Transformer-based Predictive Maintenance of Degrading Turbofan Engines

Master's thesis submitted for the degree of:

Master of Science program Civil Engineering.

Department of Civil, Geo and Environmental Engineering.
Technische Universität München.

Author: Jitendra Tiwari

Mat. Num. 03750161

Heiglhofstraße 66/313, 81377 München

jitendra.tiwari@tum.de

Supervisors: M.Sc. Daniel Koutas

Univ.-Prof. Dr. Daniel Straub

Professorship of Engineering Risk Analysis

Date of issue: September 30, 2024

Date of submission: October 15, 2024

Technische Universität München

Involved Organizations

Engineering Risk Analysis Group
Department of Civil, Geo and Environmental Engineering
Technische Universität München
Arcisstraße 21
80333 München

Declaration

With this statement I declare that I have independently completed this Master's thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

München, October 15, 2024

Jitendra Tiwari

Jitendra Tiwari
Heiglhofstraße 66/313, 81377 München

Abstract

A unique approach to predictive maintenance (PdM) and prognostic health management (PHM) is presented in this thesis. As the increasing importance of predicting component and system failures in various industries has been recognized, data-driven prognostic algorithms have been developed. However, it has been shown that the evaluation of these algorithms' practical impact on maintenance choices is inadequate.

To evaluate data-driven prognostic algorithms according to their impact on predictive maintenance choices, a new metric is introduced. This metric emphasizes component procurement and replacement strategy scenarios and is linked to certain decision contexts and PdM regulations. By using run-to-failure data to predict maintenance costs per unit time over the long term, existing PDM tactics are improved.

The Transformer architecture is used to forecast the Remaining Useful Life (RUL) of components and analyze multivariate time series data. It is widely acknowledged for its exceptional effectiveness in natural language processing and computer vision. Data from the C-MAPSS aviation engine simulator is used to determine the best course of maintenance and prediction for turbofan engines.

First, a theoretical example is used to investigate the behavior of the suggested metric. Next, three prognostic techniques are evaluated in a simulated degradation scenario of turbofan engines. An analysis of the prognostic algorithm and PdM policy interaction on the measure yields a decision-making-focused performance comparison.

The implications of this study go beyond turbofan engines because other industries where predictive maintenance is crucial can also use the recommended metric and methodology. Subsequent research ought to focus on expanding this approach to different parts and systems, as this could lead to significant improvements in maintenance practices in a variety of industries.

Acknowledgements

I would like to thank **Prof. Dr. Daniel Straub** and **MSc. Daniel Koutas** for granting me the honor of being a member of the Engineering Risk Analysis group(ERA) and for their invaluable help and guidance. I would like to express my deep sense of gratitude to **Daniel Koutas**, **Rohan Raj Das** for their important help and technical suggestions. I am grateful to **Daniel Koutas** for his constant unconditional support and valuable feedback. I want to thank all members of ERA group who have directly or indirectly contributed to my thesis work. Finally I would like to thank my family and my friends for their constant moral support.

Table of Contents

List of Figures

List of Tables

1. Introduction

A unique approach to predictive maintenance (PdM) and prognostic health management (PHM) is presented in this thesis. As the increasing importance of predicting component and system failures in various industries has been recognized, data-driven prognostic algorithms have been developed. There is a discernible deficiency in evaluating the pragmatic influence of these algorithms on maintenance choices. A new metric is introduced to evaluate data-driven prognostic algorithms according to their impact on predictive maintenance choices. This statistic emphasizes component procurement and replacement strategy scenarios and is linked to certain decision frameworks and PdM rules. Using run-to-failure data to project long-term expected maintenance costs per unit of time improves current PdM rules. The transformer architecture predicts components' Remaining Useful Life (RUL) and analyzes multivariate time series data. It is well-known for its exceptional effectiveness in natural language processing and computer vision. The C-MAPSS aviation engine simulator generates data to analyze turbofan engine maintenance and prediction. This study intends to demonstrate the use of the proposed metric and technique in several industries where predictive maintenance is critical. This could lead to significant improvements in maintenance plans in various industries.

1.1. Motivation

This study introduces a metric based on the influence of data-driven prognostic models on downstream predictive maintenance (PdM) decisions (Kamariotis et al. 2024a) for evaluating their effectiveness. It is highlighted how important PdM rules were in creating this statistic, which can be used in any setting where decisions are made. We look at two specific PdM decision frameworks common in industrial settings: (i) component replacement planning and (ii) component ordering-replacement planning. Using Remaining Useful Life (RUL) predictions, various PdM policies from the literature are examined. The goal is to find actions that maximize component longevity or minimize spare inventory while minimizing failure and late order risks. Alternatives and improvements to these policies are proposed, and their effectiveness is assessed using a run-to-failure dataset to calculate the long-term expected maintenance cost per unit of time. This estimate technique is used to evaluate the effectiveness of the proposed metric.

1.2. Objectives

Transformers have received a lot of interest in the time series domain due to their remarkable performance in computer vision and natural language processing tasks. Their ability to capture long-range connections and interactions validates their suitability for time series modeling and drives developments in various applications. A transformer model was created to predict the remaining useful life (RUL) using already-existing datasets. The outcomes of the prognostic model will then guide the best maintenance planning choices.

1.3. Outline

The structure of the dissertation is as follows: A literature review on predictive maintenance policies for lifetime prediction of components is introduced in Chapter 2. The predictive maintenance and decision metric to evaluate those policies is introduced in Chapter 3, along with an explanation of its data-driven estimation using run-to-failure trial samples. The chapter clarifies certain PdM regulations by placing the statistic inside two common PdM decision frames. A thorough explanation of the Transformer model architecture is provided. Chapter 4 outlines the approach, including steps for prepping the data, organizing the data, and training and optimizing the model's hyperparameters. In Chapter 5, we perform numerical evaluations of degrading turbofan engines using the well-known CMAPSS prognostic dataset (Saxena et al. 2008). Three classifiers, one deep learning model, and three data-driven prognostic algorithms have been used and assessed using the suggested decision-oriented metric. It also looks at the connection between the PdM policy on the metric and the RUL prediction algorithm. The research is discussed and concluded in Chapter 5.

2. Literature Review

The main goal of prognostics is to guess when a system or part will stop working properly (Kim, An, and Choi 2017). For online monitoring, this means using a predictive model to guess the Remaining Useful Life (RUL) based on feedback from the system (Galar et al. 2021). The process of using this predictive information to make maintenance planning better is called health management, and it is also known as predictive Health Management (PHM) (Luque and Straub 2019). A stochastic method is often used for RUL estimation because there are a lot of unknowns in the prognostics process (Fink et al. 2020). Because of this, planning upkeep turns into a problem of making sequential decisions when there is uncertainty (Arcieri et al. 2023).

You can tell the difference between model-based and data-driven techniques to prognostication (Kim, An, and Choi 2017). A new study (Lei et al. 2018) divided these approaches into four groups: models based on physics, models based on statistics, approaches using artificial intelligence (AI) (Li, Zhang, and Ding 2019) and mixed approaches. Data-driven AI techniques are becoming more and more common, especially when physics- or statistics-based methods fail to easily model degradation patterns(Nectoux et al. 2012). One big problem with machine learning (ML) models, though, is that it's hard to measure how uncertain their results are (Nguyen, Medjaher, and Gogu 2022). It is very important to accurately record this uncertainty when planning upkeep.

The Prognostics and Health Management (PHM) field has created a wealth of datasets from real and simulated component failures. These datasets track the decline of various parts like bearings (Nectoux et al. 2012), batteries (Saha 2007), and engines (Saxena et al. 2008) and industrial machines (Purohit et al. 2019) over time. Many are publicly available through NASA (Saxena and Goebel 2008), spurring the development of numerous data-driven prediction algorithms. This rich resource allows researchers to create and test new methods for predicting equipment failures. Most existing research concentrates on predicting Remaining Useful Life (RUL) (Chao et al. 2022) and often overlooks the follow-up health management tasks.

In health management based on RUL predictions, the predictive maintenance (PDM) paradigm is prominent (Fink 2020). PDM typically involves scheduling intermittent inspections and maintenance (Kim, Choi, and Kim 2022) or planning actions based on continuous monitoring (Pater, Reijns, and Mitici 2022). PDM can be categorized as model-based or data-driven (Nguyen and Medjaher 2019). Model-based PDM relies on physics-based models, like the Paris-Erdogan law for fatigue crack growth (Paris and Erdogan 1997), or statistical models such as the Gamma or Wiener processes to describe degradation. The effectiveness of model-based PDM (Kamariotis, Chatzi, and Straub 2022) depends on the chosen model. Most studies focus on model-based PDM or use simplified prognostic models for maintenance decision optimization (Benaggoune et al. 2020). End-to-end data-driven PDM frameworks, integrating prognostics and PDM planning, have emerged and been applied to prognostic datasets (Zhuang, Xu, and Wang 2023). These

frameworks depend on ample monitoring data from run-to-failure experiments to train prognostic algorithms and evaluate PDM policies.

3. Predictive Maintenance Decision Policies

3.1. Predictive Maintenance Decision Policies

The efficacy of several prognostic algorithms has been examined in subsequent PdM decision-making in order to assess the quality of these algorithms that produce RUL forecasts. Policies are presented in order to assess the RUL-based decisions (Jensen and Nielsen 2007). A policy is a rule that, at time t, decides what to do depending on the information that is now accessible, such as historical monitoring data and completed actions. A policy might respond, for instance, yes, no to the query, "Preventively replace the component?" In this work, we exclusively take into account decision circumstances where the policy remains constant throughout; we designate this stationary policy (Kochenderfer 2015) as the PdM policy. As illustrated in Fig.1 represents a flow chart from collecting sensor information to processing the data and extracting the features from processed data using the trained model via test data and predicting the probabilities of RUL and using these probabilities in maintenance decisions.

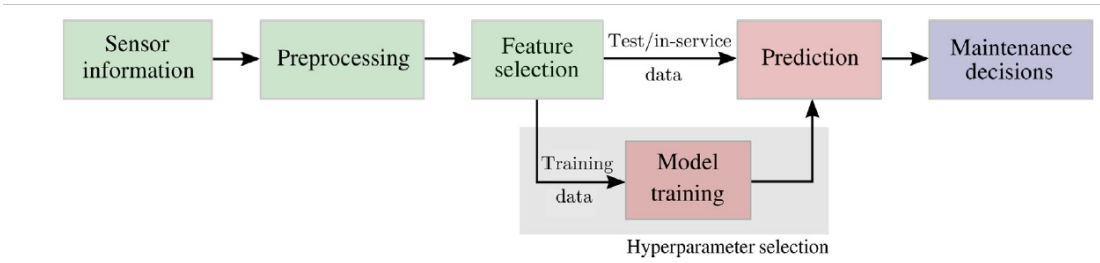

Figure 1 An overview of the suggested framework based on the suggested metric M for a decision-oriented performance evaluation and data-driven prognostic algorithm optimization(Kamariotis et al. 2024a)

3.1.1. Data-based evaluation of the perfect PDM policy

We consider a predictive maintenance policy's long-term cost effectiveness while evaluating it(Tijms 2004). We compute this as the average cost per unit time over a long period of time using ideas from renewal theory. The projected maintenance cycle cost divided by the expected cycle time represents this as a straightforward ratio.

$$R = \frac{\mathrm{E}[C_{\mathrm{m}}]}{\mathrm{E}[T_{\mathrm{lc}}]} \tag{3.1}$$

This formula compares the expected maintenance cost$\mathrm{E}[C_{\mathrm{m}}]$ to the expected lifecycle$\mathrm{E}[T_{\mathrm{lc}}]$ of a component under a specific predictive maintenance policy. It's typically evaluated using

simulations, but with real-world data from multiple failure tests, we can calculate it directly. We apply the policy to n identical independent components, record their maintenance costs and lifetimes, and then estimate the long-term cost efficiency. This approach bridges theoretical models with practical, data-driven assessments of maintenance strategies. The expectations in the numerator and denominator of Eq.3.2 can then be approximated as:

$$\frac{\mathrm{E}\left[C_{\mathrm{m}}\right]}{\mathrm{E}\left[T_{\mathrm{lc}}\right]} \approx \hat{R} = \frac{\frac{1}{n}\sum_{i=1}^{n} C_{\mathrm{m}}^{(i)}}{\frac{1}{n}\sum_{i=1}^{n} T_{\mathrm{lc}}^{(i)}} \tag{3.2}$$

where $\hat{R}$ denotes the data-based estimator of the quantity in Eq.3.2, $C_{\mathrm{m}}^{(i)}$ and $T_{\mathrm{lc}}^{(i)}$ are the cost of maintenance and the lifetime of the ith component, respectively. The lifetime $T_{\mathrm{lc}}^{(i)}$ is the time to failure or replacement of the component.

3.1.2. Data-based evaluation of the perfect PDM policy

To set a benchmark, we imagine a scenario with flawless predictions – where we know exactly when each component will fail. This ideal situation would lead to perfect maintenance decisions. Using data from our real-world tests, we can estimate the long-term cost efficiency of this perfect policy. This estimate serves as our gold standard, showing us the best possible performance to aim for in predictive maintenance. Based on the n run-to-failure experiments, the long-run expected maintenance cost per unit time of the perfect PDM policy can be estimated via Eq.3.3

$$R_{\text{perfect}} = \frac{\mathrm{E}\left[C_{\mathrm{m,\ perfect}}\right]}{\mathrm{E}\left[T_{\mathrm{lc,\ perfect}}\right]} \approx \hat{R}_{\mathrm{perfect}} = \frac{\frac{1}{n}\sum_{i=1}^{n} C_{\mathrm{m,\ perfect}}^{(i)}}{\frac{1}{n}\sum_{i=1}^{n} T_{\mathrm{lc,\ perfect}}^{(i)}} \tag{3.3}$$

where $C_{\mathrm{m,\ perfect}}^{(i)}$ and $T_{\mathrm{lc,\ perfect}}^{(i)}$ are the optimal cost of maintenance and length of the first life-cycle of the ith component, respectively.

