ESTIMATING TOOL WEAR USING MULTI-SENSOR DATA FUSION AND MACHINE LEARNING TECHNIQUES.

by

Tanner Owen Jones

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Industrial and Management Systems Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

December 2024

DEDICATION

I would like to dedicate this thesis to my family Stephanie Jones, Christopher Jones, Hunter Jones, and Blue Jones. Thank you for your continuous care and support throughout my time here at Montana State University. You all helped me become who I am today, this world would be an empty world without you four!

ACKNOWLEDGEMENTS

First, I would like to express my gratitude to my advisor, Yang Cao. I appreciate Professor Cao for giving me a chance and taking me under his wing. This research project Professor Cao introduced to me helped me grow as a student, academically and professionally. Because of this project, I have obtained knowledge and skills that give me great confidence to apply what I have learned in the real world. Professor Cao was always there when needed, and he never let me down. I would also like to express my gratitude to the Montana Space Grant Consortium for providing me with the Montana Space Grant Fellowship. Their generous support has been instrumental in providing motivation and support during the late stages of my research. I am deeply appreciative of their commitment to fostering educational and research opportunities in science and engineering fields. I would like to thank the senior capstone team for setting up the necessary equipment and software in the manufacturing cell, and for overall making this project possible. All members of the capstone team provided me with the necessary knowledge and skills to operate the CNC machine. Finally, I would like to thank all my professors that had me as a graduate and undergraduate student. The professors provided me with the necessary tools and skills to be a successful Graduate Research Assistant.

TABLE OF CONTENTS

TABLE OF CONTENTS CONTINUED

TABLE OF CONTENTS CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES CONTINUED

Figure Page

LIST OF CODE BLOCKS

Code Block Page

GLOSSARY

AE	Acoustic Emission
ANN	Artificial Neural Network
BR	Bagging Regressor
CNN	Convolutional Neural Network
CNC	Computer Numerical Control
CSV	Comma-Separated Values
DAQ	Data Acquisition
DOE	Design of Experiments
FFT	Fast Fourier Transform
GMDH	Group Method of Data Handling
HMM	Hidden Markov Model
ISSA	Improved Sparrow Search Algorithm
LSBoost	Least Squares Boosting
MLR	Multiple Linear Regression
MSDF	Multi-Sensor Data Fusion
MSE	Mean Squared Error
PCA	Principal Component Analysis
PHM	Prognostics and Health Management
R^2	Coefficient of Determination
ReLU	Rectified Linear Unit
RF	Random Forest

GLOSSARY CONTINUED

RMSE	Root Mean Squared Error
RTSCNN	Reshaped Time Series Convolutional Neural Network
SDA	Spiral Dynamic Algorithm
SVR	Support Vector Regression
TCM	Tool Condition Monitoring
XGBoost	Extreme Gradient Boosting

ABSTRACT

Modern manufacturing industries are being transformed by the integration of sensor technology, data science, and machine learning, leading to smarter, more efficient operations. Advancements in equipment health monitoring are crucial for improving productivity, extending equipment lifespan, and ensuring consistent product quality. In computer numerical control (CNC) machining, worn tools contribute to increased forces and vibrations, negatively impacting both machine performance and part quality. Traditional tool condition monitoring methods, which rely on manual offline inspections, result in machine downtime and decreased productivity. Modern tool condition monitoring methods involve monitoring tools based on single-sensor analysis. While a single sensor can detect tool wear within a machine, it fails to capture the full range of system behavior, potentially overlooking critical anomalies indictive of tool wear. To address these challenges, automated monitoring systems utilizing multisensory data and machine learning techniques have been developed, enabling real-time monitoring and prediction of tool wear. This research introduces a novel three-level data fusion framework for predicting tool flank wear in CNC machining. Force, vibration, and sound data was collected using various sensors during a CNC milling operation. The raw sensor data was processed and transformed into distinct statistical features to train machine learning models. A stacking ensemble method combining a random forest, artificial neural network, and extreme gradient boosting algorithm was employed to enhance predictive accuracy, achieving an R^2 value of 0.982, and root mean squared error of 37.146 micrometers. The proposed three-level fusion framework proved to be highly effective in predicting tool flank wear and shows great potential for monitoring the health of engineering equipment across a variety of industries.

CHAPTER ONE

INTRODUCTION

Predictive Maintenance

The increasing integration of sensor technology, data science, and machine learning is driving advancements across a variety of industries, transforming operations into smarter, more reliable, and highly efficient systems. A vital application of these technologies is in equipment health monitoring, where data-driven, real-time monitoring systems are employed to reduce downtime, extend equipment lifespan, and ensure product quality. This practice is known as predictive maintenance, a proactive approach that uses data analysis and monitoring tools to predict the health of equipment. This is particularly crucial in CNC machining, where maintenance emerges as a key solution to these challenges, offering significant improvements in operational efficiency and product quality. Predictive maintenance continues to become a more important practice for modern industries, because it transforms maintenance strategies from reactive to proactive, providing multiple financial, quality, and operational-related benefits.

The Role of Sensors in CNC Machining

In CNC machining, the physical condition of a cutting tool is directly linked to the forces, vibrations, and acoustic emissions generated during the cutting process. As tools wear, these physical signals increase, heightening the risk of part defects and potential damage to machine components. As a work piece is machined, the cutting tool gradually becomes dull due to the friction between the tool and the part. A dull tool requires greater cutting forces to machine the same workpiece compared to a sharp tool. This means that both static and dynamic cutting forces

increase as tool wear progresses, with cutting force serving as a key indicator of tool wear during the machining process. Vibration is caused by the interaction of machine components and the machining process, while chatter arises from the regeneration of surface waviness due to the contact between the workpiece and the tool at specific spindle frequencies (Teti et al., 2010). The vibration signal provides insight into a tool's condition, which increases as tool wear increases. Acoustic emission refers to the spontaneous release of transient elastic energy in a material undergoing deformation, fracture, or both. The energy released is closely related to the deformation rate and the volume of the material. The primary source of acoustic emission in machining is the sliding friction between the cutter and the workpiece (Siddhpura and Paurobally, 2013).

Monitoring these phenomena with various sensors that indicate tool wear help maintain the integrity of the production process. The monetization of tool wear is known as tool condition monitoring (TCM) and is crucial for ensuring product quality during machining operations. As tools wear, the sharpness on the cutting edges begin to become dull, or chip, which can directly impact the surface finish or dimensional accuracy of machined parts. Another phenomenon that can happen is complete failure of a cutting tool. Failure of cutting tools can lead to unexpected machine downtime or expensive repairs. The failure of machine tools can contribute up to 20% of machine downtime on a production floor (Zhang et al., 2016). To address these issues, the integration of predictive maintenance practices for TCM is essential for improving operational efficiency, reducing costs, maintaining product quality, and extending the lifespan of both tools and machinery.

Limitations of Traditional TCM Methods

Traditional methods for monitoring tool condition often involve halting operations for visual inspections. This time-consuming process relies heavily on the expertise of operators, who must assess tool wear based on experience and subjective judgement. Such interruptions not only decrease productivity, but also introduce variability in monitoring effectiveness, as different operators may have different thresholds for tool wear. Moreover, these inspections can lead to unscheduled downtime, which can result in financial loss and decreased competitiveness.

Other methods in modern manufacturing environments involve monitoring tools based on single-sensor analysis. Some examples of this are monitoring triaxial forces and spindle power. While a single sensor can detect tool wear within a machine or system, it does not capture the full range of system behavior, potentially overlooking critical anomalies that indicate tool wear. Tool wear is influenced by multiple interacting factors, and a single sensor alone cannot capture these complex interactions.

Advancements in Automated Monitoring Systems

To address the limitations of traditional TCM methods, automated monitoring systems have been developed to enable continuous tool wear assessment, without the need for interruptions or manual inspections. These systems utilize a combination of advanced sensors, machine learning algorithms, and data analytics to capture real-time data on cutting forces, vibrations, and acoustic emissions. By continuously analyzing this data, these automated monitoring systems can detect patterns indicative of tool wear, providing valuable insights into a tool’s condition.

Predictive Maintenance Benefits

Predictive maintenance leverages the data collected by these automated systems to forecast when a tool is likely to fail or require replacement. Machine learning models can analyze historical and real-time data to identify trends and anomalies, allowing manufacturers to make better informed decisions about maintenance schedules. This proactive approach minimizes unexpected breakdowns, reduces maintenance costs, and optimizes production schedules. Some of the benefits of these enhanced methods include:

1) Reduced downtime – by predicting failures before failures occur, manufacturers can schedule maintenance during non-productive times, thereby minimizing disruptions to production.
2) Extended equipment lifespan – continuous monitoring allows for timely interventions that can prolong the life of tools and machinery, resulting in lower replacement costs.
3) Improved product quality – by maintaining optimal tool performance, manufacturers can ensure consistent part quality, reducing reworks and defects.
4) Data-driven decision making – The insights gained from data analytics empower operators and managers to make informed decisions based on real-time data rather than experience.

The Importance of Tool Condition Monitoring

The integration of sensor technology, data science, and machine learning in CNC machining is transforming how manufacturers approach TCM. By shifting from reactive maintenance to predictive maintenance, companies can enhance operational efficiency, improve product quality, and reduce costs. As these technologies continue to evolve, the potential for

further advancements in equipment health monitoring will undoubtably reshape the landscape of manufacturing, and the landscape of a variety of industries.

