Remaining Useful Life Prediction for Turbofan Engines on NASA N-CMAPSS Data

Group 81: Vaibhav Attre (GRU), Calvin Zhang (Random Forest), Alexander Zhang (Linear Regression)

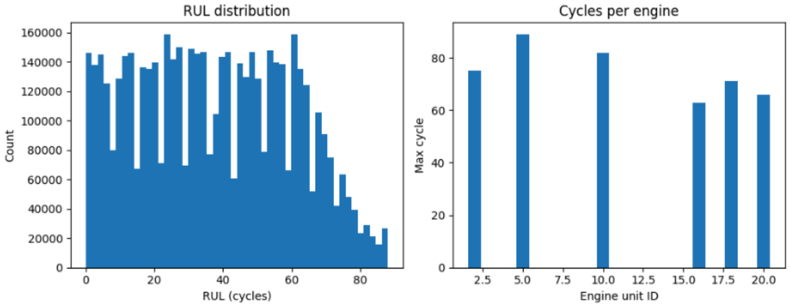
Introduction

Modern aerospace systems are fitted with various sensors, particularly in engines, that generate large volumes of time-series data. These sensors play a crucial role in predictive maintenance that not only reduces the operational costs for customers, but also provides higher degrees of safety for passengers. In our case, the sensors in turbofan engines help forecast a variable called remaining useful life (RUL) which tells how many cycles are left before a component, the engine, fails or needs repair. As we train, we observe the full trajectories of each engine as well as the RUL at each cycle, but during testing, we constrain the inputs to only sensor, and other flight related data to predict RUL of each engine. There are many challenges with this problem such as how interconnected all the sensors are, non-linear relationships between raw sensor readings and RUL, and more.

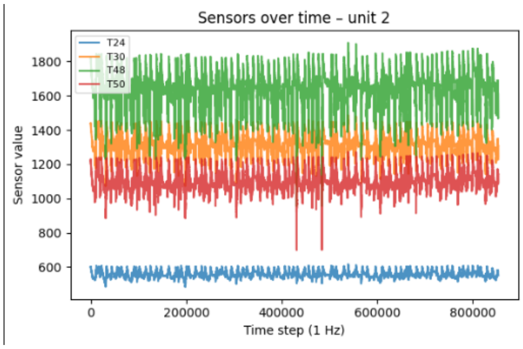
All three models used a similar feature engineering method. For each engine and time step, we construct an input vector using a sliding window that scans across the time-series data and trains a model to output an RUL value for given conditions. We had three main goals in this project: understanding the dataset and its properties using exploratory data analysis, comparing three different models against each other, and studying how various techniques like hyperparameter selection, architecture selection, and regularization can be used to improve the accuracy and generalizability of each model. We hope to identify choices that can be used to predict the RUL on given turbofan engine data and draw broader conclusions surrounding time-series modeling for predictive maintenance.

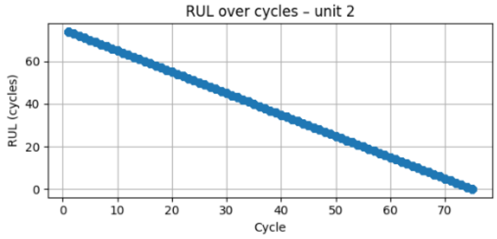
Dataset and Exploratory Analysis

This project used NASA's C-MAPSS to simulate turbofan engine degradation, and the dataset itself can be found on NASA's website. The dataset provides multivariate time-series data sampled at 1 Hz. For each engine unit, the dataset records multiple flight cycles, and within each cycle, sensor and operating conditions are logged once per second from the beginning of the cycle till the end. Thus, each engine is represented by a sequence of cycles, and each cycle consists of many 1 Hz samples. The data recorded includes altitude, mach number, throttle resolver angle, and many sensor measurements for temperature, pressure, and rotational speeds. We use these variables to predict the RUL, measured in cycles, at each time step for the engine. To aid with feature development and training, we constructed a development set that splits the available run to failure trajectories into train, validation, and test subsets.