3.1.3. Decision Metric to Evaluate the Predictive Maintenance Policy

To define the decision-oriented metric for assessing prognostic algorithms, we calculate the long-run expected maintenance cost per unit of time achieved by combining a specific prognostic algorithm with a PDM policy. This is then compared to the cost resulting from perfect prognostics. We introduce a scalar metric, M, representing the relative difference between the two, which allows us to evaluate the performance of the predictive maintenance algorithm.

$$M = \frac{\frac{\mathrm{E}[C_{\mathrm{m}}]}{\mathrm{E}[T_{\mathrm{lc}}]} - \frac{\mathrm{E}\left[C_{\mathrm{m,\ perfect}}\right]}{\mathrm{E}\left[T_{\mathrm{lc,\ perfect}}\right]}}{\frac{\mathrm{E}\left[C_{\mathrm{m,\ perfect}}\right]}{\mathrm{E}\left[T_{\mathrm{lc,\ perfect}}\right]}} \tag{3.4}$$

Based on the n run-to-failure experiments, the metric M can be estimated as:

$$\hat{M} = \frac{\hat{R} - \hat{R}_{\text{perfect}}}{\hat{R}_{\text{perfect}}} \tag{3.5}$$

An M value of 0 indicates optimal performance; the higher the M value, the poorer the performance of the prognostic algorithm. The metric M cannot be negative. Broadly speaking, M is conceptually linked to the Value of Information (VoI) metric from Bayesian decision theory.
The estimate of the metric M in Eq.3.5 carries some uncertainty. Assuming the variance of $\hat{R}_{\text{perfect}}$ is negligible, the variance of $\hat{M}$ can be expressed as:

$$\mathrm{Var}[\hat{M}] = \mathrm{Var}\left[\frac{\hat{R} - \hat{R}_{\text{perfect}}}{\hat{R}_{\text{perfect}}}\right] = \mathrm{Var}\left[\frac{\hat{R}}{\hat{R}_{\text{perfect}}} - 1\right] \approx \frac{1}{\hat{R}^2_{\text{perfect}}} \cdot \mathrm{Var}[\hat{R}] \tag{3.6}$$

Due to the assumption that the variance of $\hat{R}_{\text{perfect}}$ is negligible, the covariance of $\hat{R}$ and $\hat{R}_{\text{perfect}}$ is neglected.

The decision-oriented metric M can also be used as an objective function for optimizing PDM policies and refining the training of prognostic algorithms, with a direct focus on improving subsequent PDM decision-making (see Fig.1). This result holds without discounting. With discounting, renewal theory allows evaluating expectations for a PdM policy by applying it across n independent single life cycles or n identical components.

3.2. PDM decision settings

This thesis examines two common PDM decision scenarios for industrial assets. In the first, more basic scenario (Section 2.3), the sole decision revolves around when to replace a component. The second scenario (Section 2.4) adds a layer by including the decision of when to order and store a replacement component. Both scenarios share the following key characteristics:

- The analysis is focused on individual component issues.
- Real-time monitoring data for each component is accessible.
- This data is used to generate a probabilistic estimate of the Remaining Useful Life (RUL).
- Routine inspections are not considered in the study.
- Maintenance decisions are made at discrete time intervals, denoted as $t_k = k \cdot \Delta T$, where ΔT is a fixed interval, and k is an integer ranging from 1 to N. The selection of ΔT depends on the context. For instance, in the case of aero-engines, a typical interval of $\Delta T = 5 - 10$ flight cycles is considered practical, based on expert insights.
- Two types of replacement actions are considered:
  1. **Preventive replacement**, which incurs a cost of c_p.
  2. **Corrective replacement**, caused when a part breaks down before a preventative replacement is done. Due to the longer downtime and the need for repair, the cost c_c is higher in this case: $c_c > c_p$.
- A replacement is considered perfect, restoring the component to its original condition. Replacing at time t marks the end of one life cycle, after which the component begins deteriorating again, following the same probabilistic process as at time 0. These assumptions enable the application of renewal theory, where the time between two consecutive replacements defines a renewal cycle.
- Maintenance can be performed within the decision horizon.
- Failures are assumed to be self-announcing.
- Future costs are not discounted.

3.3. Predictive maintenance (PdM) planning for replacement

We start with a straightforward dynamic PdM decision process, where at each time step t_k, the choice is made whether to perform a preventive replacement. It is assumed that a new component is immediately available for preventive or corrective replacement.

3.3.1. Prognostic Performance Metric for PdM Planning focused on Component Replacement

According to the PdM policy (detailed in Sections 2.3.2–2.3.4), each component's lifecycle ends either with a preventive replacement at $T_{\mathrm{R}}^{(i)}$ or a corrective replacement upon failure at $T_{\mathrm{F}}^{(i)}$. Preventive replacements are only allowed at specific time intervals ($T_{\mathrm{R}}^{(i)}$ lies within $\{t_k = k \cdot \Delta T, k = 1, 2, \dots\}$, while corrective replacements occur immediately upon failure at $T_{\mathrm{F}}^{(i)}$. The non-discounted cost of the ith component for the replacement action is:

$$C_{\mathrm{rep}}^{(i)} = \begin{cases} c_{\mathrm{p}}, & \text{if } T_{\mathrm{R}}^{(i)} < T_{\mathrm{F}}^{(i)} \\ c_{\mathrm{c}}, & \text{else} \end{cases} \tag{3.7}$$

Whether preventive or corrective, replacement marks the end of a component's life cycle. The duration of the ith component's life cycle is given by $\min\left[T_{\mathrm{lc}}^{(i)}, T_{\mathrm{F}}^{(i)}\right]$. In this context, the maintenance cost for the ith component equals the replacement cost, i.e., $C_{\mathrm{m}}^{(i)} = C_{\mathrm{rep}}^{(i)}$.

In this decision setting, the perfect PdM policy avoids corrective replacements, which are more expensive, and early preventive replacements, which reduce the component's life cycle. Instead, it performs a preventive replacement at cost $C_{\mathrm{m,perfect}}^{(i)} = c_p$ at the optimal time step $t_k = k\Delta T$ just before $T_{\mathrm{F}}^{(i)}$, denoted as $T_{\mathrm{R,perfect}}^{(i)}$. The long-run expected maintenance cost per unit time for this Perfect PDM policy is then calculated via Eq.3.3

The decision-oriented metric introduced in Eq.3.4, especially when applied with a PdM replacement policy, is estimated as:

$$\hat{M} = \frac{\frac{1}{n}\sum_{i=1}^{n} C_{\mathrm{rep}}^{(i)}}{\frac{1}{n}\sum_{i=1}^{n} T_{\mathrm{lc}}^{(i)}} - \frac{c_{\mathrm{p}}}{\frac{1}{n}\sum_{i=1}^{n} T_{\mathrm{R,\,perfect}}^{(i)}} \Bigg/ \frac{c_{\mathrm{p}}}{\frac{1}{n}\sum_{i=1}^{n} T_{\mathrm{R,\,perfect}}^{(i)}} \tag{3.8}$$

3.3.2. PDM policy 1: simple heuristic PdM policy for preventive replacement

The first PdM policy is a simple heuristic approach. Heuristic policies use intuitive decision rules that engineers and operators can follow easily. At each time step $t_k = k \cdot \Delta T$, the policy determines the action $a_{\mathrm{rep},k}$ as follows:

$$a_{\mathrm{rep},k} = \begin{cases} \mathrm{DN}, & \text{if } \Pr\left(RUL_{\text{pred},k} \leq \Delta T\right) < p_{\text{thres}}^{\mathrm{rep}} \\ \mathrm{PR} & \text{else} \end{cases} \tag{3.9}$$

In this policy, DN stands for "do nothing," PR indicates "preventive replacement," and $RUL_{\text{pred},k}$ is the predicted remaining useful life (RUL) estimated at discrete time step t_k using the prognostic algorithm. In [Galar et al. 2021] variable threshold, $p_{\text{thres}}^{\mathrm{rep}}$, has been set to $p_{\text{thres}}^{\mathrm{rep}} = c_p/c_c$. The PR action incurs a cost of c_p, while the DN action carries the predicted risk of component failure within the next time step, $\Pr\left(RUL_{\text{pred},k} \leq \Delta T\right) \cdot c_{\mathrm{c}}$. The rationale behind setting $p_{\text{thres}}^{\mathrm{rep}} = c_p/c_c$ is that preventive replacement is performed at time t_k only when its cost is less than the predicted risk of failure. However, this is a simplified approach, as it doesn't consider future steps where a new component would have lower average maintenance costs, making the $p_{\text{thres}}^{\mathrm{rep}} = c_p/c_c$ choice suboptimal.

The simple heuristic PdM policy relies solely on the predicted probability of RUL exceedance within the next decision step, $\Pr\left(RUL_{\text{pred},k} \leq \Delta T\right)$, to make the DN vs. PR decision at each discrete time step. Different prognostic algorithms compute this probability using various methods. For prognostic classifiers, it is derived as the probability of $RUL_{\text{pred},k}$ falling into a specific class (i.e., $RUL_{\text{pred},k} \leq \Delta T$). In the case of prognostic regression models (Galar et al. 2021), uncertainty quantification in RUL predictions is needed to calculate this probability. Eventually, this heuristic PdM policy informs the replacement decision for each ith component, leading to $C_{\mathrm{rep}}^{(i)}$ and $T_{\mathrm{lc}}^{(i)}$.

3.3.3. PDM policy 2: simple heuristic PdM policy for preventive replacement based on the full RUL distribution

We are introducing a second predictive maintenance policy for preventive replacement. This policy decides at each time step t_k whether to replace a component or not, the action $a_{\mathrm{rep},k}$ taken as.

$$a_{\mathrm{rep},k} = \begin{cases} \mathrm{PR}, & \text{if } t_k + \Delta T \geq T_{\mathrm{R},k}^{*} \\ \mathrm{DN} & \text{else} \end{cases} \tag{3.10}$$

This policy determines the optimal replacement time $T^*_{\mathrm{R},k}$ at each time steps t_k by solving an optimization problem. It decides to replace the component preventively if $T^*_{\mathrm{R},k}$ falls within the current decision interval (i.e., before or at $t_k + \Delta T$, where ΔT is the time interval until the next decision).

The most common objective function for determining the optimal replacement time $T^*_{\mathrm{R},k}$ in RUL-based predictive maintenance (Zeng and Liang 2022). It works when we have a full probability distribution of the predicted Remaining Useful Life (RUL) at each time steps t_k. We use $f_{RUL_{pred,k}}(t)$ to represent this RUL distribution. From this, we can derive the predicted time to failure distribution, $f_{TF_{pred,k}}(t)$, which starts from the current time t_k is $f_{TF_{pred,k}}(t) = f_{TF_{pred,k}}(t - t_k)$ The goal is to minimize an objective function at discrete time steps t_k. This function balances the costs of early replacement against the risks of late replacement, helping to determine the most cost-effective time for maintenance.

$$f\left(T_{\mathrm{R},k}\right) = \frac{\mathrm{E}\left[C_{\mathrm{rep}}\left(T_{\mathrm{R},k}\right)\right]}{\mathrm{E}\left[T_{\mathrm{lc}}\left(T_{\mathrm{R},k}\right)\right]} = \frac{P_{\mathrm{PR}} \cdot c_{\mathrm{p}} + \left(1 - P_{\mathrm{PR}}\right) \cdot c_{\mathrm{c}}}{P_{\mathrm{PR}} \cdot \left(T_{\mathrm{R},k}\right) + \int_t^{T_{\mathrm{R},k}} t f_{RUL_{\mathrm{Pred},k}}\left(t - t_k\right) \mathrm{d}t} \tag{3.11}$$

where:

$$P_{\mathrm{PR}} = \int_{T_{\mathrm{R},k}}^{\infty} f_{RUL_{\mathrm{Pred},k}}\left(t - t_k\right) \mathrm{d}t \tag{3.12}$$

3.3.4. PDM policy 3: Enhanced predictive maintenance strategy for preventive replacement based on full RUL distribution

We propose an improved objective function defined in Eq.**??** that eliminates the assumption of uniform lifetime distributions for future components. Instead, we estimate the long-term average maintenance cost per unit time for all future components $\frac{\mathrm{E}_{\bar{T}_{\mathrm{F}}}[C_{\mathrm{rep}}]}{\mathrm{E}_{\bar{T}_{\mathrm{F}}}[T_{l\mathrm{c}}]}$ and utilized it to develop an objective function to determine the optimal replacement time for the current component. This function balances two factors:

1. The maintenance cost of the current component
2. The opportunity cost of early replacement

The opportunity cost is calculated as the expected remaining lifetime beyond the replacement time, multiplied by the ratio of average replacement cost to average component lifetime. This approach optimizes the timing of component replacement $T^*_{\mathrm{R},k}$ at each time step.

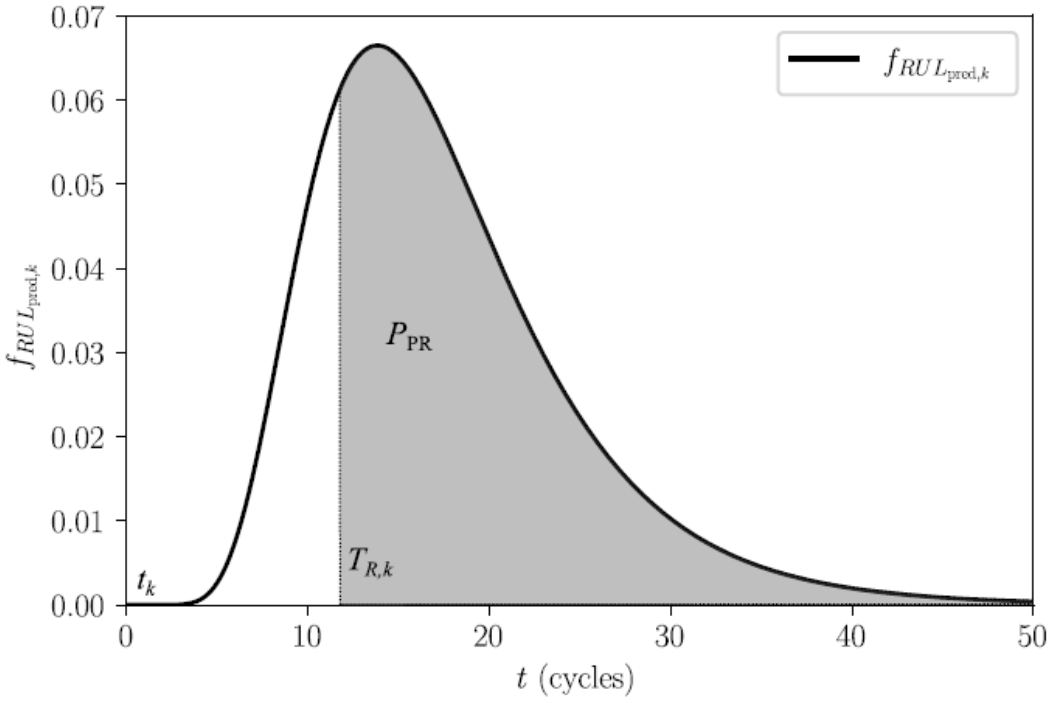

Figure 2 Optimization of $T_{\mathrm{R},k} \cdot P_{\mathrm{PR}}$ corresponds to the probability of a preventive replacement for a fixed $T_{\mathrm{R},k}$, which appear in the objective function of Eq. (Kamariotis et al. 2024b)

$$f\left(T_{\mathrm{R},k}\right) = P_{\mathrm{PR}} \cdot c_{\mathrm{P}} + \left(1 - P_{\mathrm{PR}}\right) \cdot c_{\mathrm{C}} + \int_{T_{\mathrm{R},k}}^{\infty} \left(t - T_{\mathrm{R},k}\right) \cdot \frac{\mathrm{E}_{\bar{T}_{\mathrm{F}}}\left[C_{\mathrm{rep}}\right]}{\mathrm{E}_{\bar{T}_{\mathrm{F}}}\left[T_{lc}\right]} f_{RUL_{\mathrm{pred},k}}\left(t - t_k\right) \mathrm{d}t \tag{3.13}$$

The first two terms of Eq.3.13 are the expected replacement cost of the component. The expected lifetime of the component beyond $T^*_{\mathrm{R},k}$ is the integral and last term of Eq.3.13. To estimate the long-term average maintenance cost per unit time for future components, we use:

1. The ratio of expected replacement cost to expected component lifetime
2. The estimated failure time distribution for the component population

This approach allows us to make reasonable assumptions for approximating $\frac{\mathrm{E}_{\bar{T}_{\mathrm{F}}}[C_{\mathrm{rep}}]}{\mathrm{E}_{\bar{T}_{\mathrm{F}}}[T_{lc}]}$:

1. For an assumed case "without" monitoring, we set the ratio of $\frac{\mathrm{E}_{\bar{T}_{\mathrm{F}}}[C_{\mathrm{rep}}]}{\mathrm{E}_{\bar{T}_{\mathrm{F}}}[T_{lc}]}$ equal to the optimal long-term maintenance cost per unit time. This approach: provides an upper bound for the cost ratio, Increases penalties for early replacements, and results in a more aggressive predictive maintenance policy.