Prognostics and Health Management

These automated predictive maintenance systems are central to the field of Prognostics and Health Management (PHM), which focuses on predicting the future health of systems and equipment. TCM is a subcategory of PHM, specifically focused on predicting the health of manufacturing machinery tools. PHM approaches are generally divided into two categories: (1) model-based and (2) data-driven.

Model-Based Prognostics

Model-based prognostics are based on a mathematical description of a system. These models use system dynamics, physical laws, or statistical approaches to simulate how components degrade and predict when failure may occur. Some previous practices of model-based prognostics in TCM include using Hidden Markov Models (Dong and He, 2007), Kalman filters (Niaki et al., 2016), Wiener processes (Si et al., 2013), and Taylor's Tool Life equation (Lau et al., 1980). For model-based prognostics to be an effective method for monitoring the health of tools or equipment, a deep understanding of the system is required, which can be difficult in many circumstances because previous knowledge on the behavior of complex manufacturing systems is not usually known.

Data-Driven Prognostics

Data-driven prognostics offer an alternative, leveraging machine learning algorithms and sensor data from equipment to create predictive models. This approach does not require prior

knowledge of the system, allowing for the development of adaptable models that learn patterns directly from operational data. Machine learning is a subset of artificial intelligence, with the goal of giving computers the ability to perform data analysis tasks without explicit instructions. Machine learning is utilized in a variety of industries and domains and will continue to become more embedded in society as time goes on. Specifically, in TCM, common machine learning models used include multiple linear regression (MLR) (Ozel and Karpat, 2005), support vector regression (SVR) (Wu et al., 2017), artificial neural networks (ANN) (Chen and Chen, 2004), and many others. These models are trained on features such as machine operating parameters, or data obtained from sensors which characterize the behavior of a machine, with the model's output being an estimation for a desired metric, such as tool wear. The strength of data-driven models lies in their ability to manage the complexity and variability inherent in real-world systems, particularly in scenarios where the system's behavior is poorly understood. This ability to make accurate predictions empowers industries to transition from reactive maintenance to data-driven, proactive maintenance strategies. As PHM systems continue to evolve, these systems are expected to play a vital role in achieving fully autonomous environments, particularly in manufacturing environments where machinery health can be managed with minimal human intervention.

Thesis Overview

Multi-Sensor Data Fusion

Multi-sensor data fusion (MSDF) involves extracting data from multiple sources or sensors to obtain more comprehensive information of a system. Hall and Llinas (1997) characterized data fusion as a multi-level technique consisted of three levels: (1) raw data fusion, which aims to provide comprehensive data for the behavior of an environment; (2) feature-level fusion, which is

the transformation of raw data in order to provide more relevant or focused information; and (3) decision-level fusion, which aims to combine the predictions from multiple models in order to produce a more robust, accurate final prediction. MSDF plays a vital role in gaining a deeper understanding of systems or environments, because it aims to integrate and synthesize data from multiple sensors or sources, providing a more comprehensive and accurate representation of the system's state or behavior than any single source could achieve.

Research Objective

The aim of this research is to develop a data-driven model capable of accurately estimating tool flank wear during a CNC milling facing operation, utilizing data capturing force, vibration and sound signals obtained from multiple sensors during cutting. Typically, previous research in TCM has utilized raw data fusion and feature-level fusion to build predictive models. The main contribution of this research is the introduction of a three-level fusion approach (Figure 1):

1) Raw data fusion – a load cell, accelerometer, and an acoustic emission sensor were used to collect cutting forces, vibrations, and sound signals during a CNC milling operation to provide comprehensive information about the machine's behavior.
2) Feature-level fusion – several important statistical features were extracted from the raw data to provide more focused information for the machine learning model training process.
3) Decision-level fusion – a stacking ensemble is employed to fuse the predictions from various machine learning models. This ensemble consists of two base models, each trained on the selected statistical features and measured tool flank wear values, and a higher-level meta-model that synthesizes the predictions from the base models. By

training the meta-model on these base model predictions, the ensemble achieves a more robust, refined, and accurate tool wear prediction.

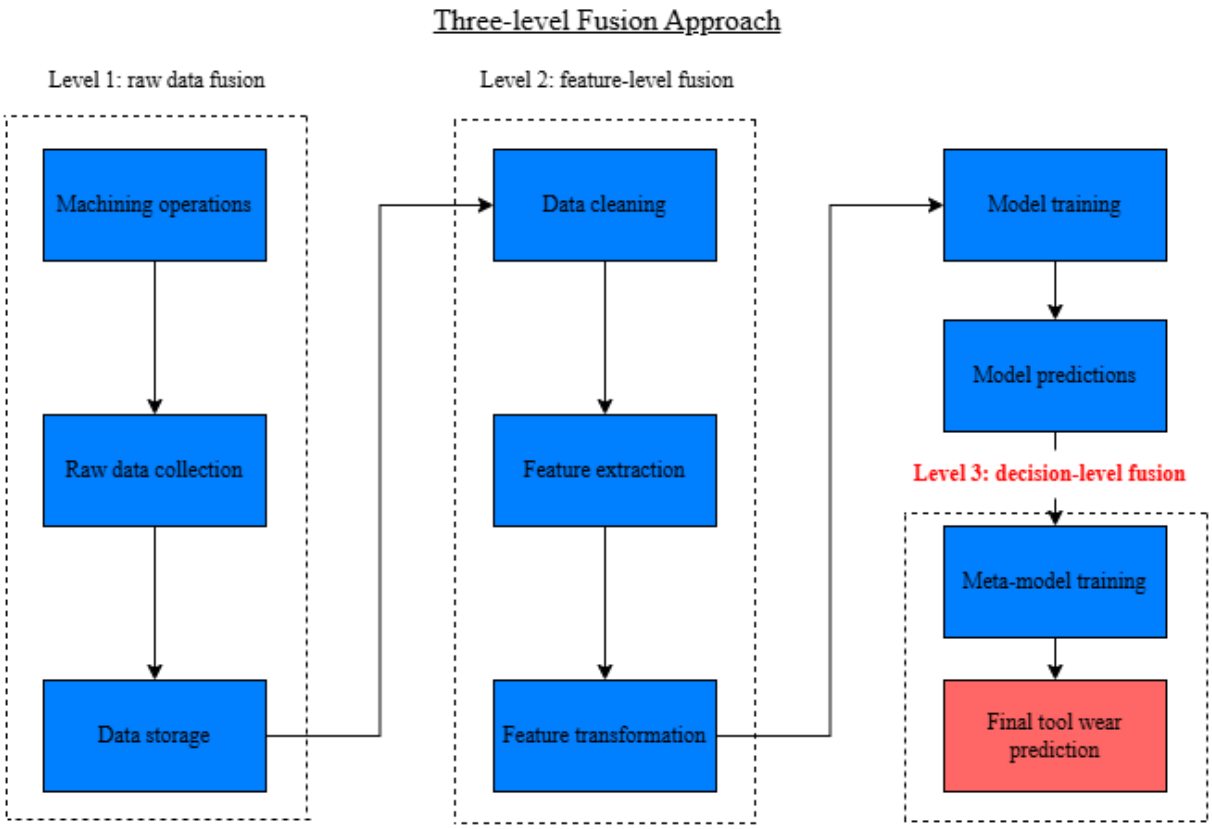


Figure 1. Flowchart of three-level fusion approach

Thesis Framework

Chapter Two will be a literature review on signal acquisition, multi-sensor data fusion, and machine learning techniques that have previously been applied in TCM research. Chapter Three covers the experimental phase of this research, specifically the experimental procedures, instruments, software, sensors, data collection, and data processing. Chapter Four is dedicated to machine learning, particularly the algorithm selection process, construction process, and the exploration of key components and how each algorithm functions. Chapter Five will cover the

results, the performance of each developed model, as well as optimal hyperparameters. Finally, Chapter Six will discuss the outcomes, limitations, and future applications of the research. The sequence of this paper is illustrated below in Figure 2.

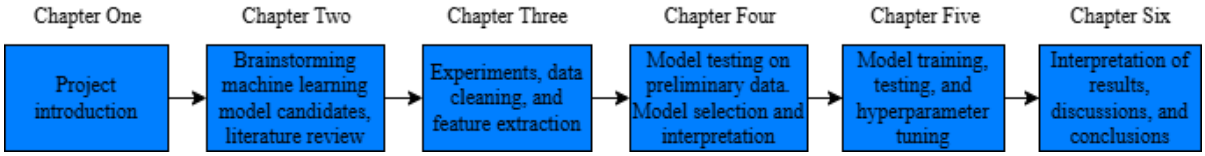


Figure 2. Flowchart illustrating sequence of thesis and research project

CHAPTER TWO

LITERATURE REVIEW

Signal Acquisition

For data-driven prognostics to be effective in PHM, detailed data which characterizes a system's operating conditions and behavior is essential. Previous research in data-driven tool wear prediction has focused on using machine operating parameters to characterize a machine's behavior. For instance, Rathod et al. (2022) investigated CNC milling tool life using machine operating parameters such as depth of cut, cutting speed, and feed rate. While methods based on machine operating parameters can be effective in certain circumstances, these methods fail to capture the real dynamic behavior of a system. One evolving method that characterizes the dynamic behavior of a system is signal acquisition. Signal acquisition refers to capturing a physical signal (such as force, power, or electric current) from sensors or instruments, and converting those signals into a data product that can be analyzed or stored. Signal acquisition is a popular practice in the field of TCM with many researchers utilizing sensors measuring force, acceleration, sound, power, current, and even torque.