The figure on the left shows the distribution of RUL labels in the dev set. The RUL values range from 0 to roughly 90 cycles. The values have fairly broad coverage across the interval but there is a smaller amount of data for the largest remaining lifetimes. The figure on the right reports the maximum cycles per engine. These values range from roughly 60 to 90 cycles. This ensures that our model doesn't learn from a constant time to failure but use the sensor data alongside operating conditions to predict how much an engine degrades.





We can further investigate the degradation itself by examining the evolution of a few temperature sensors for a random engine (engine 2 in this case). The left panel shows 4 sensors over full time of the engine. While the signals are noisy, the gradual drift in it hints and is consistent with progressive wear of engines over time. The right panel shows the corresponding RUL for the given engine. We can see that it drops linearly across each cycle, but the combination with sensor trajectories hint that similar RUL values can correspond to varying sensor ranges depending on the engine and its profile. One thing to note, although the RUL labels decrease linearly with cycle index, the mapping from sensor time-series to RUL is not necessarily linear, but rather dependent on history. While predicting, we don't actually know what the current failure cycle is, thus we must think about different models than just a simple linear regression.

These observations, amidst others present in the GitHub, motivate our modeling choices for each model. They suggest that RUL prediction can be treated as a supervised learning problem, and that the models should be able to capture the relationship between our given variables and RUL.

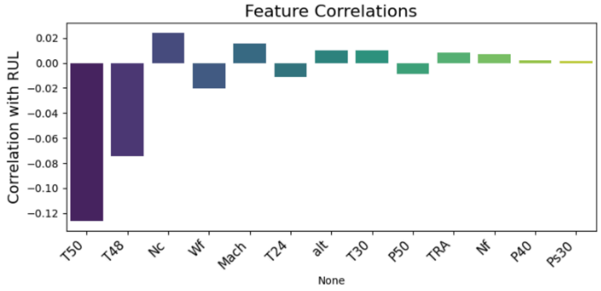
Methods

Problem and Feature Engineering

We treat the RUL prediction as a supervised regression problem at per-cycle level. The dataset provides us: altitude, mach number, throttle resolver angle, inlet temperature, and multiple internal sensors from the engine such as shaft speeds, temperatures, and pressures. There were many other features beside these, but we used a feature engineering technique to reduce this pool to something meaningful. We first began with all possible candidates, and then performed one step for the first two models and an additional step for the GRU model.

1. Correlation with RUL:

Using the full development set, we computed the correlation between each candidate feature and RUL and ranked them to get the correlation magnitude.



2. Random Forest feature importance:

   In the GRU model, we first do step 1 to get all the correlations. After that, we fit a random forest regressor (50 trees, max depth of 8) on a 5% subsample (for computational efficiency) of the data using RUL as the target. Then we extracted the importances of each feature from the model and ranked them. After this process, we are finally ready to take the union of the top k features of both rankings, giving us a feature selection that is strongly correlated with RUL and useful for non-linear models.

All features are then standardized using only the training engines and then this transformation is applied to the validation and test sets. We also split the data by engine such that each engine is assigned to one of the train, validation, or test sets to make sure that the model doesn't see the same engine in both training and testing.

Sliding window sequence

To allow our models to not only use snapshots at a time, but rather the recent history of an engine, we converted each engine's data into sequences using sliding windows over the cycles. For a given window length W, we first sort all rows for that engine by cycle index. Then for each cycle i, we construct a window consisting the previous W cycles, and set the RUL at cycle i as the target label for a given sequence. Since our stride is only one cycle, the neighboring windows overlap, sharing history, allowing our model to predict the RUL better. This procedure is applied to each of the train, validation, and test datasets, producing sets that are fed to each model.