2. For an assumed "perfect" monitoring case. All component replacements are preventive. The average component lifespan equals the mean of the population's failure time distribution. $\frac{\mathrm{E}_{\bar{T}_\mathrm{F}}[C_\mathrm{rep}]}{\mathrm{E}_{\bar{T}_\mathrm{F}}[T_{lc}]} = \frac{c_\mathrm{p}}{\mu_{\bar{T}_\mathrm{F}}}$

3. A value for $\frac{\mathrm{E}_{\bar{T}_\mathrm{F}}[C_\mathrm{rep}]}{\mathrm{E}_{\bar{T}_\mathrm{F}}[T_{lc}]}$ is an average value between option 1 and option 2

but "Perfect" monitoring has been adopted for PdM policy3.

3.4. Predictive maintenance (PdM) planning for replacement

In the First decision setting 3.1, the assumption was new components were always available for immediate replacement, but now we assume a fixed lead time (L) between ordering and delivery i.e. L is multiple of ΔT

3.4.1. Prognostic Performance Metric for PdM Planning focused on Component ordering $\&$ replacement

The replacement cost(preventive or corrective) of components due to late ordering is as follows:

$$C^{(i)}_{\text{delay}} = \max\left(T^{(i)}_{\text{order}} + L - T^{(i)}_{\mathrm{lc}}, 0\right) \cdot c_{\text{unav}} \tag{3.14}$$

where c_{unav} is the unavailability cost per unit of time-related to the shutdown duration. Shutdown begins when the component fails or reaches end-of-life, and shutdown ends when the replacement component arrives.

the holding inventory cost is:

$$C^{(i)}_{\mathrm{stock}} = \max\left(T^{(i)}_{\mathrm{lc}} - \left(T^{(i)}_{\text{order}} + L\right), 0\right) \cdot c_{\text{inv}} \tag{3.15}$$

where c_{inv} is the holding inventory cost per unit time for a component. The total maintenance cost, excluding the cost of the new component, is:

$$C^{(i)}_{\mathrm{m}} = C^{(i)}_{\mathrm{rep}} + C^{(i)}_{\mathrm{delay}} + C^{(i)}_{\mathrm{stock}} \tag{3.16}$$

The long-run expected maintenance cost per unit time is achieved by applying a predictive maintenance policy to n independent components via Eq.3.2

Ideal predictive maintenance strategy for i^{th} component :

1. Replace at $T^{(i)}_{\mathrm{R,perfect}}$: Last time step before failure $T^{(i)}_{\mathrm{F}}$.

2. Order at $T^{(i)}_{\text{order,perfect}} = T^{(i)}_{\text{R,perfect}} - L$: Lead time before replacement.

3. Resulting costs: $C^{(i)}_{\text{stock,perfect}} = 0, C^{(i)}_{\text{delay,perfect}} = 0, C^{(i)}_{\text{rep,perfect}} = 0$ resulting in $C^{(i)}_{\text{m,perfect}} = c_{\text{p}}$

the decision-oriented metric for component ordering and replacement is computed as:

$$\hat{M} = \frac{\frac{\frac{1}{n}\sum_{i=1}^{n}\left(C^{(i)}_{\text{rep}}+C^{(i)}_{\text{delay}}+C^{(i)}_{\text{stock}}\right)}{\frac{1}{n}\sum_{i=1}^{n}T^{(i)}_{\text{lc}}} - \frac{c_{\text{p}}}{\frac{1}{n}\sum_{i=1}^{n}T^{(i)}_{\text{R, perfect}}}}{\frac{c_{\text{p}}}{\frac{1}{n}\sum_{i=1}^{n}T^{(i)}_{\text{R, perfect}}}} \tag{3.17}$$

3.4.2. Simple heuristic PdM policy for component ordering and preventive replacement

At each time step t_k:

1. Check if a replacement component has already been ordered.

2. If not, then based on prognostic data, decide:
   » Order (O): Place a new component order
   » No Order (NO): Wait for the next time step.

$$a_{\text{order},k} = \begin{cases} \text{O}, & \text{if } \Pr\left(RUL_{\text{pred},k} \leq w + \Delta T\right) \geq p^{\text{order}}_{\text{thres}} \\ \text{NO} & \text{else} \end{cases} \tag{3.18}$$

Order lead time is shown by $w = \left\lceil \frac{L}{\Delta T} \right\rceil \cdot \Delta T$. $p^{\text{order}}_{\text{thres}}$ variable heuristic cutoff that has been optimized.

The ordering policy strikes a mix between planning ahead and having what you need at the right time. It looks at the time it takes to order and receive a part and sets the earliest chance to replace it at $t_k + L$. The choice of whether to order (O) or not (NO) depends on a key factor: the chance of needing a new part a little after it arrives, calculated as $t_k + (w + \Delta T)$. This method makes sure that parts are bought early enough to be ready when they're needed, but not so early that they cost too much to store. Once an order is placed, the policy stops anyone from placing another order until the component has been repaired. This runs the maintenance process more smoothly.

4. Model Architecture

The transformer model architecture has been adopted to train the CMAPSS dataset. It uses attention to boost the speed with which the models can be trained. The Transformer was proposed in (Vaswani 2017)

A transformer model is an encoder-decoder architecture in which the encoder components gather features from datasets while the decoder generates class probabilities for classification tasks(Vaswani 2017). The Transformer's architecture comprises two primary components: an encoder stack and a decoder stack(Vaswani 2017). These stacks consist of several identical layers. Nonetheless, this figure is not immutable, and four layers have been selected according to the particular demands of the task outlined by (*Bayesian Hyperparameter Optimization* n.d.) API, which is elaborated upon in chapter 4. The symmetry between the encoder and decoder stacks facilitates efficient processing of input and output sequences, allowing the model to manage intricate time series datasets.

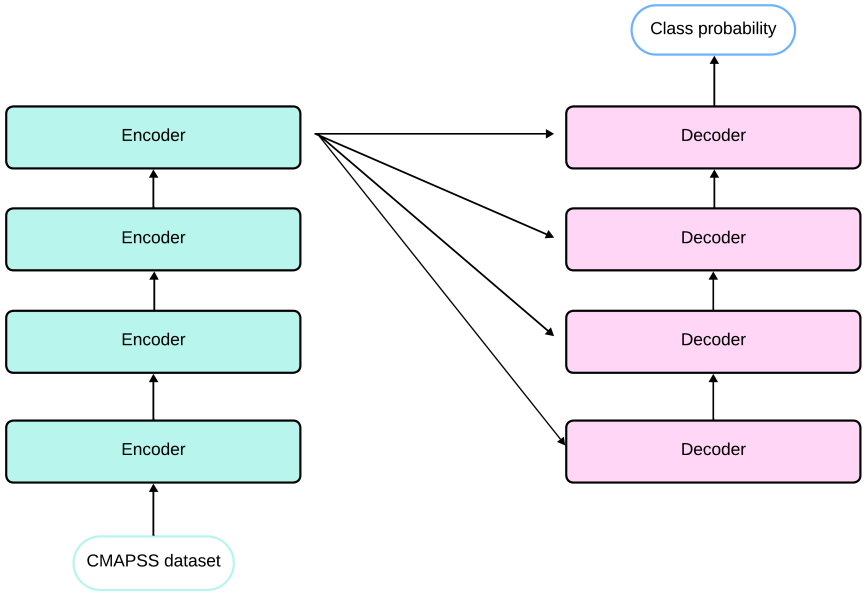

Figure 3 Encoder-Decoder stack

As seen in Fig.3, the Transformer architecture uses a series of similar encoders and decoders, each with a different weight. The input data first passes via a self-attention layer in the encoder, which allows every token to be understood in the context of the entire input sequence. This output is subsequently input into a feed-forward neural network, which is uniformly applied across all positions. The decoder replicates this structure while incorporating an extra attention layer between the self-attention and feed-forward elements. This extra layer lets the processor focus on the input sequence's important parts, making it better at giving accurate results that make sense in the given context. This method expertly handles complex syntactic connections, making Transformers very good at many different natural language processing tasks.

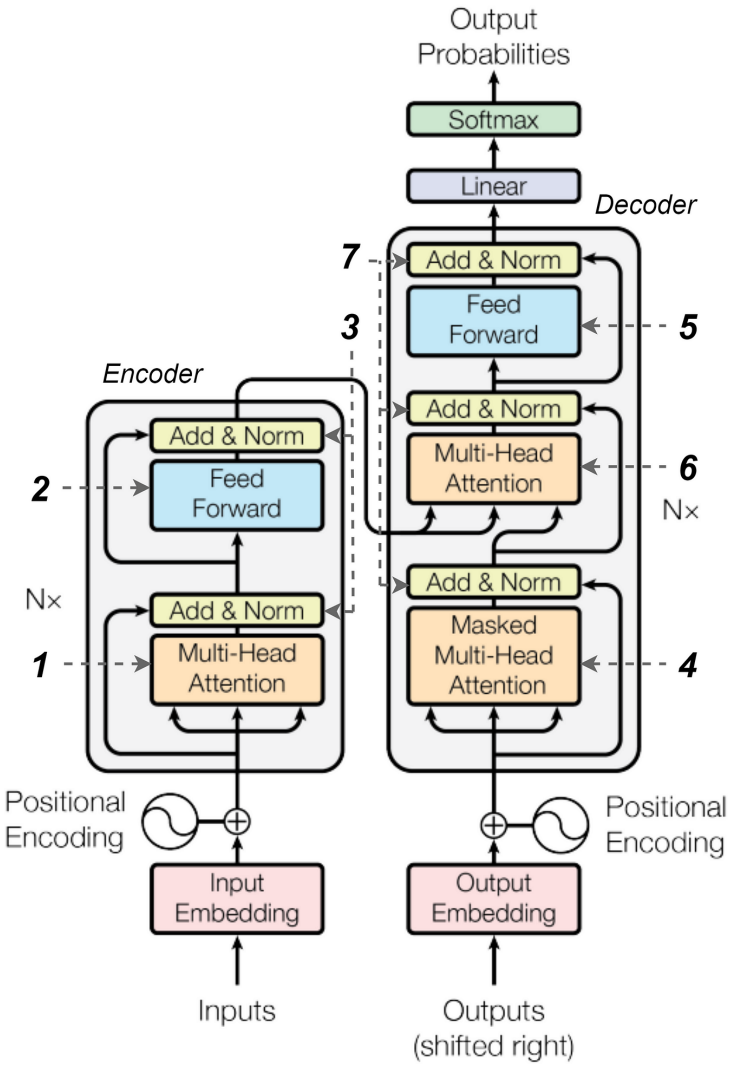

Figure 4 Transformer architecture(Vaswani 2017)

The Transformer encoder stack processes input data sequentially, converting the features of each time step from the input dimension to the model dimension. The API discussed in detail in chapter 4 determined the input and model dimensions. Each subsequent encoder receives a list of vectors with a model dimension size of 32, which may consist of vector embeddings from the bottom layer or outputs from the preceding encoder, as illustrated in Fig.5. The size of the list is a configurable hyperparameter. Following embedding, tokens traverse two essential layers within each encoder: self-attention Fig.5 and feed-forward neural network Fig.4.

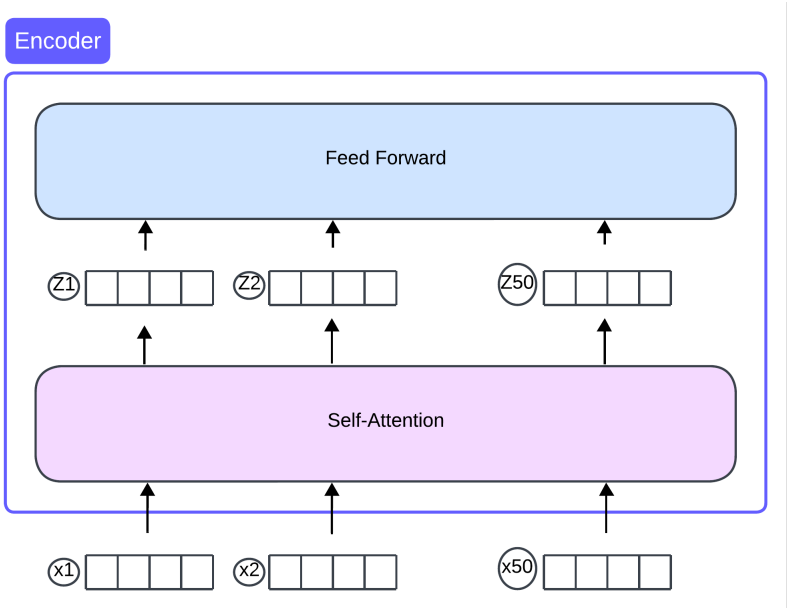

Figure 5 Following the embedding of tokens in the input sequence, each token progresses through both layers of the encoder(*The Illustrated transformer @JayAlammar* n.d.).