Tool Condition Monitoring with Force Signals

Force signal acquisition is highly common in the field of TCM. In various studies, force is usually measured with a dynamometer or a load cell. For instance, Lee and Targng (1999) utilized cutting force signals in combination with advanced signal processing techniques such as the discrete wavelet transform. This wavelet transform allows for multi-level signal decomposition, enabling the extraction of distinctive features related to cutter breakage. Experimental results

further confirm that this approach can successfully detect mulling cutter breakage across a range of operational conditions. These findings contribute to the growing body of literature on using force sensors for TCM.

Bhattacharyya et al. (2007) presented a method for real-time tool wear estimation in face milling using cutting force signals. Several signal processing techniques were combined, including linear filtering, time-domain averaging, and wavelet transformation, to extract key features from the cutting force data that indicate a worn tool. The researchers found these methods to be effective in estimating how worn a cutting tool is. This research confirms that collecting force signals during milling operations is a highly effective practice in TCM.

Tool Condition Monitoring with Vibration Signals

Vibration signals play a critical role in the development of effective monitoring systems for TCM. For instance, Chen and Chen (1999) developed an in-process monitoring system utilizing accelerometer-based vibration signals during an end mill operation to detect tool breakage. The captured signals were transformed into a frequency domain using a Fast Fourier transform (FFT) to isolate breakage indicators. Thresholds were established to differentiate between tool states across various experimental conditions. The developed method showed success in achieving tool breakage detection, highlighting the effectiveness of utilizing vibration signals to characterize the in-system behavior of a machine during operations.

In another study, Yesilyurt and Ozturk (2006) demonstrated the use of vibration signal analysis in a milling process to detect and track damage in cutting tools. Cutting experiments on mild steel were performed, and vibration signals were analyzed both in the time and frequency domains to extract signals indicative of tool wear. A scalogram analysis and mean frequency

variation were used to detect localized and progressive wear patterns. The mean frequency variation provided a consistent and reliable indicator of wear progression. This study underscores the effectiveness of vibration signal analysis, making it a valuable approach for TCM.

Tool Condition Monitoring with Acoustic Emission

Acoustic emission signals have emerged as a valuable tool for TCM. Acoustic emission signals provide insights into tool condition by capturing the differences between the rubbing friction on the tool's wear land and dislocation activities in the shear zones. Liang and Dornfeld (1989) introduced a time series analysis approach for acoustic emission signals, using an auto aggressive modeling scheme to capture wear-related signal characteristics during cutting. By encoding acoustic emission signal features into a time-varying model, this method enables robust tool wear detection. This study highlights the potential of sound signal-based monitoring systems in detecting and tracking tool wear.

In a separate study, Atlas et al. (2002) highlighted acoustic vibrations offering valuable insights into tool wear. The researchers looked at detectable changes in acoustic signal patterns to determine wear characteristics. A HMM was used to track the evolution of acoustic emission signals over time in a milling process. By examining signals across three-time scales, the developed HMM was shown to accurately predict wear progression. This application further underscores the potential of acoustic emission-based models for precise, adaptive wear monitoring.

Tool Condition Monitoring with Multiple Sensors

While the single-sensor practice to detect wear has shown to be effective in recent literature, a single-sensor based approach still fails to fully capture the dynamic behavior of a system or machine. To address this limitation, MSDF techniques are employed to integrate data

from multiple sensors, providing a more accurate, comprehensive understanding of system behavior. Recent advances in signal processing have enabled the widespread use of online sensors to continuously monitor tool conditions. MSDF techniques combine data from multiple sensors, or related information from data to achieve improved accuracies than those of which could be achieved by a single sensor alone (Hall and Llinas, 1997). Humans and animals use multiple senses to gain a better understanding of the surrounding world. For instance, it might not be possible to assess the quality of an edible substance with taste alone, but evaluation of the quality of an edible substance can be greater achieved using a combination of senses such as sight, touch, smell, and taste. The combination of multiple senses gives humans and animals a greater understanding of the surrounding world, ultimately helping with survival. In a similar manner, MSDF provides a more comprehensive understanding of complex systems, such as machinery health in manufacturing. MSDF has been applied in numerous studies to investigate CNC machine tool wear, utilizing a range of sensors including but not limited to force sensors, vibration sensors, acoustic emission sensors, and power sensors. The combination of these sensors contributes to more accurate tool wear predictions by combining complementary and redundant information, thereby enhancing the reliability of the results.

MSDF approaches have become essential for monitoring tool wear in the TCM field, as the combination of multiple sensors leverages diverse signal signatures to enhance the accuracy and reliability of tool condition assessment. Haber et al. (2004) investigated the integration of dynamometers, accelerometers, and acoustic emission sensors to detect tool wear during a high-speed machining process. The respective signals were analyzed in both the time and frequency domains. By comparing and analyzing the deviations in these signals, the findings showed cutting

force and vibration signals being significant indicators of tool wear. Additionally, the findings showed acoustic emission signals at certain threshold frequencies being an indicator of tool wear. This study highlighted the effectiveness of complementary information produced by multiple sensors to investigate tool wear.

Boud and Gindy (2008) evaluated multiple sensor outputs, including acoustic emission, cutting force, vibration, hydraulic pressure, and table displacement during a broaching operation to monitor tool condition and surface anomalies on parts. The results demonstrated that these multi-sensor signals effectively detected surface deviations and anomalies. The results also showed that each sensor contributed to valuable insights. Cutting force, pressure, and table displacement signals were especially effective in identifying a worn tool, while acoustic emission signals were highly effective in identifying surface anomalies. This research highlights the value of integrating multiple sensor types to provide a comprehensive view of machining conditions, particularly with each sensor providing unique insights.

Bagga et al. (2021) utilized a multi-sensor setup on a lathe during the machining of AISI 4140, applying data fusion from sensors monitoring vibration, power, temperature, force, and surface roughness to predict flank wear. Using a factorial design and the Taguchi Method (Karna and Sahai, 2012), the study also evaluated how machining parameters influence tool wear. An ANN was developed to predict tool wear, with results closely aligning with offline measurements. This research further highlights the advantages of the utilization of multiple sensors to characterize the behavior of a machine, particularly to estimate the state of a tool.

Wang et al. (2024) investigated tool wear using data gathered by force, vibration, and acoustic emission sensors. A denoise transformer was proposed as a preprocessing tool for TCM

classifiers. The developed model improved the feature extraction by emphasizing intrinsic wear characteristics and selecting appropriate features from multi-sensor signals, reducing the influence of noise. The developed model achieved high accuracy in classifying a worn tool. This research demonstrated the use of multiple sensors as a robust approach for classifying worn tools.

Huang et al. (2019) utilized force, acceleration, and sound signals to predict tool wear during a milling operation. The research proposed a new multisensory data-driven tool wear predicting method based on a reshaped time series convolutional neural network (RTSCNN). The reshaped time series layer in the neural network was used to represent the raw signals from the sensors. The convolutional and pooling layers of the developed model were employed to adaptively learn distinct characteristics of tool wear directly from raw signals. The results showed that the RTSCNN-based signal data-driven method produced very accurate results. This research highlighted the effectiveness of using data from multiple sensors and machine learning for TCM.

As demonstrated by previous research, MSDF plays a crucial role in advancing TCM and PHM by providing real-time, dynamic data that characterizes the behavior of systems. This integration of sensor data is critical to improve the predictive accuracy of data-driven models in manufacturing systems.

Machine Learning Algorithms

This section of the chapter covers the various machine learning algorithms that have been used as predictive models in the TCM research domain, as well ensemble methods utilized in other fields of research. Some of the common algorithms include ANNs, SVR, MLR, random forests (RF), and many others. Machine learning algorithms and ensembles have the ability to learn and

model intricate relationships in data, making them especially powerful in TCM, where data is often complex.

Artificial Neural Networks

Schwabacher and Goebel (2007) provided a comprehensive review of commonly used data-driven methods in PHM, with ANNs being a widely adopted approach. ANNs are a class of computational models inspired by biological neural networks, designed to capture, and model complex relationships between inputs and outputs. In addition to their extensive use in PHM, ANNs have been applied for many practices, such as image and speech recognition, as well as financial forecasting.

ANNs are computational models consisting of multiple layers of interconnected neurons organized into one input layer, one or more hidden layers, and an output layer. Each neuron performs mathematical computations based on the data it receives, applying weights to inputs and passing the results through an activation function. As data moves from layer to layer, neurons extract progressively higher-level features, which enables ANNs to capture complex patters in data. The training process, typically achieved through backpropagation and optimization algorithms like gradient descent, iteratively adjusts the weights and biases of the network in response to errors between predicted and actual values. This iteration-based learning allows ANNs to adapt to non-linear and complex relationships within data. The high generalizing capabilities make ANNs highly effective in regression applications, meaning ANNs can be used to model complex system behaviors, anticipate failures, and enhance predictive maintenance systems. By learning from historical or real-time sensor data, ANNs can detect subtle changes on system performance, making this algorithm an attractive algorithm candidate for this research.