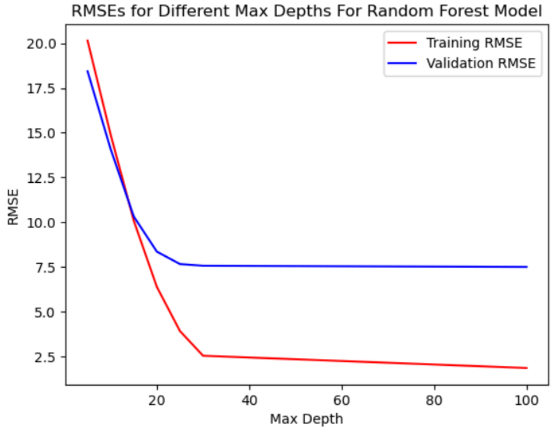
Linear Regression with L2 Regularization

For our baseline model, we used Linear Regression with Ridge (L2) regularization which operates on engineered features extracted from sliding windows of sensor data. The model takes rolling statistics computed over windows of length W=30 cycles for each sensor, including mean, standard deviation, minimum, maximum, and linear slope. These 5 statistics per sensor create a rich feature representation that captures both instantaneous values and short-term trends in the degradation trajectory. The main reason to choose Linear Regression as a baseline is that it provides a simple, interpretable benchmark that trains quickly and helps us understand the linear relationships between sensor features and RUL. By establishing this baseline, we can quantify the gains achieved by more complex models and validate whether added complexity is justified.

The model is trained using standard least-squares optimization with L2 regularization to prevent overfitting, scaled features using StandardScaler, and evaluated using MSE, RMSE, MAE, and NASA's asymmetric scoring function. We performed feature selection by analyzing correlations with RUL, selecting the top 10 most correlated sensors from the 17 available physical measurements (excluding virtual sensors and health parameters per the dataset documentation). The regularization strength was tested across alpha values [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0] using 80/10/10 train/validation/test split. This approach was to accommodate hardware limitations; however, it gives us a stable, interpretable baseline that establishes the performance floor while providing insights into which sensors are most predictive of remaining useful life. The simplicity of the LR model also makes it computationally efficient, training in seconds compared to hours for deep learning approaches.

Random Forest

The second model used was a random forest model with windowing. Window size was again set to 30, with the same features extracted from each window. Data was split into train-validation-tests along a 70-15-15 split. Since six engines were in the dataset, 4 were used for training, 1 for validation and 1 for testing. Validation data was used for hyperparameter tuning by finding the best max depth allowed, minimum samples per leaf, and the number of trees built. The RMSE was used to find the best model each time. Feature selection for the random forest included the same features from the baseline model, but also three other features that ranked top 10 by importance.

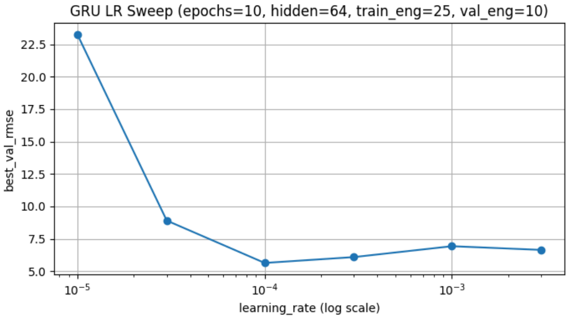


Applied to this problem, random foresting can be a useful technique because it naturally avoids overfitting. Each decision tree only samples a portion of the data, so that enforces diversity among the trees and keeps the complexity of the patterns learned to a limit. This was demonstrated to me when I explored various max_depth and min_samples_leaf values: adding regularization by decreasing the former or increasing the latter only led to a worse validation RMSE.