The Transformer architecture facilitates parallel processing by permitting each token to traverse its individual pathway within the encoder. The self-attention layer establishes dependencies across various paths, while the feed-forward layer functions independently for each position, promoting efficient parallel computation and improving overall processing performance.

4.1. Self -Attention

1. The initial phase of computing the self-attention mechanism in Transformers involves generating three vectors: Query, Key, and Value, from each input embedding. The vectors are generated by multiplying the embedding with three learned matrices, usually yielding lower-dimensional representations (e.g., 8 dimensions) than the original embeddings (e.g., 32 dimensions). The embedding has been uniformly allocated to each layer. The attention layer will generate eight distinct sets of key, query, and value vectors for each layer, as seen in Fig.8. This dimensionality reduction is an architectural decision to preserve computational efficiency in multi-headed attention, as illustrated in Fig.6.

2. The attention score (Figure 7) is found in the second step of the self-attention analysis.

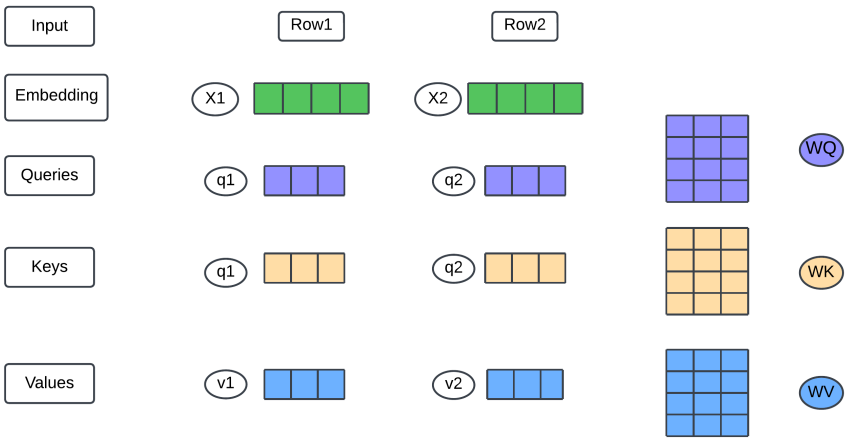

Figure 6 The multiplication of x_1 by the W_Q matrix produces q_1, the query vector, as well as key and value projections for each token in the input sequence(*The Illustrated transformer @JayAlammar* n.d.).

The question, key, and value vectors are theoretical ideas that are used in Transformers' self-attention system. These vectors are made from input embeddings and are used to figure out attention scores, which show how much weight was given to different parts of the input sequence while each token was being encoded. The process involves finding the dot products between the question and key vectors to get scores. These scores are then used to figure out how important different tokens are in the context.

3. The third and fourth steps standardize scores by dividing them by the square root of the main vector dimension (usually 8) and subsequently applying a softmax function, as seen in Fig.9. This procedure stabilizes gradients and guarantees that the final attention weights are positive and total to 1, so forming a probability distribution over the input items.

4. In the sixth stage, Transformers' self-attention system uses softmax scores to figure out how important each character is in the context. It multiplies these scores by their own value vectors, highlighting important tokens and downplaying ones that aren't needed. Look at Figure 9.

5. In the sixth step, the weighted vectors are added together to make the self-attention layer's output for each point. This is then sent to a feed-forward neural network. Matrix processes use this method to make computers work faster.

4.2. Calculation of Self-Attention Matrix

The first stage in the self-attention method entails calculating the Query, Key, and Value matrices by multiplying the input embedding matrix X with the learned weight matrices WQ, WK, and WV, respectively.

The calculation of attention scores, applying softmax, and generating weighted sums are inte-

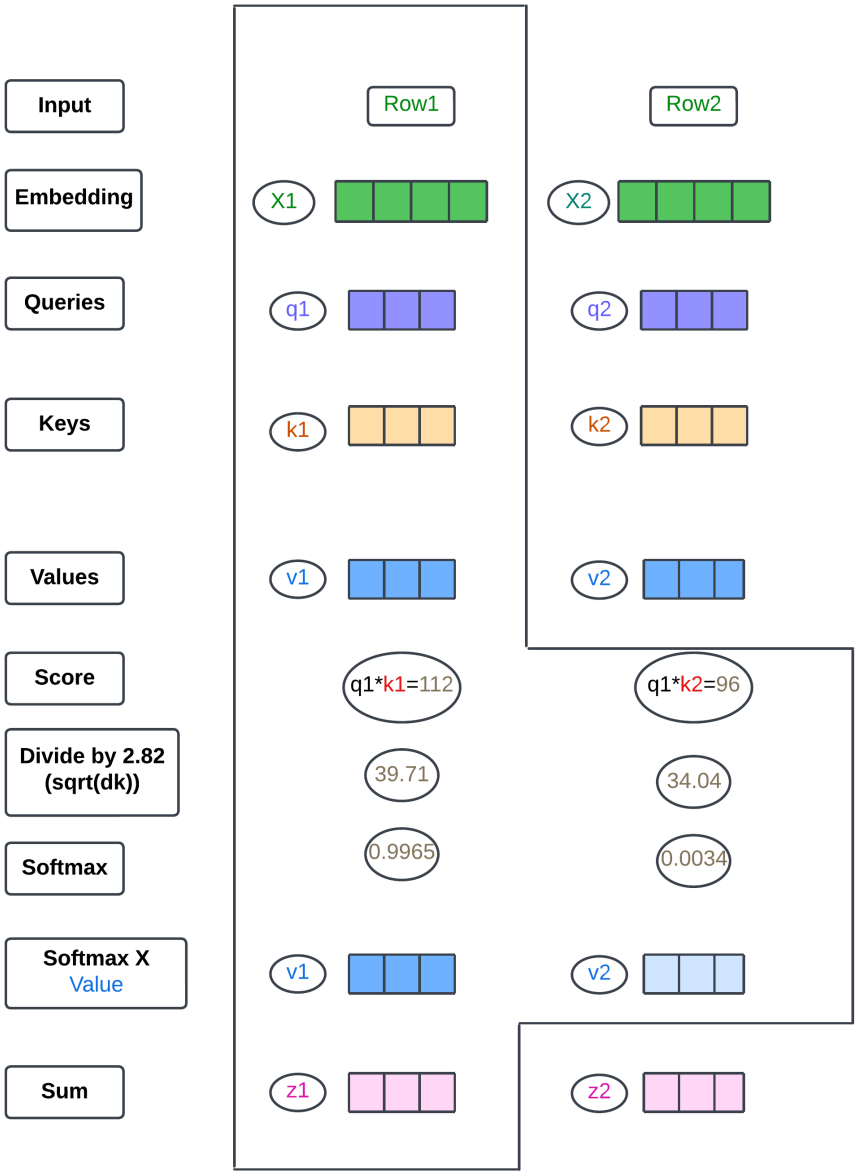

Figure 7 The resulting vector (Z) is fed into the feed-forward neural network(*The Illustrated transformer @JayAlammar* n.d.).

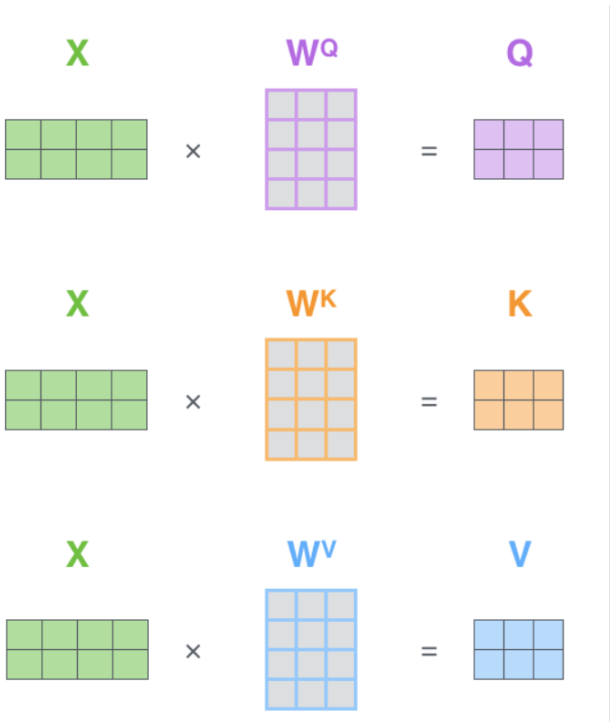

Figure 8 The input embedding matrix X comprises a row for each token in the sequence, with an embedding dimension exceeding that of the generated Query, Key, and Value vector dimensions. For instance, token embeddings may possess 32 dimensions, but Q/K/V vectors are 8-dimensional(*The Illustrated transformer @JayAlammar* n.d.).

grated into a single streamlined operation.

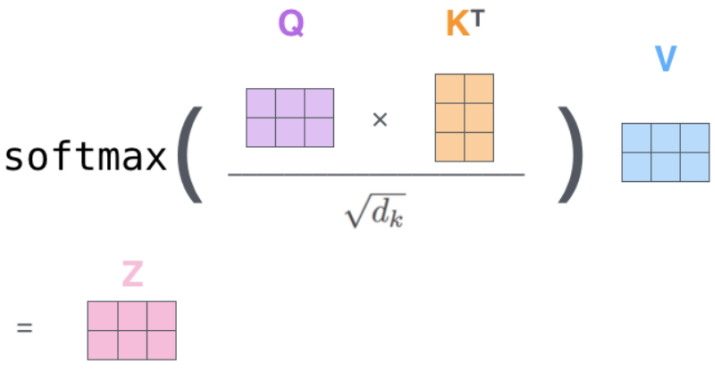

Figure 9 The self-attention score calculation in matrix form(*The Illustrated transformer @JayAlammar* n.d.)

4.3. Multi-headed Attention Matrix

Multi-headed attention Fig.10 in Transformers enhances self-attention by:

1. Draw more attention to the different parts in the input sequence.
2. Creating many "representation subspaces" by using different sets of Query/Key/Value weight matrices.
3. Allowing the model to extract multiple facets of feature information from a single input.
4. enabling better handling of linguistic allusions and complex connections.

   This method enhances the model's ability to analyze and comprehend complex settings. Four headers have been selected according to the unique criteria established by the (*Bayesian Hyperparameter Optimization* n.d.) API, which is elaborated more in chapter 4.

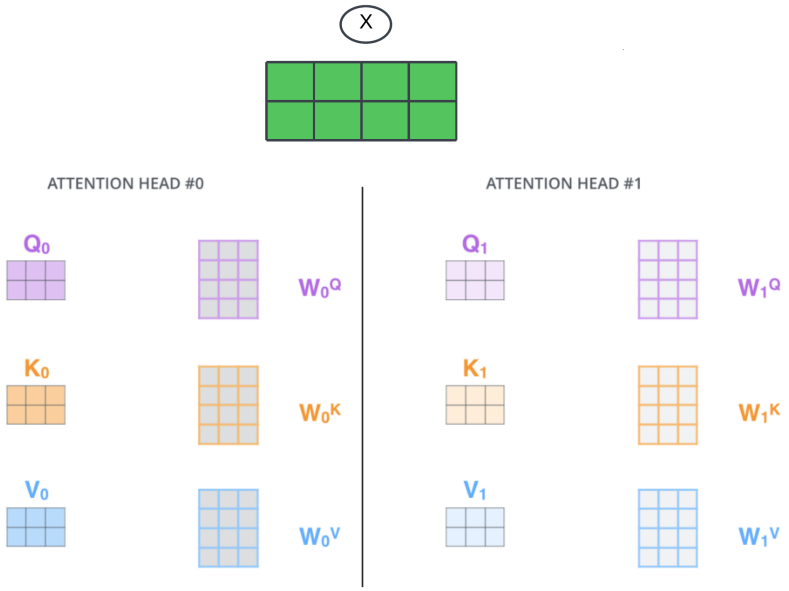

Figure 10 Multi-headed attention in Transformers uses distinct sets of Query, Key, and Value weight matrices for each attention head(*The Illustrated transformer @JayAlammar* n.d.).

The multi-headed attention mechanism in Transformers executes self-attention computations four times using distinct weight matrices, resulting in four individual output matrices. The concatenated matrices are multiplied by an additional weight matrix WO, resulting in a single matrix appropriate for input into the subsequent feed-forward layer (Fig. 11).

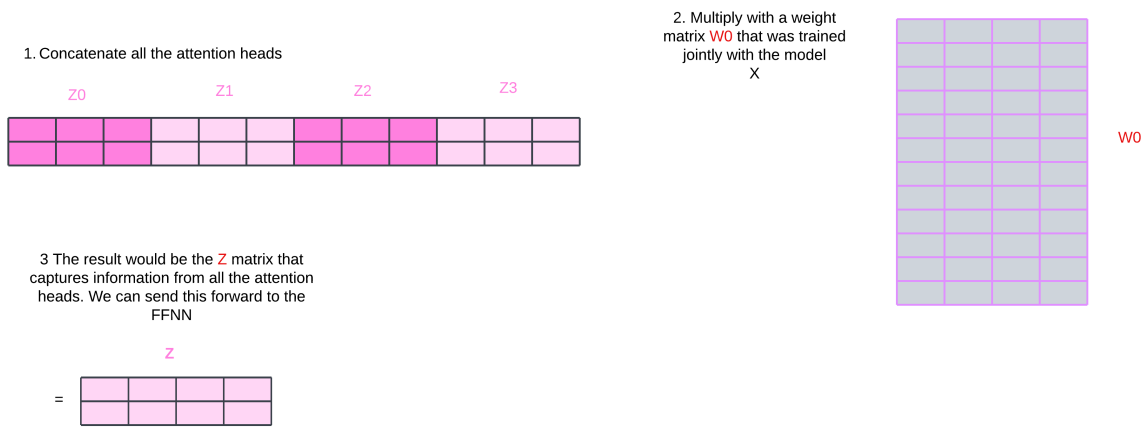

Figure 11 Multi-headed attention in Transformers

4.4. Utilizing Positional Encoding to Represent Sequence Order

Positional encoding is used in the Transformer method, which adds learned position vectors to input embeddings. This method lets the model think about the order of the tokens and their relative places in the input sequence. This makes it better at quickly understanding sequential information.Look at Figure 12.