Several TCM studies have demonstrated the effectiveness of ANNs in predicting tool wear and other machine outcomes. Ozel and Karpat (2005) proposed a predictive model for tool flank wear and surface roughness in a hard turning process using an ANN. The model utilized input features such as workpiece hardness, cutting speed, force, axial cutting depth, and feed rate. To assess the performance of the ANN, it was compared with a MLR model. The trained ANN achieved high accuracy predicting surface roughness and tool wear, outperforming the MLR model. The research showed that ANNs provide better prediction capabilities than linear regression, especially when more complex non-linearities and interactions are present.

Chen and Chen (2004) developed an in-process tool wear prediction system using an ANN trained on 100 experimental data points obtained during CNC milling operations, with inputs including average peak cutting forces, feed rate and depth of cut. Chen and Chen (2004) used the collected data to train an ANN. The results showed that for the data, an ANN consisting of two hidden layers, each with eight nodes performed the best. On average, the model predicted tool wear with an error of just 0.037 mm. This low prediction error demonstrates the ability of ANNs, being a robust approach for tool wear prediction.

Similarly, Karayel (2009) designed an ANN model to predict and control surface roughness in a turning operation, using parameters such as depth of cut, cutting speed, and feed rate. The ANN was designed to predict various surface roughness metrics, such as Ra, Rz, and Rmax, with different hidden layer configurations depending on the metric. The results of the ANN predictions were compared to the offline measured values, and the ANN achieved high accuracy. Furthermore, the study developed a control algorithm that used the predicted values to adjust the cutting parameters in real time, to ensure that the surface roughness remained within desirable limits. This

study highlights the role of ANNs not only as a predictive model, but also as a promising approach to be integrated into an adaptive control system.

Huang et al. (2019) introduced a novel approach by employing a reshaped time series convolutional neural network (RTSCNN), where the reshaped time series layer represented raw signal data and the convolutional layers learned distinctive characteristics of tool wear adaptively. Convolutional neural networks (CNNs) are specialized types of networks, with the purpose of processing grid-like data. CNNs are widely used for complex data patterns, such as image and speech recognition. The RTSCNN developed by Huang et al. (2019) outperformed other developed models, such as a recurrent neural network (RNN) and support vector regression model (SVR). The RTSCNN was highly accurate, producing a root mean squared error of just 2.2 mm. The research showed ANNs, even with complex structures like a CNN, are powerful predictive models for regression tasks.

Palanisamy et al. (2007) carried out a study comparing the performance between two models when predicting tool wear, a MLR model and ANN model. The study was focused on a milling operation, where flank wear was measured offline as the response variable for each experiment carried out. Input parameters for the models were cutting speed, feed rate, and depth of cut. Experiments were conducted using a Design of Experiments (DOE) framework. The collected data was utilized to develop both the MLR and ANN models. The results showed that the ANN was found to outperform the MLR model when estimating tool wear. This study highlights the superior capabilities of ANNs in predicting tool wear compared to more simple linear models such as MLR.

Previous research has proven ANNs to be an effective ML model for data-driven prognostics. ANN's ability to capture complex system behaviors and patterns make ANNs a highly effective predictive model in PHM. The architecture of ANNs allows for the modeling of non-linear relationships between input features and outputs, which is particularly valuable in scenarios where traditional statistical methods might struggle. ANNs can learn from small or large datasets, adapting to new information and improving the predictive accuracy over time. This adaptation is crucial in dynamic environments, where system behavior is subject to change due to various factors. Moreover, ANNs can integrate multiple types of data inputs, including time series, categorical, and numerical data, making ANNs versatile tools for analyzing complex, dynamic systems. In summary, the robustness, adaptability, and the versatility of ANNs contribute to the strong performance in predictive maintenance applications, especially in TCM. The effectiveness of ANNs from the literature positions an ANN model as a strong algorithm candidate for this research.

Decision Trees

Decision trees are another widely used data-driven method in PHM. Decision trees are a versatile supervised learning technique applied to both classification and regression problems without assuming any data distribution. The goal of a decision tree is to create a model that predicts the value of a target variable by applying decision rules based on input features. Structurally, decision trees resemble a flowchart: internal nodes represent conditions or feature splits, branches indicate the outcomes of those tests, and leaf nodes correspond to the predicted class label or value.

Jia and Dornfeld (1998) utilized a decision tree method, along with the group method of data handling (GMDH), to predict tool wear during a turning process using acoustic emission and

cutting force signals. The decision tree model was able to generalize heuristic rules from learning samples, enabling reliable classifications of tool wear states. The results showed that the decision tree-based approach combined with GMDH, achieved tool wear predictions with an accuracy within 5% of the measured values. This study highlighted the effectiveness of decision trees when used as predictive models, especially when the problem involves classification.

Elangoven et al. (2011) explored the use of data-mining techniques alongside a decision tree to monitor tool condition, particularly focusing on vibration signals to detect hidden patterns of tool wear. Statistical features were derived from the vibration data, with principal component analysis (PCA) applied as an alternative feature transformation method. There is a good reason for applying PCA to transform features, being that trees can only make axis-parallel splits, whereas PCA can rotate onto axes that best capture variability. To enhance the robustness of the decision tree classifier, feature reduction was employed. The results showed that the combination of decision trees and feature reduction with appropriate classifiers offers an effective predictive model approach.

Random Forests

While decision trees excel at interpreting complex data patterns, they can be highly sensitive to slight variations in the training data, which can often lead to overfitting. To mitigate overfitting, ensemble methods have been developed that combine multiple decision trees to improve robustness and predictive performance. Two popular ensemble techniques for decision trees are RFs and extreme gradient boosting (XGBoost).

The RF, introduced by Leo Breiman (2001), constructs multiple decision trees using bootstrapped samples from the training data. Each tree independently predicts the outcome, and

the final prediction is the average of all the trees' outputs. This approach reduces the variance of the model and improves generalization. Breiman (2001) introduced the RF algorithm to address the limitations of decision trees. The key reasons for the development of random forests:

1) Overfitting reduction – individual decision trees can be prone to overfitting data, capturing noise or outliers in the training data. RF mitigates this by averaging multiple trees, which helps in the generalization to unseen data.
2) Improved accuracy – by combining predictions from multiple trees trained on different subsets of data and features, RF often achieves higher accuracy than single decision trees.
3) Robustness – RF is more robust to outliers and noise compared to individual decision trees, making RFs particularly useful in real-world applications where data might not be perfectly clean.
4) Feature importance – RFs provide insight into feature importance, helping users understand which variables contribute most to the model's predictions.
5) Versatility – RFs can be applied to both classification and regression tasks, making it a versatile tool for various data types and problem domains.

RFs models have previously been utilized in TCM but have not been widely used. For instance, Wu et al. (2017) applied a RF model to predict tool wear in CNC milling operations using force, acceleration and sound data obtained by sensors in a machine. The purpose of the study was to compare the performance of various algorithms, specifically an ANN, SVR, and RF. With 80% of the data used for training, the RF model achieved a mean squared error (MSE) of 8.195 micrometers and an R^2 (coefficient of determination) of 0.992, outperforming both ANN and SVR

models. The research also showed that with increased training size, the predictive performance of RFs improves. The researchers stated that this was the first time a RF was used for predicting tool wear in a milling process. The research Wu et al. (2017) conducted was an inspiration to include a RF algorithm in this research.

Lee et al. (2019) compared the performance of RF, XGBoost and SVR in predicting CNC tool wear. Using 48 statistical features as inputs for each model, the results showed that a RF model with 50 trees achieved an impressive R^2 value of 0.993. The research highlighted important hyperparameters for a RF, specifically the number of decision trees. As the number of trees increase, the model accuracy increased. However, the research also showed when the number of trees increase, the training and computational power also increases.

Previous research has established RFs as a powerful machine learning model for TCM. The ensemble nature of RFs, which combines multiple decision trees, allows RFs to effectively capture complex relationships and patterns within data, making them particularly suited for predictive modeling in PHM. RFs excel in reducing overfitting through the averaging mechanism, leading to improved generalization on unseen data. This robustness to noise and outliers enhances predictive accuracy. Additionally, RFs provides insights into feature importance, helping practitioners identify which variables are most influential. Furthermore, RFs can handle a mix of data types and are effective in both classification and regression tasks. In summary, the benefits of RFs prove this algorithm to be a key player in the advancements in predictive analytics in PHM.

Extreme Gradient Boosting

Another powerful ensemble learning method utilizing decision trees is extreme gradient boosting (XGBoost). XGBoost, developed by Chen and Guestrin (2016), works by sequentially

building an ensemble of decision trees, where each new tree aims to correct the errors made by the previous ones. This process incrementally minimizes the loss function by focusing on the residual errors from prior iterations. XGBoost was developed for the key reasons:

1) Speed and performance – XGBoost was designed to be faster than existing gradient boosting methods. By optimizing the computation of the boosting algorithm and using parallel processing, it significantly reduces the training time while maintaining high predictive accuracy.
2) Regularization – unlike traditional gradient boosting algorithms, XGBoost incorporates L_1 (LASSO) and L_2 (ridge) regularization, which helps prevent overfitting and improves the model's overall generalization.
3) Handling missing values – XGBoost has built in mechanisms for dealing with missing data, allowing the model to learn patterns even when some input features are inadequate.
4) Flexibility – the framework supports various objective functions, making it suitable for a wide range of tasks, including classification, regression, and ranking problems.
5) Feature importance - XGBoost provides insights into feature importance like RFs, allowing for better interpretation of which variables contribute most significantly to predictions.

