer, M. and Hutter, F. (2019). *Automated Machine Learning, Chapter 1*. Springer.

Goebel, K., Celaya, J., Sankararaman, S., Roychoudhury, I., Daigle, M., and Saxena, A. (2017). *Prognostics: The Science of Making Predictions*.

Goodfellow, I. J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA, USA. `http://www.deeplearningbook.org`.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2018). *Automated Machine Learning - Methods, Systems, Challenges*. Springer.

Kukačka, J., Golkov, V., and Cremers, D. (2017). Regularization for deep learning: A taxonomy.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2016). Hyperband: A novel bandit-based approach to hyperparameter optimization. *CoRR*, abs/1603.06560.

Lövberg, A. (2021). Remaining useful life prediction of aircraft engines with variable length input sequences. In *Annual Conference of the Prognostics and Health Management Society*.

Munappy, A., Bosch, J., Olsson, H. H., Arpteg, A., and Brinne, B. (2019). Data management challenges for deep learning. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*.

Peng, Y., Dong, M., and Zuo, M. J. (2010). Current status of machine prognostics in condition-based maintenance: a review. *The International Journal of Advanced Manufacturing Technology*, 50(1-4):297–313.

Saxena, A., Goebel, K., Simon, D., and Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 International Conference on Prognostics and Health Management*.

Schwabacher, M. and Goebel, K. (2007). A survey of artificial intelligence for prognostics. *Aaai fall symposium (2007)*.

Si, X.-S., Wang, W., Hu, C.-H., and Zhou, D.-H. (2011). Remaining useful life estimation–a review on the statistical data driven approaches. *European Journal of Operational Research, 213(1), (2011): 1-14*.

Strasser, C., Cook, R., Michener, W., and Budden, A. (2012). Primer on data management: What you always wanted to know. *UC Office of the President: California Digital Library*.