Suppose the embedding has four dimensions. The location encodings that go with it would look like this:

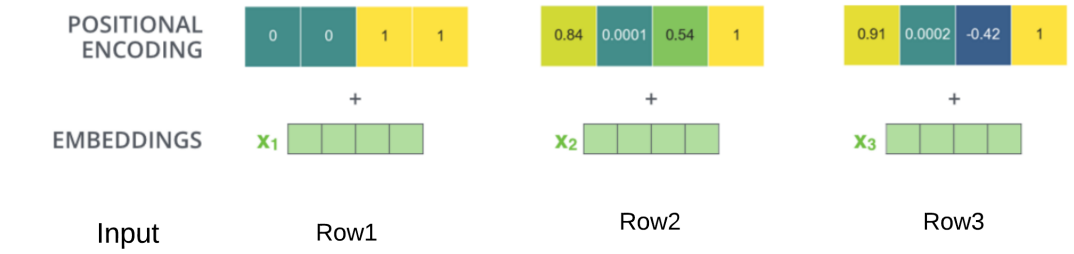

Figure 12 An authentic instance of positional encoding utilizing a toy embedding dimension of 4(*The Illustrated transformer @JayAlammar* n.d.)

Each row of the time series collection represents positional encoding and comprises 32 values, each ranging from -1 to 1.

4.5. The Residuals

In the Transformer design, the encoder and decoder stacks both have residual connections and layer normalization around each sub-layer (self-attention and feed-forward neural network). This architecture improves gradient flow and stabilizes the learning process across the network's depth. Figure 13.

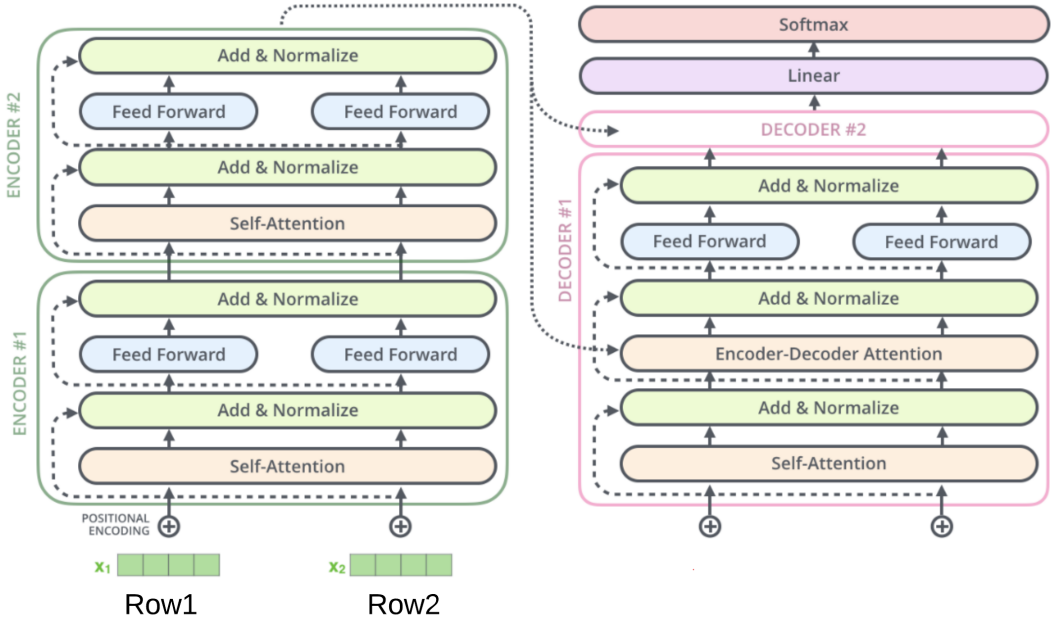

Figure 13 Layer normalization in the Encoder-decoder stack(*The Illustrated transformer @JayAlammar* n.d.)

4.6. The Decoder side

According to Fig.4, the Transformer's decoder works with the encoder, using the encoder's output as attention vectors K and V in its "encoder-decoder attention" layer. The decoder sequentially processes its input, with the output of each stage serving as input for the subsequent phase. Masking makes use of self-attention that only takes into account earlier positions in the output sequence. The encoder-decoder attention layer employs queries from the preceding layer and keys/values derived from the encoder output. This procedure continues until a designated symbol signifies completion, incorporating positional encoding to preserve sequence order information.

4.7. The Final Linear and Softmax Layer

The output of the Transformer's decoder, as depicted in Fig.4, is processed via a Linear layer and a Softmax layer to produce tokens. The Linear layer transforms the decoder's vector into an

expanded logit vector, where each element corresponds to a token in the model's vocabulary. The Softmax layer transforms these logits into class probabilities, with the highest probability associated with the predicted class.

5. Methodologies

5.1. Motivation

In this Master's thesis, I have been working on the optimal prognosis and maintenance of turbofan engines. vanilla transformer architecture has been used to process the multivariate time series data from the C-MAPSS aircraft engine simulator to output the estimated remaining lifetime of a component and then apply a heuristic maintenance strategy using the transformer's output as input. The predictive maintenance policy has already been discussed in chapter 1. This chapter explains data description, code preparation, and tuning of hyperparameters during the training of the transformer model.

5.2. Dataset description

The C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) is a MATLAB/Simulink-based tool that simulates a large commercial turbofan engine. It lets users enter and change health-related parameters and operational profiles, which lets them study how the engine works in different situations and helps with damage modeling by providing flexible simulations and sensor measurements. After ingesting the aircraft engine run-to-failure data (`PM_train.txt`) into a data frame with a shape of (20631,28), the data frame `train_df` has been sorted first by the 'id' column and then by the 'cycle' column. It ensures that the data is ordered by engine ID and cycle number, which may be necessary for certain analyses or modeling tasks. The sorted data frame is then assigned to the variable `train_df`. The final data frame would look like this after assigning the names to the columns.

```
train_df
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
26	100	196	-0.0004	-0.0003	100.0	518.67	643.49	1597.98	1428.63	14.62	...	519.49	2388.26	8137.60	8.4956	0.03	397	2388	100.0	38.49	22.9735
27	100	197	-0.0016	-0.0005	100.0	518.67	643.54	1604.50	1433.58	14.62	...	519.68	2388.22	8136.50	8.5139	0.03	395	2388	100.0	38.30	23.1594
28	100	198	0.0004	0.0000	100.0	518.67	643.42	1602.46	1428.18	14.62	...	520.01	2388.24	8141.05	8.5646	0.03	398	2388	100.0	38.44	22.9333
29	100	199	-0.0011	0.0003	100.0	518.67	643.23	1605.26	1426.53	14.62	...	519.67	2388.23	8139.29	8.5389	0.03	395	2388	100.0	38.29	23.0640
30	100	200	-0.0032	-0.0005	100.0	518.67	643.85	1600.38	1432.14	14.62	...	519.30	2388.26	8137.33	8.5036	0.03	396	2388	100.0	38.37	23.0522

31 rows × 26 columns

Figure 14 CMAPSS dataset after ingestion

5.2.1. Data Preprocessing and Data Wrangling

data preprocessing step, particularly for labeling the data for training purposes. Let's break down steps involved in processing the dataset:

1. **Data Labeling:** This part calculates the Remaining Useful Life (RUL) or Time to Failure for each engine by finding the maximum cycle number (cycle) for each engine ID (id). The result is stored in a data frame RUL with columns 'id' and 'max'.

2. **Merge RUL with Training Data:** the RUL information is merged back into the original training data frame `train_df` based on the engine ID. Each row in `train_df` also has the corresponding maximum cycle number.

3. **Calculate RUL:** calculates the RUL by subtracting the current cycle number ('cycle') from the maximum cycle number ('max') for each engine. This represents how many more cycles the engine will operate before failure.

4. **Drop Unnecessary Columns:** After calculating RUL, the 'max' column used temporarily to calculate RUL is dropped from the data frame as it's no longer needed.

5. **Labeling for Classification:** This part assigns labels to each data point based on the calculated RUL, where thresholds w1 and w0 have been used and assigns ('label1') as 1 if RUL is less than or equal to 'w1', and 0 otherwise. ('label2') as 1 if RUL is less than or equal to 'w1', 2 if RUL is less than or equal to 'w0', and 0 otherwise. The final shape of `train_df` would be (20631,29).

6. **Seperate the dataset:** The Data frame has been separated into a training/validation/test set and used 80% training sets for the training and 10% training sets as validation sets for hyperparameter tuning and the remaining 10% as test set for the PdM policy metric evaluation.

7. **Perform the min-max scaling:** Perform the min-max normalization for the selected columns to train validation and test sets. It ensures that all three datasets will have the same mean and variance.

8. **Perform the min-max scaling:** computes the Min-Max normalization for the selected columns. The resulting normalized values are stored in a new data frame called `norm_train_df`.

9. **Join normalized data frame with the original data frame:** Join the `norm_train_df` with the original data frame `train_df` but exclude the columns that were normalized. The resulting data frame `join_df` has both normalized and original columns that were not normalized.

10. **Reorder columns:** Join the `norm_train_df` In this step, Columns of `join_df` have been reordered to match the orders of columns `norm_train_df` and then assigned back to variable `norm_train_df` and final `train_df` would look like this.

11. **Generate sequences for each engine:** This step a function is defined `gen_sequence` it

```
train_df
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
26	100	196	-0.0004	-0.0003	100.0	518.67	643.49	1597.98	1428.63	14.62	...	519.49	2388.26	8137.60	8.4956	0.03	397	2388	100.0	38.49	22.9735
27	100	197	-0.0016	-0.0005	100.0	518.67	643.54	1604.50	1433.58	14.62	...	519.68	2388.22	8136.50	8.5139	0.03	395	2388	100.0	38.30	23.1594
28	100	198	0.0004	0.0000	100.0	518.67	643.42	1602.46	1428.18	14.62	...	520.01	2388.24	8141.05	8.5646	0.03	398	2388	100.0	38.44	22.9333
29	100	199	-0.0011	0.0003	100.0	518.67	643.23	1605.26	1426.53	14.62	...	519.67	2388.23	8139.29	8.5389	0.03	395	2388	100.0	38.29	23.0640
30	100	200	-0.0032	-0.0005	100.0	518.67	643.85	1600.38	1432.14	14.62	...	519.30	2388.26	8137.33	8.5036	0.03	396	2388	100.0	38.37	23.0522

31 rows × 26 columns

Figure 15 Normalized CMAPSS train dataset with a shape of (16138,30)

is a generator expression that iterates over unique engine IDs in the training data for each engine when called; it generates sequences each sequence is a list of sensor data, and multiple sequences are generated for each engine.

12. **Convert generated sequences into numpy array :** This step concatenates all the generated sequences into a single numpy array. The resulting `seq_array` contains the sequences of sensor data, with shape (12138,50,25) for training and `seq_array_validation` with a shape of (1742, 50, 25) for validation.

13. **Defined generator to generate labels:** This step defined a function `gen_labels` it generates labels for each sequence of sensor data. It ensures that the labels are correctly aligned with the sequences and handles the special case where the first sequence uses the last label as its target.

14. **Data preparation to generate labels:** This step extracts the columns specified by the label from the data frame `id_df` and converts them to a numpy array. It selects only the relevant label(s) needed for generating sequences.

15. **Label generation:** This step returns the labels associated with each sequence. It removes the first `seq_length` labels because, for each engine (id), the first sequence of size `seq_length` uses the last label as its target. The previous labels are discarded. All subsequent sequences for the same engine (id) will have one label associated with them and give the shape of (12138,1) and (1742,1) to corresponding `label_array` and `label_array_validation` respectively.

16. **Apply One-hot encoding to integer labels:**One-hot encoding converts these integer labels into binary vectors, where each vector has a length equal to the number of classes and contains a 1 in the position corresponding to the class and 0s elsewhere and give the shape of (1742,3)and (12138,3)to corresponding `dummy_label_array_validation` and `dummy_label_array` respectively.

17. **Train Dataset:** MultivariateTimeSeriesDataset class has been used to wrap the input data and labels for easy access during training. It takes `seq_array` (12138, 50, 25), `dummy_label_array` (12138, 3), `seq_length` (50) as input and returns Returns a single sample with shape (50, 25) and its corresponding label repeated 50 times (50, 3).

18. **Data Loader:** It batches the data for both the train and validation set and provides an iterable over the dataset. It takes `train_dataset`, `batch_size`, and to preserve the temporal order of the time series data, keep shuffle False.

5.2.2. Training of transformer model and Hyperparameter tuning

I have been using an online platform, Weights & Biases, to tune the hyperparameters of the transformer model when training the model for CMAPSS datasets. A hyperparameter in deep learning is a configuration variable set before the training begins and controls various aspects of the model's architecture or learning process. Bayesian hyperparameter optimization has been adopted here to find the best hyperparameters of the transformer model. These are the following steps involved in Hyperparameter tuning:

1. Pick a combination of hyperparameter values to train the model.
2. get the accuracy of model.
3. Update the belief that can lead to model improvement.
4. terminate when classification accuracy is maximized.

This platform, Weights & Biases (*Bayesian Hyperparameter Optimization* n.d.), works on the principle of 250+ year-old Bayes principle. It reads What is the probability of Y given X?

$$P(Y|X) = (P(X|Y) \cdot P(Y))/P(X) \tag{5.1}$$

So, if we try to apply the same principle to hyperparameter tuning, the Eq.5.1 would transformed into the equation.

$$P(\text{Accuracy}|\text{hyperarameter\_combination}) = \frac{P(\text{hyperarameter\_combination}|\text{Accuracy})P(\text{Accuracy})}{P(\text{hyperarameter combination})} \tag{5.2}$$

So, now, each of the above terms would be:

1. $P($Accuracy $|$ hyperarameter_combination $)$ gives the probability of given accuracy to be

maximized given then the combination of hyperparameter values.

2. $P(\text{hyperarameter\_combination} \mid \text{Accuracy})$ gives the probability of a certain hyperparameter combination if the given accuracy is maximized.

3. $P(\text{Accuracy})$ gives the initial accuracy probability.

4. $P(\text{hyperarameter\_combination})$ is the probability of getting that particular hyperparameter combinations.

Bayesian hyperparameter optimization allows us to build a probabilistic model for the objective function that we are trying to maximize to train our transformer model. Here, accuracy is the objective function that we are trying to maximize. This probabilistic model is referred to as a black-box model for the objective function and is represented as $P(\text{Accuracy}|\text{hyperarameter\_combination})$ It is much easier to optimize a surrogate model than the objective function. This is because surrogate models in hyperparameter optimization cut down on the number of calls to the computationally expensive objective function by choosing promising sets of hyperparameters more efficiently. This makes the search process more efficient overall, especially in spaces with many features. This is why Bayesian hyperparameter optimization is preferable when we tune the hyperparameter to gain the best accuracy of our model. The main objective of Bayesian reasoning is to become "less wrong" with more data. The Fig.16 illustrated below shows the tuned hyperparameter.