XGBoost predictive model approaches have also been utilized in PHM, specifically for predicting tool wear. In a study by Lee et al. (2019), an XGBoost model with 500 trees achieved an R^2 value of 0.993 for predicting tool wear. The XGBoost model performance was compared with predictive performance of RF and SVR models. The XGBoost and RF models produced

similar results. However, the XGBoost model had twice the computational training time. The research also highlighted an important hyperparameter for XGBoost, the number of trees. As the number of trees increase, the XGBoost model became more accurate.

Chen et al. (2024) integrated an Improved Sparrow Search Algorithm (ISSA) with XGBoost to optimize hyperparameters and predict tool wear states using force and torque signals obtained from a milling machine. An ISSA is an optimization algorithm inspired by the foraging behavior of sparrows, particularly a sparrow's ability to search for food efficiently while avoiding predators. The combination of ISSA and XGBoost yielded a classification accuracy of 93.62%. This research showed that XGBoost applied sequentially with optimization techniques like the ISSA is an effective predictive modeling approach for predicting the health of cutting tools.

Alajmi and Almeshal (2020) also applied a novel hybrid machine learning approach to predict tool wear during a drilling operation, specifically using XGBoost with a spiral dynamic optimization algorithm (SDA). The SDA is a nature-inspired optimization method that simulates the spiral behavior observed in certain natural processes. The SDA was used to find optimal hyperparameters for an XGBoost model. Alajmi and Almeshal (2020) compared the performance of the XGBoost-SDA algorithm to an ANN and support vector machine (SVM). The XGBoost model outperformed both SVM and ANN models, achieving a mean absolute error (MAE) of just 4.67%. The research conducted by Alajmi and Almeshal (2020) highlighted the importance of hyperparameter tuning with XGBoost, it proved that when optimal hyperparameters are used, XGBoost works as a powerful predictive model.

Previous research in TCM has demonstrated that XGBoost is a highly effective machine learning model for data-driven prognostics. Its ability to leverage gradient boosting techniques

allows XGBoost to capture complex relationships within data, making it particularly effective for predictive modeling in TCM. It stands out for its speed and efficiency. Additionally, XGBoost's capacity to handle missing values makes it particularly attractive, given that noise and missing values can be frequent in complex time-series data. Moreover, XGBoost's flexibility allows it to be applied to various tasks, including regression.

Model Stacking

Machine learning models such as ANNs, RFs and XGBoost have been proven to be highly effective predictive models with complex datasets. To further enhance the performance and predictive accuracy, ensemble methods are widely utilized in machine learning. These methods combine multiple models to capture diverse patterns in data and mitigate individual model limitations. One popular approach is model stacking, which involves training multiple base models and then using a meta-model to aggregate their predictions.

In stacking, the base models such as ANNs, RFs and XGBoost generate initial predictions on the data, while the meta-model, often a simpler model like linear regression or a more robust model like XGBoost, learns from the predictions of the base models to make a final more accurate prediction. This hierarchical structure allows stacking to leverage the strengths of different algorithms, reducing errors by correcting weaknesses in individual models, ultimately improving predictive accuracy. Stacking was explored in this research, ultimately to determine if it is an effective approach to achieve fusion at the decision level (Hall and Llinas, 1999), which is the core of what makes this research unique.

While model stacking has not been used in TCM research, it has been utilized in many other fields for regressive predictions. There are various reasons why stacking has not been used

in the TCM domain. For instance, many CNC studies use small datasets, which may not support the complexity of stacking models and the unique training process required, as these ensembles require sufficient amounts of data to prevent overfitting. Morshed-Bozorgdel et al. (2020) utilized a stacking ensemble model with eleven base machine learning models and one least squares boost (LSBoost) model as a meta-learner to model wind speed at sixteen different wind farms in Iran. On average, the stacked model was able to model wind speed from the 16 farms with an R^2 value of 0.86, which was on average 43% higher than individual base-model performance. Morshed-Bozorgdel et al. (2020) highlighted stacking's ability to synthesize diverse model predictions into a fused, more accurate prediction. It also demonstrated stacking's effectiveness when handling complex data, like wind speed, which is incredibly hard to model.

Akbari et al. (2023) used a stacking approach with XGBoost and RF as base-learners, and a bagging regressor (BR) as a meta-learner to predict electronic stopping power of charged particles. The developed stacking model was shown to be highly accurate, producing an R^2 value of 0.9955. This level of accuracy highlights the effectiveness of stacking in leveraging the strengths of multiple algorithms. Akbari et al. (2023) ultimately showed the power of stacking when multiple machine learning algorithms are included, which was part of the inspiration for using stacking as an ensemble method in this research.

As proven by previous research, the stacking ensemble approach with machine learning models is highly effective in producing more accurate predictions in regression problems, proving it to be an effective approach in TCM, specifically predicting tool wear during a CNC milling facing operation. By leveraging multiple models, each excelling in different aspects of a dataset, stacking can capture a wide range of patterns and interactions. Traditional single-model

approaches may struggle to fully capture nuances in data. Stacking allows base models to individually focus on different relationships within data. Given the need for precise, real-time predictions in TCM, the stacking ensemble offers a promising solution that could help drive advancements in tool wear prediction and other prognostic tasks.

CHAPTER THREE

METHODS

Experimental Setup

Machine and Sensors

The machine utilized in this research is a three-axis CNC milling machine (EMCO Concept Mill 55), equipped with a comprehensive multi-sensor setup to monitor machine behavior. To measure cutting forces in the X, Y, and Z directions, a three-axis load cell (ATO-LC-MA02) was mounted between the machine table and the clamping vise, providing high-resolution force measurements. Additionally, a three-axis accelerometer (Kistler 8763B050BT01) was attached to the machine head to monitor cutting vibrations in the X, Y, and Z directions. An acoustic emission sensor (Kistler 8152C003001) was positioned on a plate adjacent to the clamping vise to capture sound waves during cutting operations. Originally, a three-axis dynamometer (Kistler 9265B) was going to be utilized to measure force. Due to some calibration issues, the dynamometer was replaced with a three-axis load cell (ATO-LC-MA02). The load cell was calibrated with known forces and voltages. A picture of the machine setup is shown in Figure 3.

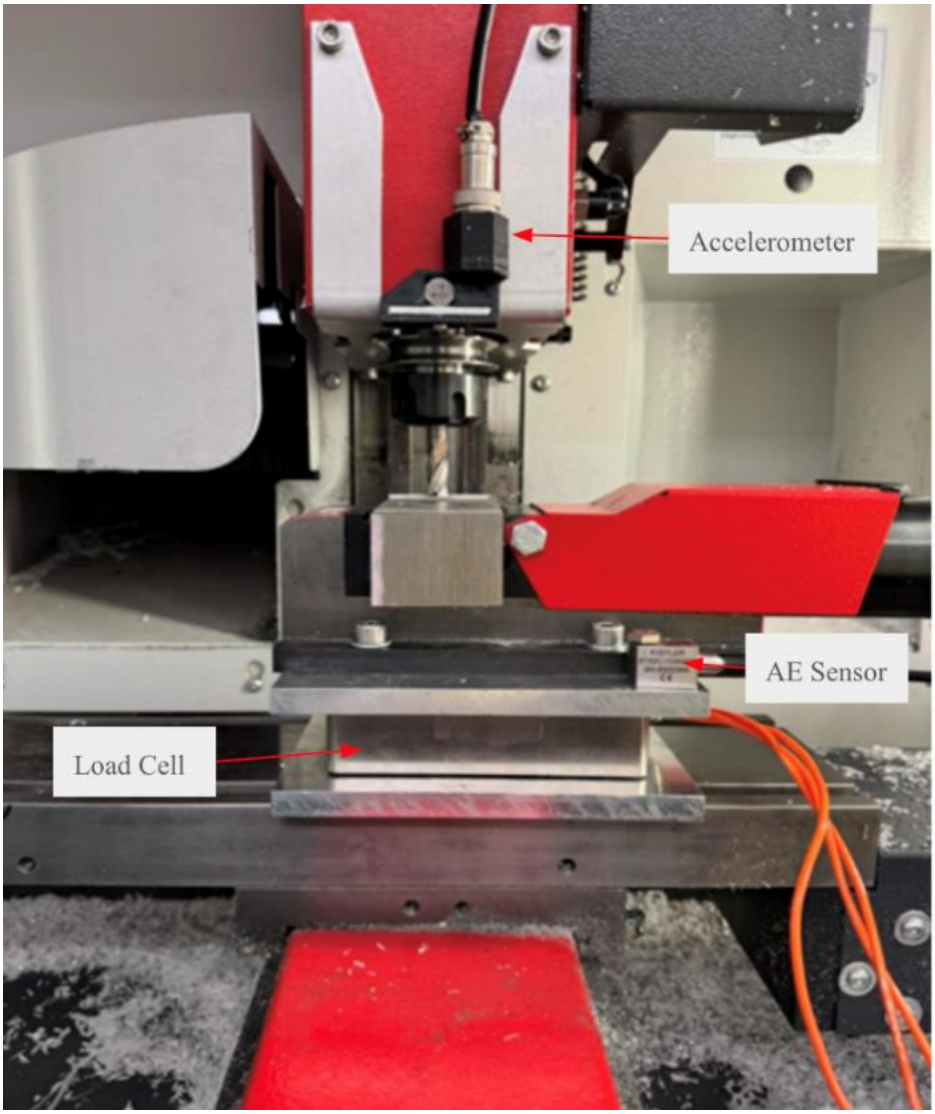


Figure 3. Machine setup

Data Acquisition

Data acquisition was where raw data fusion was achieved. Data acquisition was facilitated by two DAQ devices connected to a computer via a USB cable for a seamless data transfer. The acoustic emission sensor processed sound signals through an AE coupler (Kistler 5125C2) before transmitting data to DAQ 1 (MCC DT9816-S). The load cell measured force magnitudes in X, Y, and Z directions, sending the data to a load cell transmitter (ATO-LCTR-OA) and subsequently to