GRU and Hyperparameter Selection using Optuna

For one of the models, we used a gated recurrent unit (GRU), a network that operates on sliding windows of length W (each window is shaped W x d where d is # features). This sequence is passed through many stacked GRU layers within which it updates a hidden state summarizing the history up to that cycle. By the final window, the hidden state encodes the recent degradation trajectory of the engine, which is passed through a linear layer to predict the

RUL. The main reason to choose GRU is that RUL depends on recent trajectory of sensors, not just instantaneous values, and GRUs can learn these temporal patterns in an efficient way while filtering out noise present in the given data. The GRU is trained using MSE and optimized with Adam on mini-batches of windows.



We also performed hyperparameter tuning using Optuna on a 15% subset of the training windows using 3-fold cross validation. Specifically, we used GroupKFold with engine IDs as the groups. For each Optuna trial, we trained the GRU with the proposed hyperparameters for x epochs on each fold, recorded the best val RMSE per fold, and then calculated a score as the mean of the best RMSE values across the three folds. Optuna then proposed a new hyperparameter config to minimize that RMSE value. The learning-rate sweep shown in the right supports this choice as validation RMSE varies across different learning rates as small lr converge poorly and moderate achieve the lowest RMSE. After selecting the best hyperparameters, we restrained the GRU on the full training data and evaluated on the test data. Overall, this procedure improved our performance (see in table below), demonstrating that choices like learning rate have an impact on generalization.

Experiments & Results

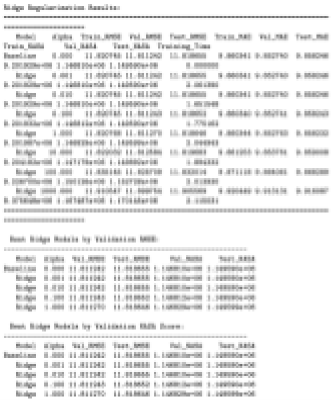
Evaluation

We evaluated all the models on held out test engines using RMSE as the main measure of accuracy since it penalizes large RUL errors. We also printed out MAE and something called the Nasa asymmetric score which penalizes under prediction and over prediction with different weights. After splitting the dataset, hyperparameters are selected based on validation dataset only. After then, we retrain each model on train and validation datasets and then evaluate on test.

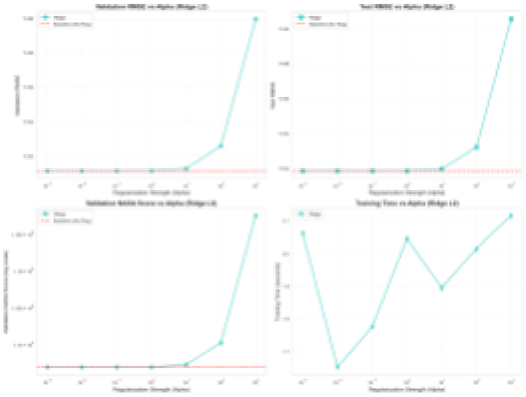
Results

Model	RMSE (Test)	MAE (Test)	NASA (Test)
Baseline	11.8187	9.8582	1,149,589.510
Random Forest	10.5896	9.0045	2,257,945.929
GNU RNN (Optuna)	5.1864	3.9203	1,205,804.614
GRU RNN (Base)	8.7505	6.76	1,597,534