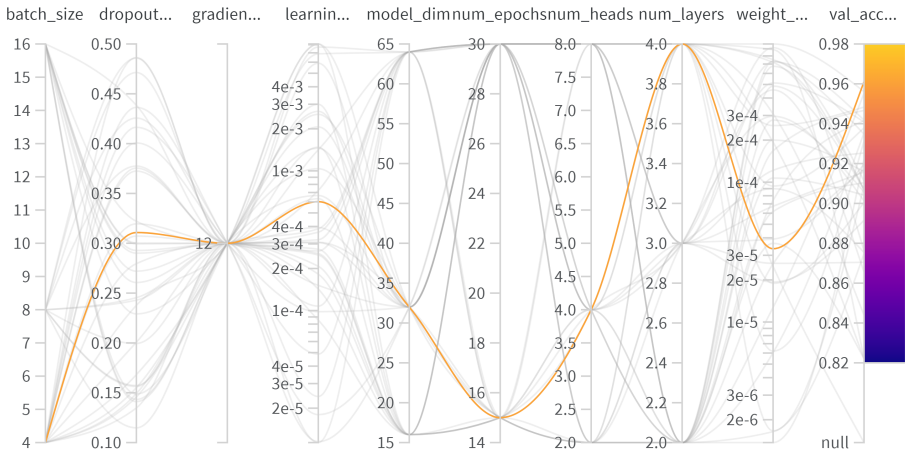

Figure 16 Tuning of hyperparameters values based on the combination of hyperparameters (*Bayesian Hyperparameter Optimization* n.d.)

There are the following model parameters and their significance:

1. **Input Dimension:** is the number of attributes in the training data.

2. **Model Dimension:** it is used when input dimension transformed into model dimension and used in the internal calculation of transformer.

Parameters name	Values
Input Dimension	25
Model Dimensio	32
Number of Heads	4
Number of Layers	4
Sequence Length	50
Number of Classes	3
Dropout Rate	0.310775
Learning Rate	0.000602
Batch Size	4
Number of Epochs	15
Weight Decay	0.000033
Patience	5
Gradient Accumulation Steps	12

Table 1 Tuned hyperparameter values of the Transformer model

Hyperparameters	Hyperparameters Range
"Methods"	"Bayes"
"Metric Name"	"Validation Accuracy"
"Metric Goal"	"Maximize"
"Model Dimension"	[16, 32, 64]
"Number of Heads"	[2, 4, 8]
"Number of Layers"	[2, 3, 4]
"Dropout Rate"	{"min": 0.1, "max": 0.5}
"Learning Rate"	{"min": 1e-5, "max":1e-2}
"Batch Size"	[4, 8, 16]
"Number of Epochs"	[30]
"Weight Decay"	{"min": 1e-6, "max":1e-3}
"Gradient Accumulation Steps"	[12]
"Number of Trials"	10

Table 2 A combination of hyperparameters values used for Tuning the model

3. **Number of Heads:** The number of attention heads in each multi-head attention layer.

4. **Number of Layers:** The number of encoder-decoder layers.

5. **Sequence of Length:** Represents nr of tokens in each sequence. each sequence contains 50 data points or tokens.

6. **Number of Classes:** Represents nr of classes to be classified by model.

7. **Dropout Rate:** Represents probability of randomly deactivating neurons during training. It prevents overfitting and improves generalization.

8. **Learning Rate:** It controls the step size of weight updates during training.

9. **Batch Size:** This is the number of sequences processed together in a single forward/backward pass. The transformer model processes 4 sequences, i.e., processing 50*4 =200 data points (tokens).

10. **Number of Epochs:** This is the number of sequences processed together in a single forward/backward pass. The transformer model processes 4 sequences, i.e., processing 50*4 =200 data points (tokens).

11. **Weight Decay:** It adds a penalty term to the loss function and prevents overfitting of the model during training.

12. **Patience:** Training will stop if the model doesn't improve for 5 consecutive epochs.

13. **Gradient Accumulation Steps:** it is helpful when massive models are reluctant to fit a large batch into memory.

5.2.3. Implementation of Predictive Maintenance Policy1 With component ordering and replacement

This is a simple heuristic Pdm Policy that determines at each time step t_k whether a component should be preventively replaced or not. The assumption here is that the new component is readily available when a preventive replacement is decided or a corrective replacement is imposed, as referred to in policies. section(3.3.2, 3.1)

The following steps are taken in the implementation of Predictive Maintenance Policy1:

1. The function $calculate\_probabilities\_for\_pdm\_policy\_1\_With\_ordering$ calculates the probabilities and costs associated with a preventive maintenance policy for a given dataset.

2. Empty arrays like t_order_array, t_lc_array, and $costs\_rep\_array$, $costs\_delay\_array$, and $costs\_stock\_array$ have been initialized to store the values of corresponding ordering cycles and replacement cycles, and corresponding replacement costs, delay costs and holding inventory costs respectively

3. Current cycles have been checked at discrete time steps as mentioned in decision settings

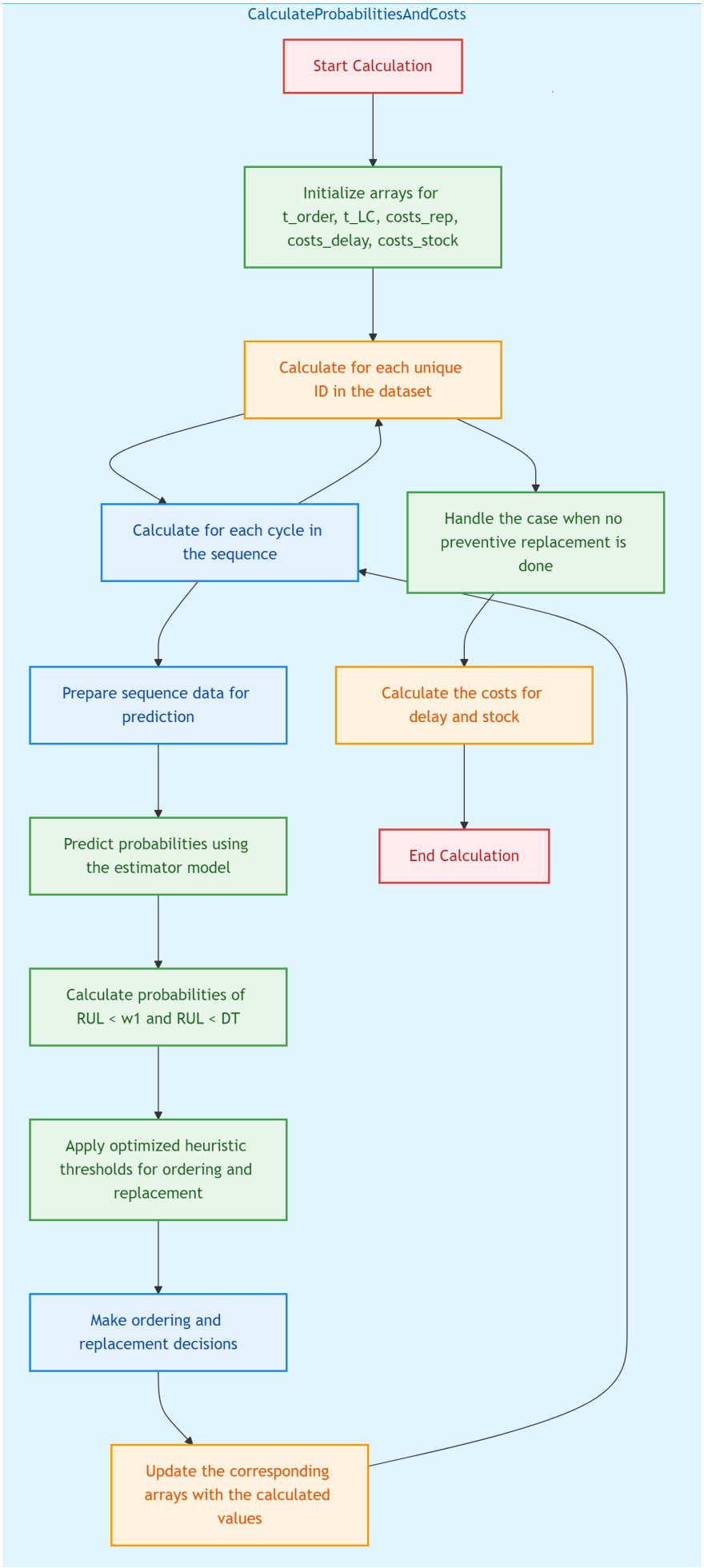

Figure 17 Flow chart of Code implementation of PdM Policy1 with Ordering and Component Replacement.

1.2.

4. The probabilities of each unique engine ID have been calculated for their corresponding cycles in the dataset using the trained transformer model. It predicts the remaining useful life (RUL) of the components and gives probabilities of class 0(high RUL), class 1(medium RUL), class 2(low RUL).

5. Preventive replacement decisions have been made at discrete time steps if class 2 probabilities would be greater than optimized threshold values, which is 0.5.

6. Ordering decisions have been made if the collective probabilities of class 1 and class 2 would be greater than the threshold optimized value, which is 0.11 in our case.

7. At the end, the decision times, replacement times, and associated costs for each component have been updated and returned

5.2.4. Implementation of Predictive Maintenance Policy2 and Policy3 With component ordering and replacement

The following steps are taken in the implementation of Predictive Maintenance Policy2 and Policy3:

1. The function $calculate\_probabilities\_for\_pdm\_policy\_2\_With\_ordering$ calculates the probabilities and costs associated with a preventive maintenance policy for a given dataset.

2. Empty arrays like t_order_array, t_lc_array, and $costs\_rep\_array$, $costs\_delay\_array$, and $costs\_stock\_array$ have been initialized to store the values of corresponding ordering cycles and replacement cycles, and corresponding replacement costs, delay costs and holding inventory costs respectively

3. Current cycles have been checked at discrete time steps as mentioned in decision settings 1.2.

4. The probabilities of each unique engine ID have been calculated for their corresponding cycles in the dataset using the trained transformer model. It predicts the remaining useful life (RUL) of the components and gives probabilities of class 0(high RUL), class 1(medium RUL), class 2(low RUL).

5. The Optimal replacement time $T\_R\_k\_optimal$ for each unique engine ID has been calculated for their corresponding cycles in the dataset using the objective function(3.11) based on the predicted probabilities in the previous step.

6. Preventive replacement decisions have been made at discrete time steps if $\Delta T = 10$ flight cycles would be greater than Optimal replacement time $T\_R\_k\_optimal$ else do nothing.

7. Ordering decisions have been made if the collective probabilities of class 1 and class 2 would be greater than the threshold optimized value, which is 0.11 in our case.

8. At the end, the decision times, replacement times, and associated costs for each compo-

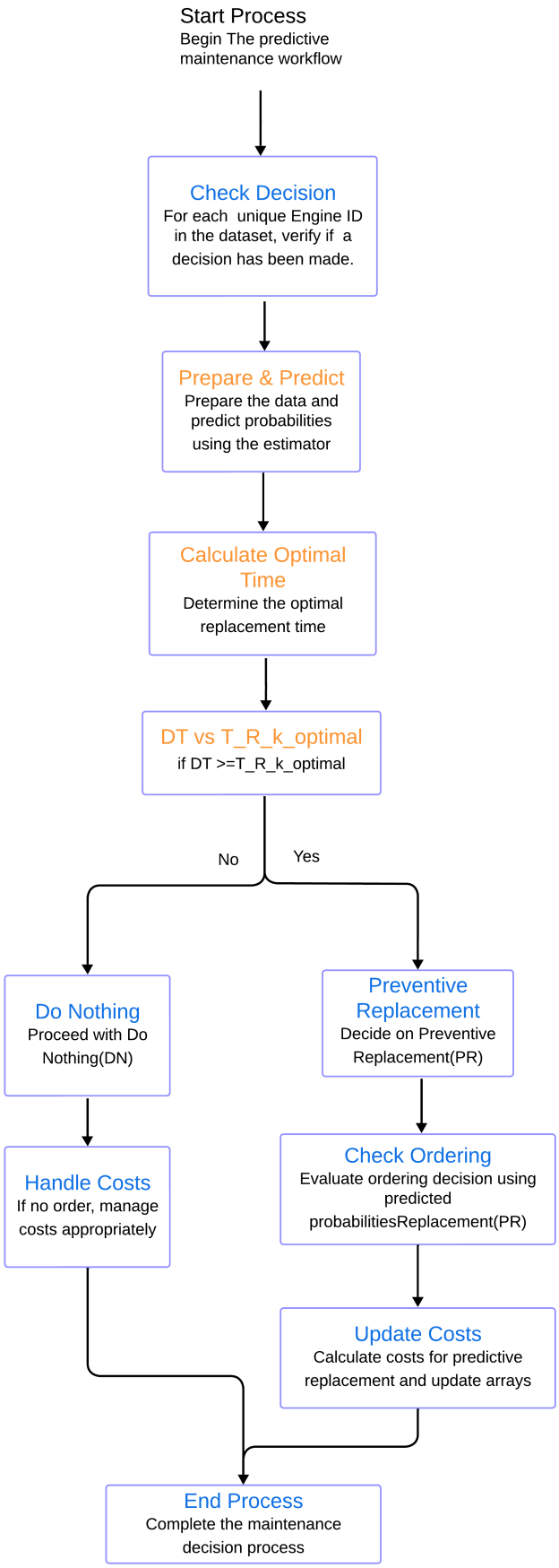

Figure 18 Flow chart of Code implementation of PdM Policy2 and PdM Policy3 with Ordering and replacement of component.

nent have been returned

So, the implementation of Predictive Maintenance Policy 2 and Policy 3 for component ordering and replacements follow the same steps. The only difference here is that the calculation of the Optimal replacement time is different since the objective functions used in the optimization are different.Sec(3.11, 3.13)

6. Results

The findings of predictive maintenance policies using different models have been discussed in this chapter. LSTM model (Kamariotis et al. 2024b) and Transformer models have been compared based on different metric scores to determine the best model for the CMAPSS dataset. The Predictive maintenance policies have been evaluated based on the decision-oriented metric section(3.17) via using each component's decision times, replacement times, and associated costs as was explained in section(5.2.3, 5.2.3). The objective function behavior of PdM policy 2 and policy 3 has been discussed, and changes in the objective function behavior have been explained with their respective plots. In the end, the replacement cycles of the LSTM and transformer models were compared.

6.1. Training model comparison

1. **F1-score:** LSTM outperforms Transformer in all three classes.
2. **F2-score:** LSTM performs better in two classes, while Transformer is slightly better in one class.
3. **Precision:** LSTM is better for two classes, but the Transformer achieves perfect precision for the third class.
4. **Recall:** LSTM performs better in two classes, while the Transformer has a slightly higher recall in one class.

where:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{6.1}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{6.2}$$

F1 score, which is a harmonic mean of precision and recall, indicates a better balance between precision and recall.
True Positives: Those are positive instances correctly classified as positive instances.
False Positives: Those are negative instances incorrectly classified as positive instances.