DAQ 1. The accelerometer's vibration measurements were conditioned by three signal conditioners (IEPE) and sent to DAQ 2 (MCC USB-201). The output values from each DAQ were measured in voltage. The wiring diagram of the whole sensing system is illustrated in Figure 4. Using a LabVIEW program, force and vibration signals were converted to newtons (N) and meters per second squared (m/s^2), using calibration methods and known constants, while raw voltage readings were retained for sound signals. The LabVIEW program also managed data transfer and storage, organizing sensor signals into CSV files. The block diagram of the LabVIEW program is provided in Figure 5.

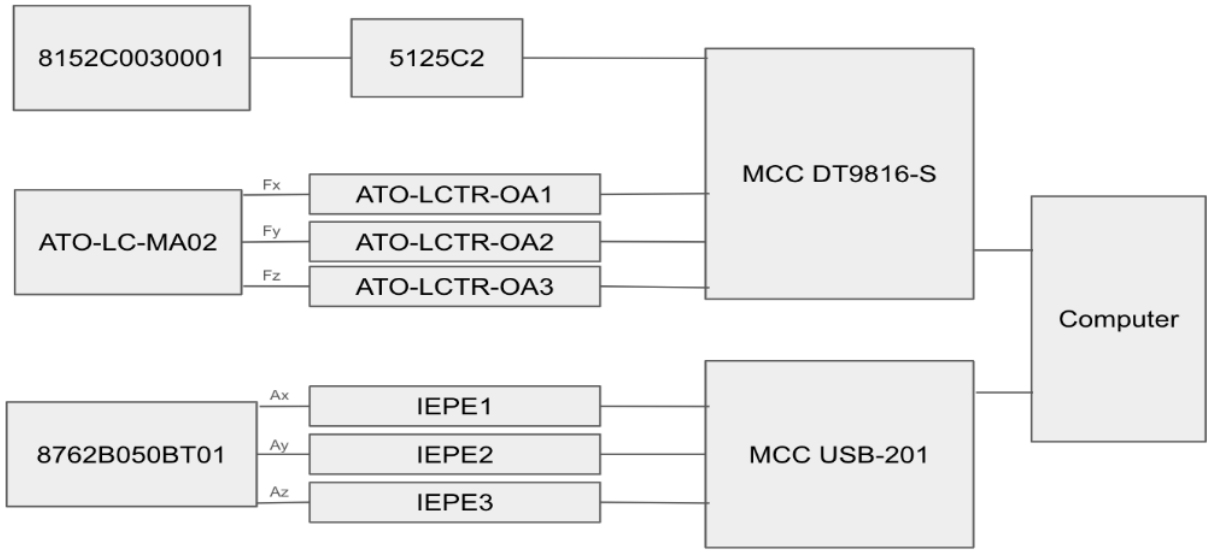


Figure 4. Wiring diagram

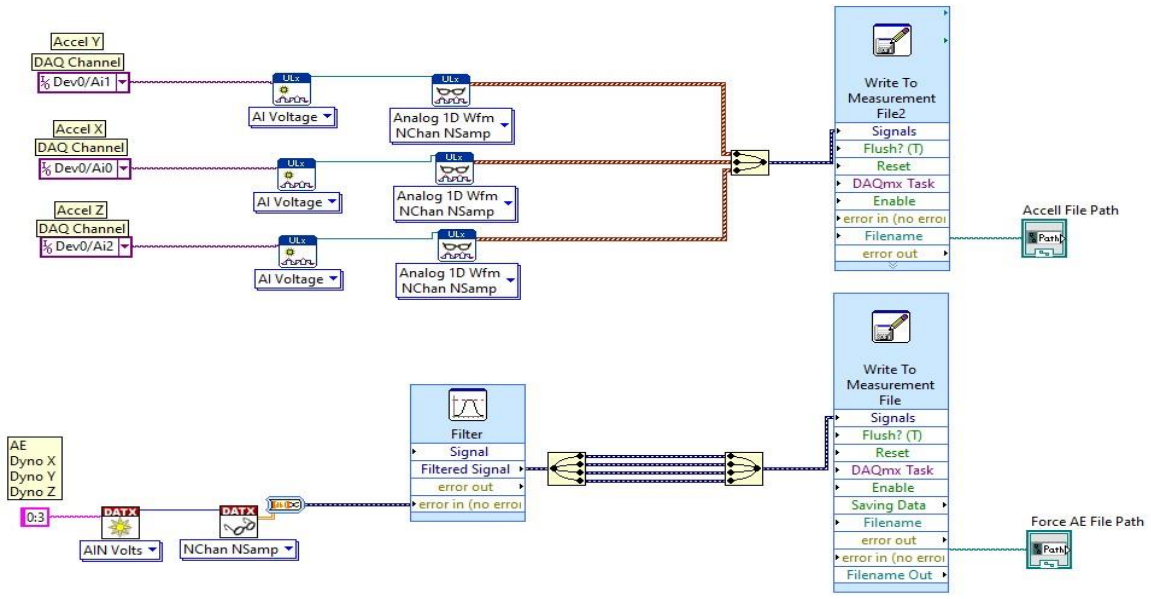


Figure 5. Simplified LabVIEW block diagram

Experimental Procedures

A total of 219 face milling tests were conducted. The experiments employed twenty-six 3/8 inch four-flute high-speed steel square end mills, and 4140 alloy steel billets as work piece material. All tests were performed at consistent operating parameters: the spindle speed was set at 700 RPM, the cutting depth was 0.015 inches, the axial cutting depth (Y) was 0.15 inches, and the feed rate was 2 inches per minute. The spindle speed and feed rate were determined through the GibbsCAM software. In this software, the material of the work piece and tool, along with the power of the machine are inputs, the output is the recommended spindle speed and feed rate. The cutting depths were recommended by a machinist. During each test, seven signal channels were monitored in real time in the LabVIEW program on wave form graphs, capturing data on cutting forces (X, Y and Z), vibrations (X, Y and Z), and acoustic emissions. The sampling rate per channel was set at 50 kHz. Each test involved a single pass facing operation, lasting approximately 40

seconds. The LabVIEW program was run from the moment the tool touched the workpiece, to the moment the tool left the workpiece. After each test, flank wear was measured on each cutting face of a tool's flutes using a microscope (AmScope SM-1T-80S). The recorded tool flank wear value represents the average wear magnitude across the faces of the four flutes, which served as the output variable in the training and testing process for the machine learning models. Tools were utilized until noticeable chipping occurred on any of the cutting edges of the flutes. A given tool could be used for anywhere between two and twenty tests before reaching critical damage. When a tool reached critical damage, a new tool was used for further experiments. Figures 6, 7, and 8 illustrate the raw time-series data extracted for force, vibration, and acoustic emission, respectively.

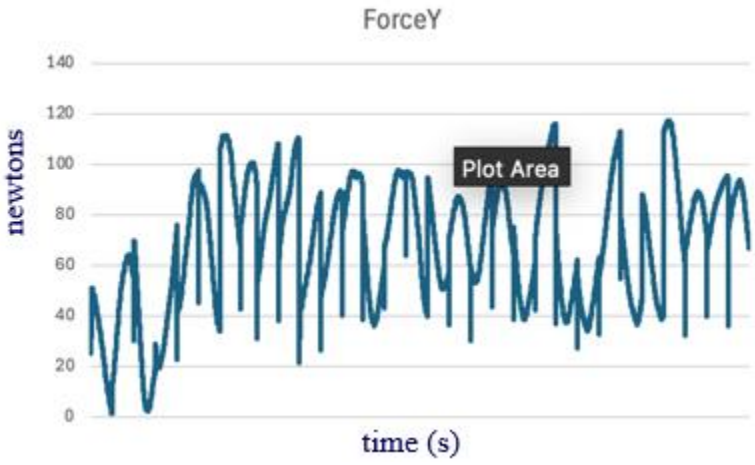


Figure 6. Raw time-series force data for an experiment (time vs. N)