LR with L2 Regularization Results

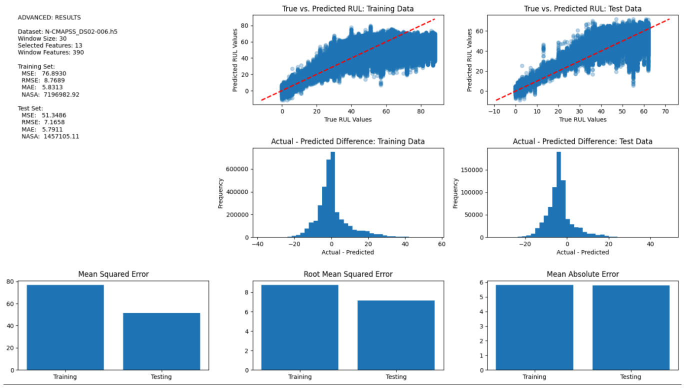


```
Ridge Regularization Results:
====================================================================================================
====================
    Model    Alpha  Train_RMSE  Val_RMSE  Test_RMSE  Train_MAE  Val_MAE  Test_MAE
Train_NASA     Val_NASA    Test_NASA  Training_Time
Baseline    0.000  [illegible]
[illegible]
   Ridge    0.001  [illegible]
[illegible]
   Ridge    0.010  [illegible]
[illegible]
   Ridge    0.100  [illegible]
[illegible]
   Ridge    1.000  [illegible]
[illegible]
   Ridge   10.000  [illegible]
[illegible]
   Ridge  100.000  [illegible]
[illegible]
   Ridge 1000.000  [illegible]
[illegible]
====================================================================================================
====================

 Best Ridge Models by Validation RMSE:
--------------------------------------------------------------------------------
   Model  Alpha  Val_RMSE  Test_RMSE    Val_NASA    Test_NASA
Baseline  0.000  [illegible]
   Ridge  0.001  [illegible]
   Ridge  0.010  [illegible]
   Ridge  0.100  [illegible]
   Ridge  1.000  [illegible]

 Best Ridge Models by Validation NASA Score:
--------------------------------------------------------------------------------
   Model  Alpha  Val_RMSE  Test_RMSE    Val_NASA    Test_NASA
Baseline  0.000  [illegible]
   Ridge  0.001  [illegible]
   Ridge  0.010  [illegible]
   Ridge  0.100  [illegible]
   Ridge  1.000  [illegible]
```



As you can see across the graphs, GRU model achieves the lowest RMSE and MAE on the test engines, indicating that modeling short-term degradation using sliding windows is an effective architectural decision. Moreover, the gap between GRU and the random forest is significant and is consistent with expectations as GRU tends to deliver the best RMSE values for test sets.

We can also see the importance of different tactics on the error rates. In the linear regressor, we can see that with L2 regularization we can see that regularization does not meaningfully improve performance as the validation/test RMSE stays flat for small to moderate alpha values. As alpha becomes large, the model over-regularized, leading to a quick degradation in metrics like the NASA score due to aggressively shrinking coefficients. These results reinforce that the main limitation here is model capacity, rather than the lack of hyperparameter selection. This motivates using other models such as a GRU or a random forest that better capture nonlinear, time-dependent patterns that other models might struggle with.



```
ADVANCED: RESULTS

Dataset: N-CMAPSS_DS02-006.h5
Window Size: 30
Selected Features: 13
Window Features: 390

Training Set:
  MSE:   76.8930
  RMSE:  8.7689
  MAE:   5.8313
  NASA:  7196982.92

Test Set:
  MSE:   51.3486
  RMSE:  7.1658
  MAE:   5.7911
  NASA:  1457105.11
```

GRU (with Optuna)

Finally, within the GRU, hyperparameter selection clearly matters a lot. The Optuna-Tuned GRU consistently improves over the baseline GRU (trained with fixed defaults), giving us lower RMSE and MAE, suggesting that parameters like learning rate and hidden size, amongst others, significantly affect model generalization to unseen data.

Conclusion

Overall, the GRU achieved the best test performance, indicating that using sliding windows to model recent history is a great fit to predict time-series data such as RUL. Random forest was also competitive and outperformed linear regression in all cases, showing that it could be a computationally efficient alternative to the GRU. Going forward, we would evaluate each model on much larger N-CMAPSS datasets to test how the models generalize over different operating conditions. Furthermore, exploring window length parameter testing more carefully would be something to look carefully into as it has a direct impact on RUL prediction. Considering GRU performed great on this dataset, we would also like to see how an LTSM would perform as, while more computationally expensive, tends to perform even better than the GRU.