False Negatives: Those are positive instances incorrectly classified as negative instances.

Based on the scores in the table 3, LSTM performed better in the classification task than the transformer.

Score	LSTM			Transformer		
	Class 0	Class 1	Class 2	Class 0	Class 1	Class 2
F1-score	0.9864	0.8686	0.9246	0.9796	0.7773	0.8957
F2-score	0.9887	0.8442	0.9470	0.9913	0.7323	0.8429
Precision	0.9826	0.9125	0.8897	0.9606	0.8662	1.00
Recall	0.9902	0.8287	0.9625	0.9993	0.7050	0.8111

Table 3 A comparison table of different scores between LSTM and transformers model

Confusion matrix Transformer = $\begin{bmatrix} 72000 & 50 & 0 \\ 2950 & 7050 & 0 \\ 0 & 1039 & 4461 \end{bmatrix}$

Confusion matrix LSTM = $\begin{bmatrix} 9654 & 94 & 0 \\ 169 & 1326 & 105 \\ 0 & 33 & 847 \end{bmatrix}$ Based on the confusion matrix of the transformer, the following information has been deduced:

1. The transformer has been predicting 72000 times correctly, and only 50 instances were misclassified for class 0
2. The transformer has been predicting 7050 times correctly, and 2950 instances were misclassified for class 1.
3. The transformer has been predicting 4461 times correctly, and 105 instance and 1039 instances were misclassified for class 2.

Based on the confusion matrix of the LSTM, the following information has been deduced:

1. The LSTM has been predicting 9654 times correctly, and only 94 instances were misclassified for class 0
2. The LSTM has been predicting 1326 times correctly for class 1, but 169 instances and 105 instances were misclassified as class 0 and class 2, respectively.
3. The LSTM has been predicting 847 times correctly, and 33 instances were misclassified for class 2.

where class 0, class 1, and class 2 represent classes of high RUL, medium RUL, and low RUL, respectively. The trainable parameters of LSTM are 80753 less than the trainable parameter of the transformer, which is 88163. Still, due to the attention mechanism in the transformer, all the tokens in the sequence are processed parallelly, whereas LSTM processes the tokens sequentially with memory. The transformer model starts overfitting after epoch=8, whereas LSTM starts overfitting after epoch=10.

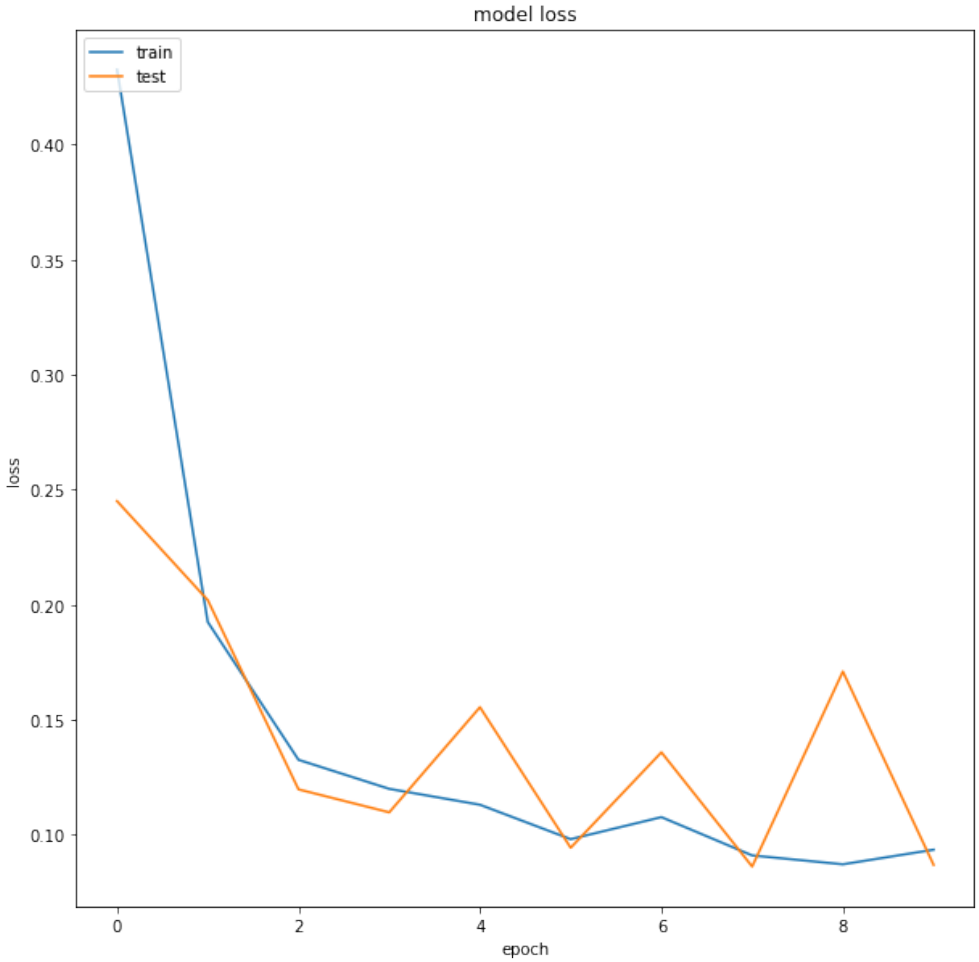

Figure 19 Train-Validation curve of LSTM

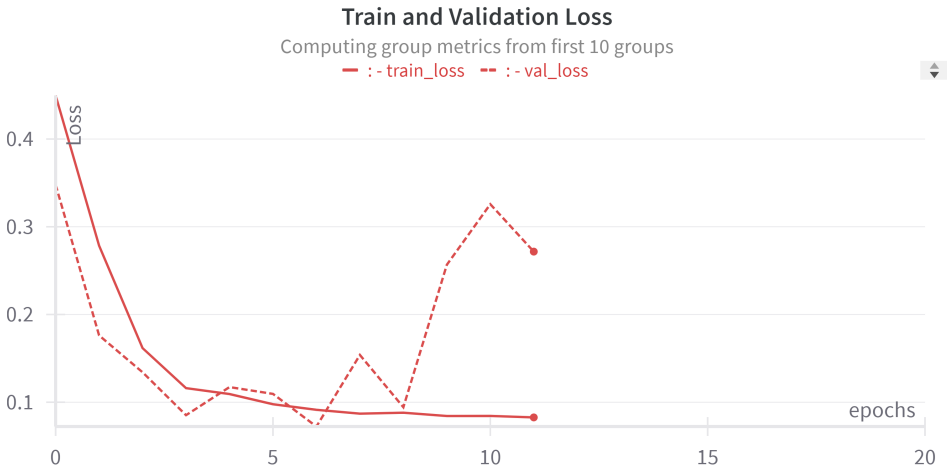

Figure 20 Train-Validation curve of Transformer

6.2. Predictive Maintenance Policy Evaluation

In Fig.21 below, one can see that the replacement action has been taken when low RUL probabilities exceeded the threshold value of 0.5 section(3.9). The sharp increase in probabilities occurred at the discrete time steps just before the failure event, which helped us track turbojet engine replacement cycles.

In Fig.22 below, one can see the replacement action has been taken when $\Delta T = 10$ flight cycles are greater optimal replacement time section(3.10). The sharp decrease in optimal replacement time occurred at the discrete time steps just before the failure event, when optimal replacement is below 10, and replacement action has been taken

In Fig.23 below, one can see the replacement action has been taken when $\Delta T = 10$ flight cycles are greater optimal replacement time section(3.10). The objective function to calculate the optimal replacement time has been modified for Policy 3. The sharp decrease in optimal replacement time occurred at the discrete time steps just before the failure event, but it also followed a zig-zag curve before diving down to below 10 flight cycles for some engine components.

In Fig.24 below, The prognostic performance metric based on the transformer model has been a monotonically decreasing curve for all the policies, and the curves of policy 2 and policy 3 with component ordering and replacement have been superimposed on each other. Similarly, policy 2 and policy 3 curves with component replacement have been superimposed on each other since their decision metric values were exactly the same at different $C\_p/C\_cratios$.

The decision metric observed in the Transformer model is very high as compared to LSTM when $C\_p/C\_c < 0.4$; the heuristic thresholds used were optimized on the training set using the LSTM

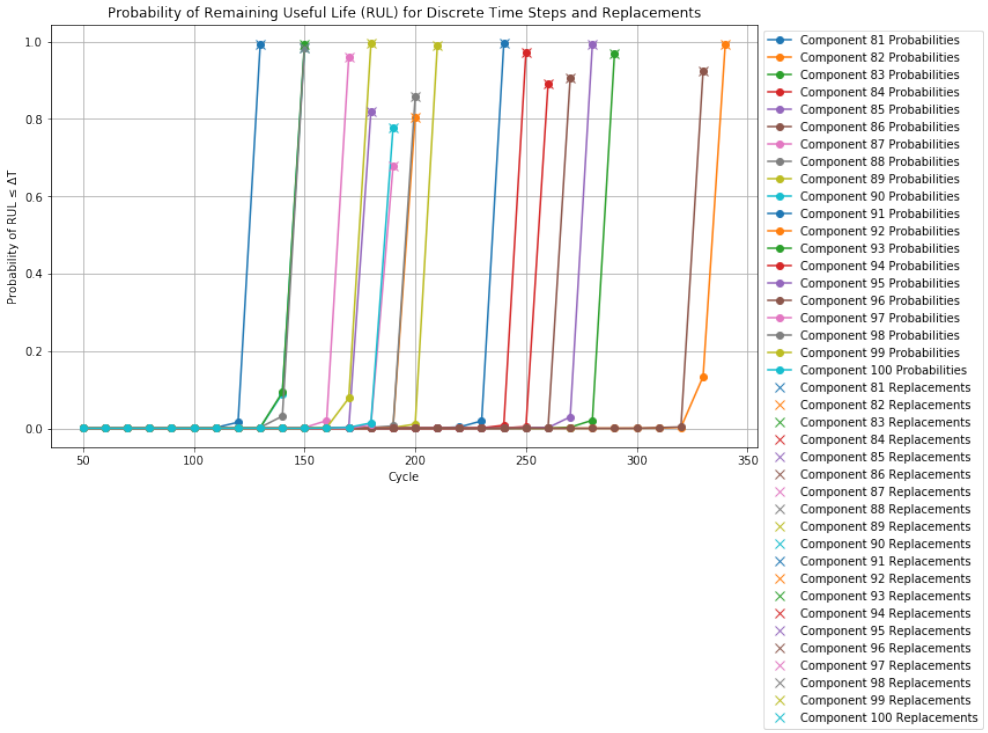

Figure 21 Plot of both Overall and Replacement Probabilities Vs Cycles, for Policy 1