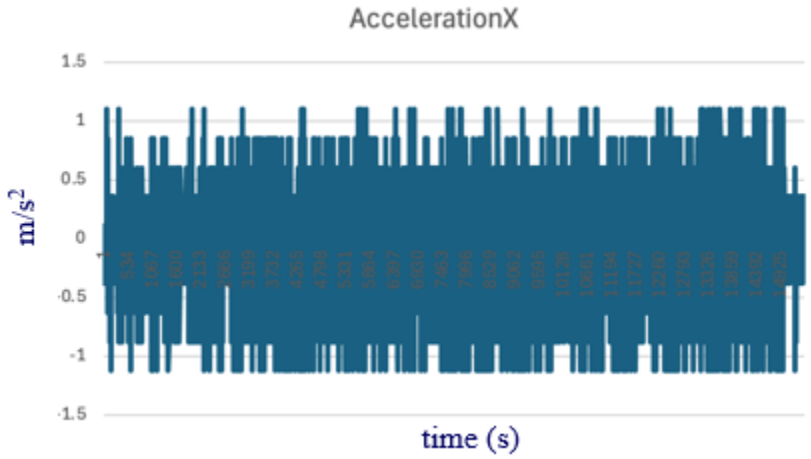


Figure 7. Raw time-series vibration data for an experiment (time vs. m/s^2)

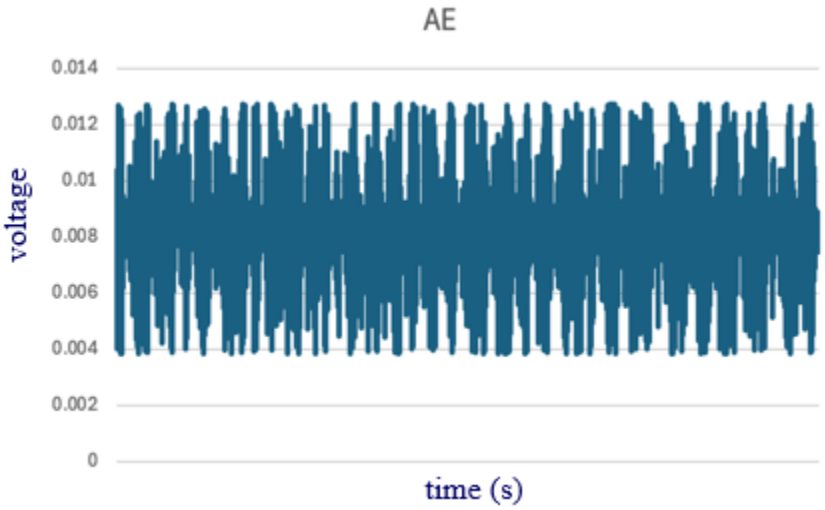


Figure 8. Raw time-series acoustic emission data for an experiment (time vs. voltage)

Machine Software

The data collection for this research involved a single pass face milling operation. The machine was operated using CAMConcept, a software specifically designed for EMCO machinery. To ensure the consistency and reliability of the experimental trials, it was crucial to have a solid understanding of both the machine and software functionality. Consistent machining

parameters, such as cutting depth, were maintained across all operations to capture reliable data. Below are the key machine and software operations that were essential to conduct the trails effectively:

1) Jogging the machine – this was a critical task for setting up each trial. Jogging allowed to manually move the machine to carry out dimensioning of stock and alignment of tools to specific parts.
2) Command prompts – the machine's various functions are controlled via command prompts, which allow for tool selection, spindle activation, and other operations. The command prompt was essential for tasks such as spinning the spindle with a wiggler edge finder to establish the coordinate system for the stock.
3) Program creation – developing a program involved selecting the work offset, choosing the operation to be performed, and designating a tool for an operation (Figure 9). Additionally, key parameters like spindle speed, feed rate, cutting depth (Z-axis), and axial cutting depth (Y-axis) were set. Since the program cannot resume from a mid-point once halted, each new trial required a manual increase in the Y-coordinate by the axial cutting depth of 0.15 inches for continuous operations.
4) Setting the work offset – for precise machining, it was necessary to establish a work offset tailored to each stock piece. This process involved entering the stock's physical dimensions (Figure 10) and using a wiggler edge finger tool to set the machine the part's location. This setup aligned the part reference zeros on the X, Y, and Z axes, ensuring a proper coordinate system for the machine.

5) Tool-to-stock alignment – each tool had to be calibrated to the tool length offset of the stock to maintain accurate cutting depths. This had to be done each time a new tool was used in the machine. Because tools remained in the tool holder during the flank wear measurement process, setting the tool to the part only had to be done anytime the tool was taken out of the tool holder. Any misalignment could lead to incorrect parameter execution and data inconsistency.

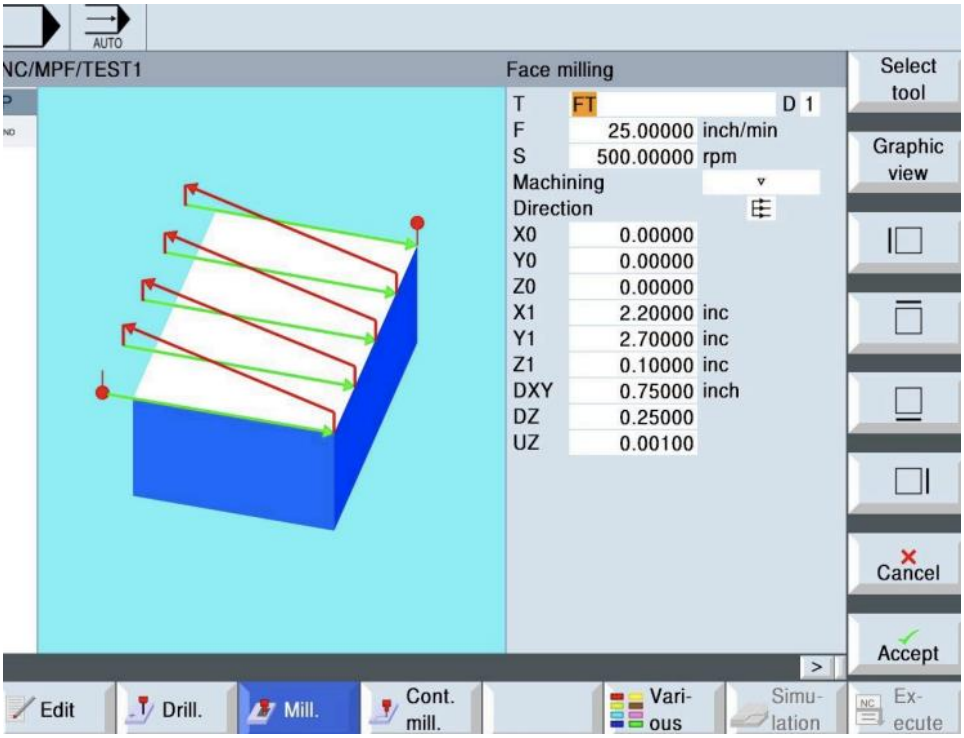


Figure 9. Software interface for program creation

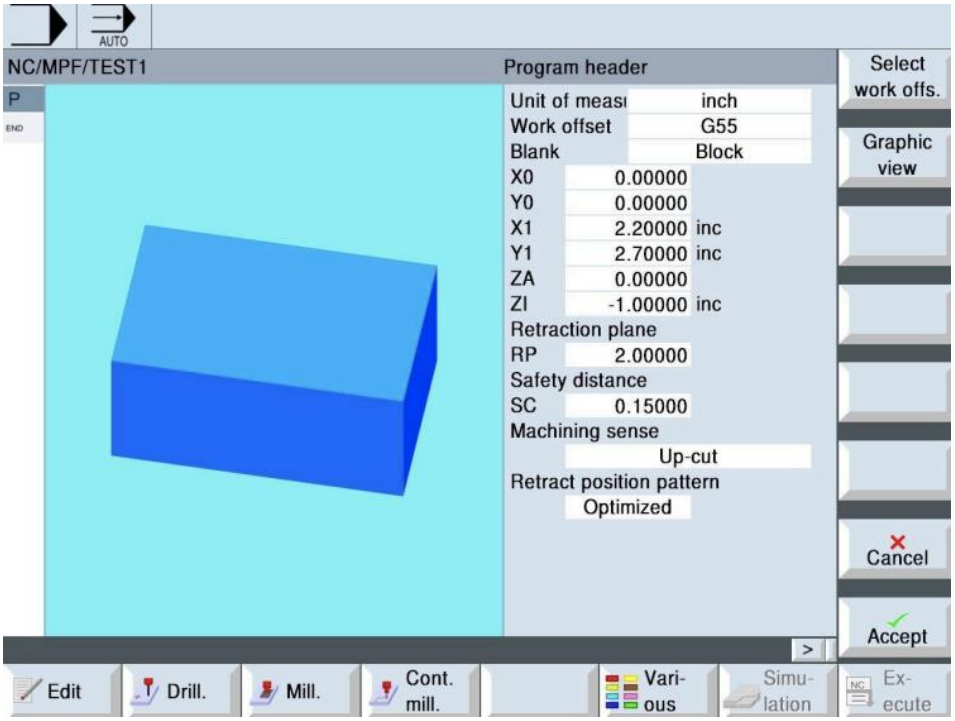


Figure 10. Software interface where stock dimensions and work offsets are assigned

Data Collection Procedures

To maximize the number of data points collected in the shortest time frame, two distinct experimental procedures were evaluated.