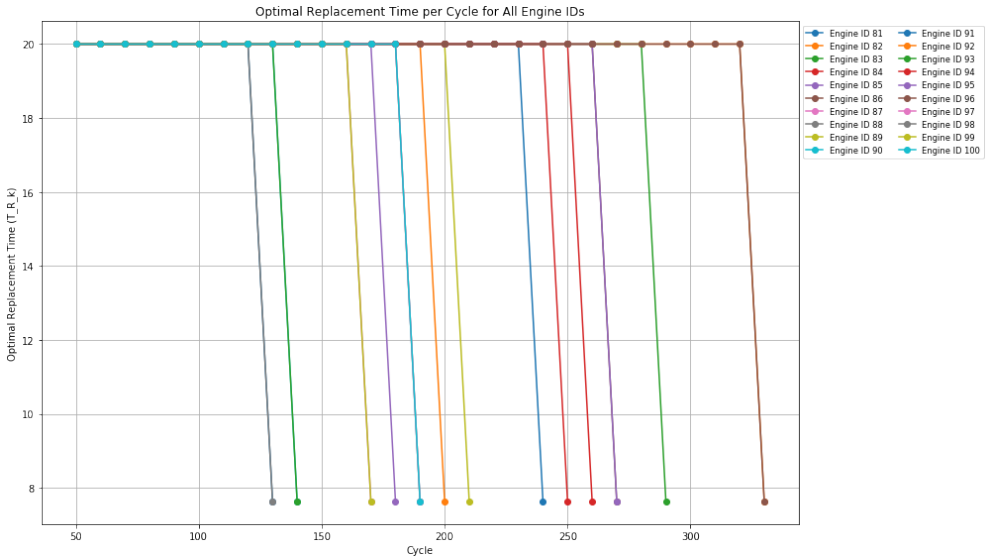

Figure 22 Optimal replacement time Vs. Cycles for Policy 2

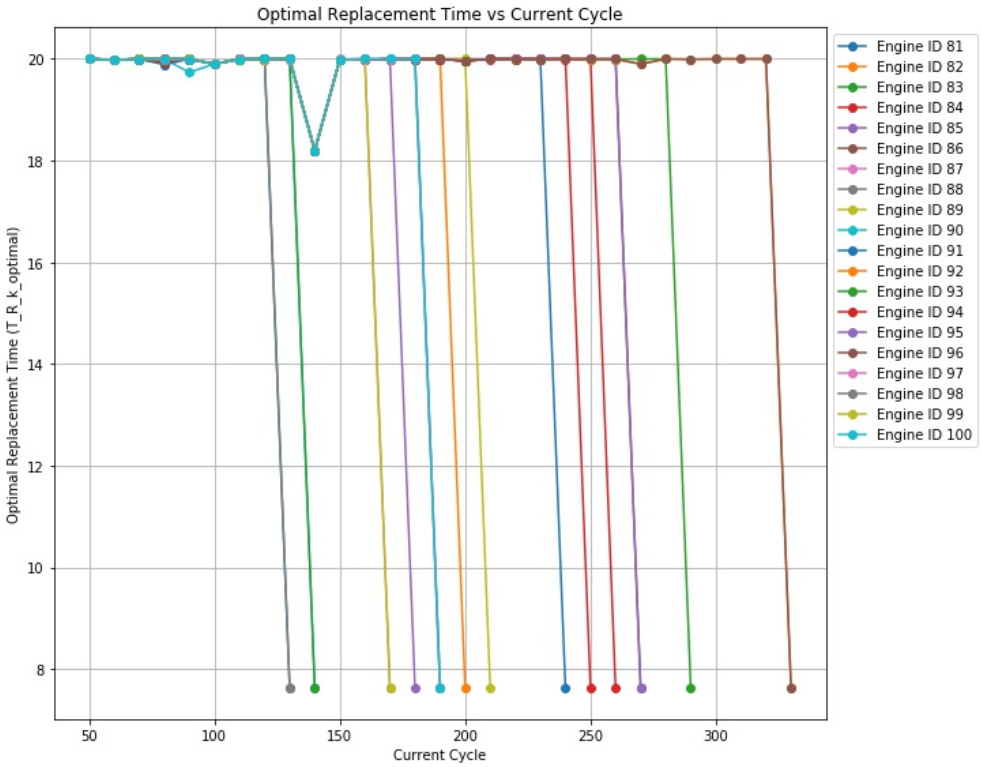

Figure 23 Optimal replacement time Vs. cycles for Policy 3

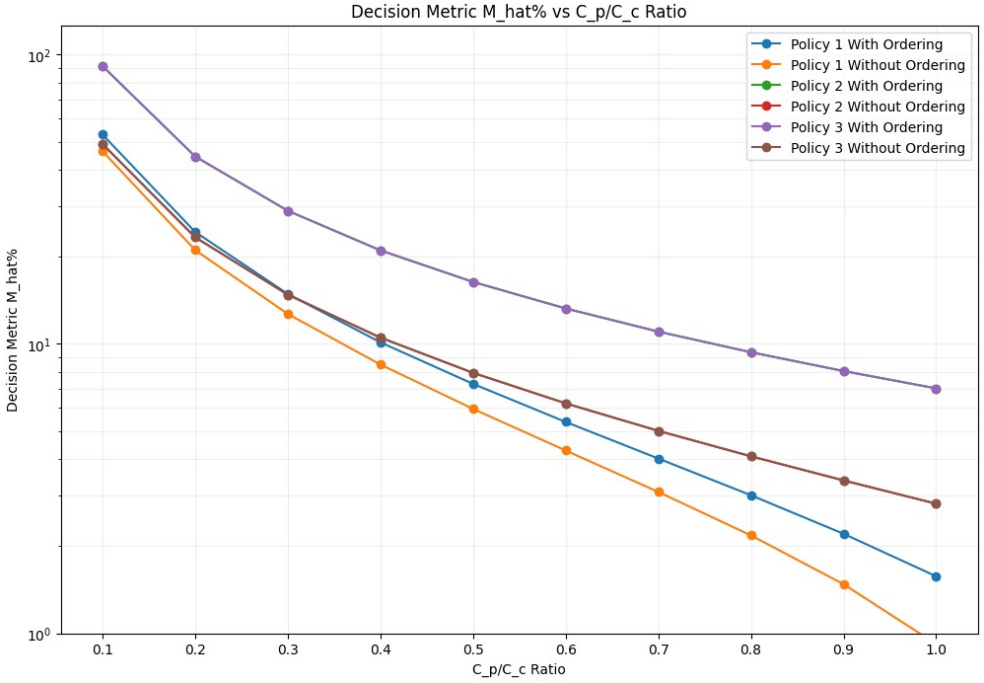

Figure 24 Decision metric in Percentage for all the PdM policies using Transformer model Vs. C_p/C_c ratios

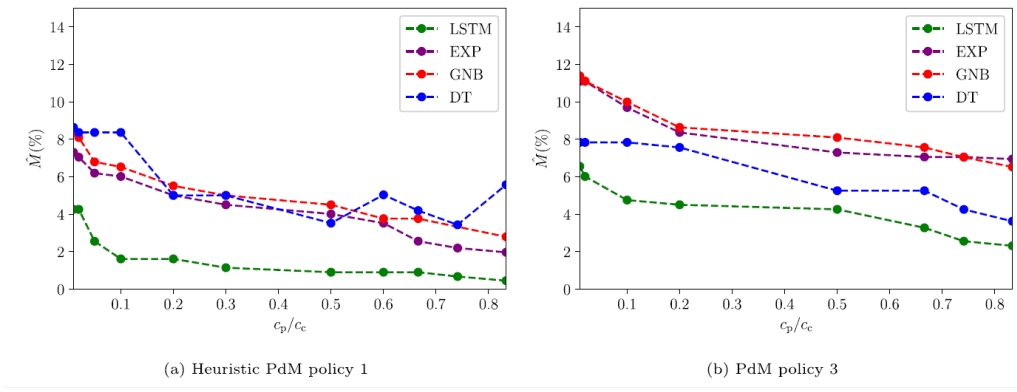

Figure 25 Decision metric in Percentage (Kamariotis et al. 2024b) for the PdM policy1 and Policy3 Vs C_p/C_c ratios

model, which might be one of the possible reasons for the poor performance of the Transformer model.

7. Conclusion

7.1. Summary

A full study on predictive repair has been carried out using advanced machine learning methods. Datasets were carefully cleaned up and used to train a Transformer model. This model was compared to a standard Long Short-Term Memory (LSTM) model(Kamariotis et al. 2024b)

The datasets were initially preprocessed to ensure optimal input for the models. Subsequently, a Transformer architecture was implemented and trained using these processed datasets. The trained model was then employed to generate prediction probabilities, which were utilized in three distinct predictive maintenance policies.

It was seen that the Transformer model made much better decisions than the LSTM model, especially when the ratio of the cost of preventive maintenance to the cost of corrective maintenance (C_p/C_c) was less than 0.4. the heuristic thresholds used were optimized on the training set using the LSTM model, which might be one of the possible reasons for the poor performance of the Transformer model.

The research thoroughly analyzed both models' performance across various maintenance scenarios. The Transformer model's ability to capture complex temporal dependencies was highlighted, although its performance was potentially hindered by threshold optimization tailored to the LSTM model.

This study provides valuable insights into applying state-of-the-art machine learning techniques in predictive maintenance, offering a foundation for future research and practical implementations in industrial settings.

7.2. Future Scope

Based on using deep learning models like Transformers and LSTMs in predictive maintenance, it has much potential for the future. Here is a short summary of the possible future directions:

1. Better accuracy in predicting: Transformers and LSTMs can handle complex temporal dependencies, which could make it easier to guess when equipment will break down(Serradilla et al. 2022).

2. Integration with IoT and sensor data: These models can handle huge amounts of real-time data from IoT devices and sensors, which lets them do more complex analyses.

Taking care of unstructured data: Deep learning models can use natural language processing to learn from unstructured data sources like technician notes and maintenance logs.

4. Improving maintenance schedules: AI-driven algorithms can help make smarter maintenance schedules that reduce downtime and operational costs.

5. Finding strange things: Deep learning models can find small trends and things that people might miss.

6. Hybrid approaches: Using data-driven deep learning methods and physics-based models to make predictive maintenance solutions more reliable(Serradilla et al. 2022).

7. Transfer learning: Using models already trained on new maintenance tasks so that big datasets aren't needed for each one.

8. Explainable AI: creating deep learning models that can be understood to show how predictions are made, which is very important for widespread use in industry.

Integrating edge computing means putting the best versions of these models on edge devices so that they can do real-time, on-site predictive repair.

10. Ongoing learning: Using online learning methods to adapt models as equipment conditions change over time.

These improvements in using deep learning for predictive maintenance could make assets much more reliable, lower maintenance costs, and make operations more efficient across many businesses.

Bibliography

Arcieri, Giacomo, Cyprien Hoelzl, Oliver Schwery, Daniel Straub, Konstantinos G Papakonstantinou, and Eleni Chatzi (2023). “Bridging POMDPs and Bayesian decision making for robust maintenance planning under model uncertainty: An application to railway systems”. In: *Reliability Engineering & System Safety* 239, p. 109496 (cit. on p. 12).

Bayesian Hyperparameter Optimization (n.d.). https://wandb.ai/wandb_fc/articles/reports/What-Is-Bayesian-Hyperparameter-Optimization-With-Tutorial---Vmlldzo1NDQyNzcw. [Online; accessed 1-12-2014] (cit. on pp. 24, 30, 37, 38).

Benaggoune, Khaled, Safa Meraghni, Jian Ma, Hayet Mouss, and Noureddine Zerhouni (May 2020). “Post Prognostic Decision for Predictive Maintenance Planning with Remaining Useful Life Uncertainty”. In: pp. 194–199. DOI: 10.1109/PHM-Besancon49106.2020.00039 (cit. on p. 12).

Chao, Manuel Arias, Chetan Kulkarni, Kai Goebel, and Olga Fink (2022). “Fusing physics-based and deep learning models for prognostics”. In: *Reliability Engineering & System Safety* 217, p. 107961 (cit. on p. 12).

Fink, Olga (2020). “Data-driven intelligent predictive maintenance of industrial assets”. In: *Women in Industrial and Systems Engineering: Key Advances and Perspectives on Emerging Topics*, pp. 589–605 (cit. on p. 12).

Fink, Olga, Qin Wang, Markus Svensen, Pierre Dersin, Wan-Jui Lee, and Melanie Ducoffe (2020). “Potential, challenges and future directions for deep learning in prognostics and health management applications”. In: *Engineering Applications of Artificial Intelligence* 92, p. 103678 (cit. on p. 12).

Galar, Diego, Kai Goebel, Peter Sandborn, and Uday Kumar (2021). *Prognostics and Remaining Useful Life (RUL) Estimation: Predicting with Confidence*. CRC Press (cit. on pp. 12, 19).

Jensen, Finn and Thomas Nielsen (Jan. 2007). *Bayesian Network and Decision Graphs*. ISBN: 978-0-387-68281-5. DOI: 10.1007/978-0-387-68282-2 (cit. on p. 14).

Kamariotis, Antonios, Eleni Chatzi, and Daniel Straub (Mar. 2022). “Value of information from vibration-based structural health monitoring extracted via Bayesian model updating”. In: *Mechanical Systems and Signal Processing* 166, p. 108465. DOI: 10.1016/j.ymssp.2021.108465 (cit. on p. 12).

Kamariotis, Antonios, Konstantinos Tatsis, Eleni Chatzi, Kai Goebel, and Daniel Straub (2024a). “A metric for assessing and optimizing data-driven prognostic algorithms for predictive maintenance”. In: *Reliability Engineering & System Safety* 242, p. 109723 (cit. on pp. 10, 14).

– (2024b). “A metric for assessing and optimizing data-driven prognostic algorithms for predictive maintenance”. In: *Reliability Engineering & System Safety* 242, p. 109723 (cit. on pp. 21, 46, 52, 53).

Kim, Nam-Ho, Dawn An, and Joo-Ho Choi (2017). “Prognostics and health management of engineering systems”. In: *Switzerland: Springer International Publishing* (cit. on p. 12).

Kim, Seokgoo, Joo-Ho Choi, and Nam Ho Kim (2022). “Inspection schedule for prognostics with uncertainty management”. In: *Reliability Engineering & System Safety* 222, p. 108391 (cit. on p. 12).

Kochenderfer, Mykel J (2015). *Decision making under uncertainty: theory and application*. MIT press (cit. on p. 14).

Lei, Yaguo, Naipeng Li, Liang Guo, Ningbo Li, Tao Yan, and Jing Lin (2018). “Machinery health prognostics: A systematic review from data acquisition to RUL prediction”. In: *Mechanical systems and signal processing* 104, pp. 799–834 (cit. on p. 12).

Li, Xiang, Wei Zhang, and Qian Ding (2019). “Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction”. In: *Reliability engineering & system safety* 182, pp. 208–218 (cit. on p. 12).

Luque, Jesus and Daniel Straub (2019). “Risk-based optimal inspection strategies for structural systems using dynamic Bayesian networks”. In: *Structural Safety* 76, pp. 68–80 (cit. on p. 12).

Nectoux, Patrick et al. (2012). “PRONOSTIA: An experimental platform for bearings accelerated degradation tests.” In: *IEEE International Conference on Prognostics and Health Management, PHM'12.* IEEE Catalog Number: CPF12PHM-CDR, pp. 1–8 (cit. on p. 12).

Nguyen, Khanh TP and Kamal Medjaher (2019). “A new dynamic predictive maintenance framework using deep learning for failure prognostics”. In: *Reliability Engineering & System Safety* 188, pp. 251–262 (cit. on p. 12).

Nguyen, Khanh TP, Kamal Medjaher, and Christian Gogu (2022). “Probabilistic deep learning methodology for uncertainty quantification of remaining useful lifetime of multi-component systems”. In: *Reliability Engineering & System Safety* 222, p. 108383 (cit. on p. 12).

Paris, Paul C and F Erdogan (1997). “A critical analysis of crack propagation laws”. In: (cit. on p. 12).

Pater, Ingeborg de, Arthur Reijns, and Mihaela Mitici (2022). “Alarm-based predictive maintenance scheduling for aircraft engines with imperfect Remaining Useful Life prognostics”. In: *Reliability Engineering & System Safety* 221, p. 108341 (cit. on p. 12).

Purohit, Harsh et al. (2019). “MIMII Dataset: Sound dataset for malfunctioning industrial machine investigation and inspection”. In: *arXiv preprint arXiv:1909.09347* (cit. on p. 12).

Saha, B (2007). “Battery Data Set”. In: *NASA AMES Progmostics Data Repository* (cit. on p. 12).

Saxena, Abhinav and Kai Goebel (2008). “Turbofan engine degradation simulation data set”. In: *NASA ames prognostics data repository* 18, pp. 878–887 (cit. on p. 12).

Saxena, Abhinav, Kai Goebel, Don Simon, and Neil Eklund (Oct. 2008). “Damage propagation modeling for aircraft engine run-to-failure simulation”. In: *International Conference on Prognostics and Health Management*. DOI: 10.1109/PHM.2008.4711414 (cit. on pp. 11, 12).

Serradilla, Oscar, Ekhi Zugasti, Jon Rodriguez, and Urko Zurutuza (2022). “Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects”. In: *Applied Intelligence* 52.10, pp. 10934–10964 (cit. on pp. 53, 54).

The Illustrated transformer @JayAlammar (n.d.). https://jalammar.github.io/illustrated-transformer/. [Online; accessed 1-12-2019] (cit. on pp. 26–32).

Tijms, Henk (Dec. 2004). “A First Course in Stochastic Models”. In: DOI: 10.1002/047001363x.ch2 (cit. on p. 14).

Vaswani, A (2017). “Attention is all you need”. In: *Advances in Neural Information Processing Systems* (cit. on pp. 24, 25).

Zeng, Junqi and Zhenglin Liang (Jan. 2022). “A Deep Gaussian Process Approach for Predictive Maintenance”. In: *IEEE Transactions on Reliability* PP, pp. 1–18. DOI: 10.1109/TR.2022.3199924 (cit. on p. 20).

Zhuang, Liangliang, Ancha Xu, and Xiao-Lin Wang (Feb. 2023). “A prognostic driven predictive maintenance framework based on Bayesian deep learning”. In: *Reliability Engineering System Safety* 234, p. 109181. DOI: 10.1016/j.ress.2023.109181 (cit. on p. 12).