Method One: In this method, a batch of tools were used, and data was collected for each operation. After the eighth tool, the tools were taken over to a microscope for tool wear measurements. For eight tools, this process took approximately 90 to 100 minutes, resulting in an average of 11.25 to 12.5 minutes per data point.

Method Two: This method involved using a single tool, gathering operational data, and then measuring tool wear under the microscope. This process was repeated until the tool showed

noticeable chipping on any of the cutting edges, allowing for data points to be obtained in approximately 9 to 10 minutes.

Comparative Analysis: Initially, Method One appeared to be the more efficient option due to reduced walking time between the machine and the microscope. Method One also introduced a restricted range of data, particularly a restricted ranger of larger flank wear values, which can make early analysis of the data difficult. It was also found in Method One that a significant amount of time was spent on tool placement in the tool holder. Additionally, reprograming the machine to recognize the length of a new tool each time further contributed to inefficiencies. Method Two had a shorter cycle time, did not require a tool to be removed from the tool holder, did not require a reprogram, and provided an unrestricted range of data for early analysis. A flowchart showing the data collection process for Method Two is provided below in Figure 11.

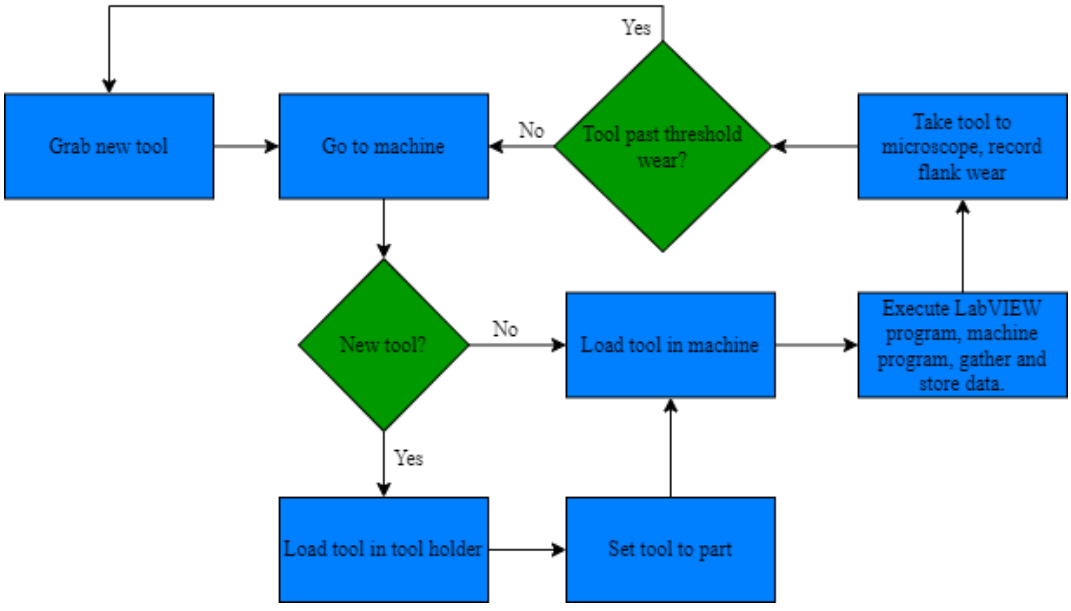


Figure 11. Flowchart of chosen data collection procedure

Tool Flank Wear

In this research, tool flank wear was measured specifically on the cutting face adjacent to the machined part, as illustrated in Figure 8. Kepczak et al. (2020) investigated the relationship between various machining parameters and tool wear during a polymer concrete milling process. Kepczak et al. (2020) identified two primary indicators of wear in end-mills: (1) uniform flank wear, which occurs on the face perpendicular to the machined part; and (2) non-uniform flank wear, which is observed on the face directly adjacent to the part being machined. Figure 12 shows how tools were measured under the microscope.

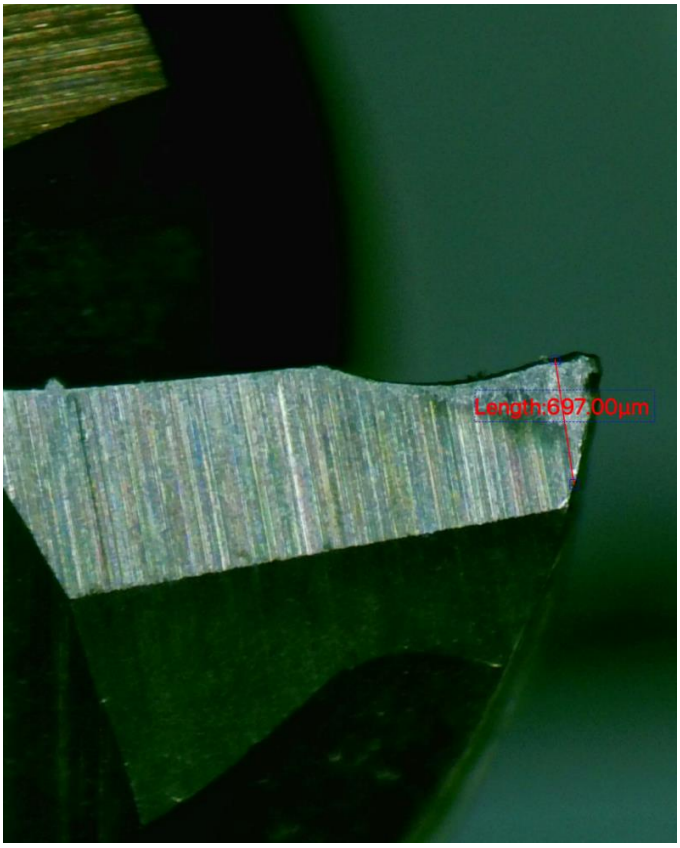


Figure 12. Non-uniform tool flank wear land measurement

During preliminary testing with the tools and stock, it was observed that a significant amount non-uniform flank wear was present on the tools, making it the focal indicator of this research. The wear land refers to the area of the cutting face that has been worn down. For this research, the key measurement recorded for each flute was the maximum distance that the wear land extended from the main cutting edge, in micrometers (µm). Flank wear magnitude values ranged from 100 to 1200 µm.

Data Processing

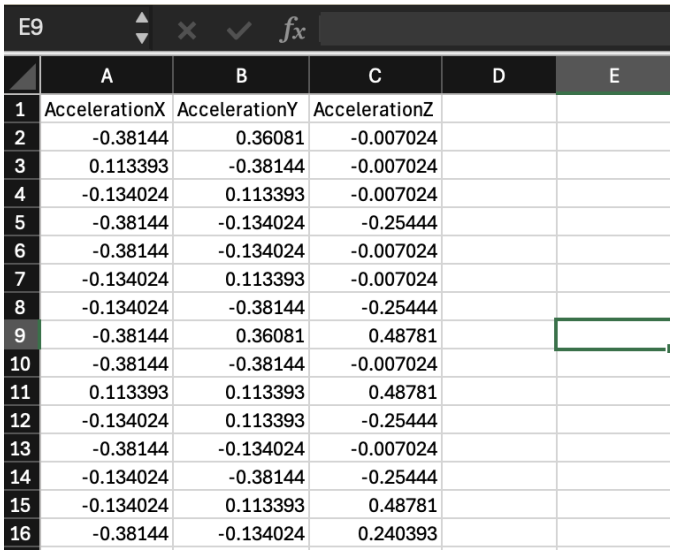
Compiling the Data

To ensure the machine learning models have adequate inputs, an effective data preprocessing step is essential. The raw data collected from the various signal channels for each experiment was transformed into a structured set of statistical features suitable for each machine learning algorithm. Figures 13 and 14 are screen shots of raw data files.

L9

	A	B	C	D	E
1	AE	ForceX	ForceY	ForceZ	
2	0.004488	-73.483654	25.421178	-23.431301	
3	0.004779	-72.720715	26.714082	-21.391089	
4	0.005069	-71.916837	28.060577	-19.312172	
5	0.005322	-71.115937	29.404839	-17.233256	
6	0.005553	-70.318758	30.742402	-15.160293	
7	0.005716	-69.529023	32.07401	-13.085842	
8	0.00591	-68.73631	33.408596	-11.010647	
9	0.006141	-67.942109	34.746159	-8.93173	
10	0.006483	-67.09506	36.151457	-6.802943	
11	0.0069	-66.282995	37.509117	-4.709139	
12	0.008121	-65.424037	38.91888	-2.573653	
13	0.008433	-64.629835	40.251977	-0.499946	
14	0.008552	-63.846799	41.573165	1.564084	
15	0.008731	-63.05483	42.903285	3.634813	
16	0.008895	-62.265839	44.230427	5.702565	

Figure 13. Screenshot of raw data file containing acoustic emission signals and force signals



	A	B	C	D	E
1	AccelerationX	AccelerationY	AccelerationZ		
2	-0.38144	0.36081	-0.007024		
3	0.113393	-0.38144	-0.007024		
4	-0.134024	0.113393	-0.007024		
5	-0.38144	-0.134024	-0.25444		
6	-0.38144	-0.134024	-0.007024		
7	-0.134024	0.113393	-0.007024		
8	-0.134024	-0.38144	-0.25444		
9	-0.38144	0.36081	0.48781		
10	-0.38144	-0.38144	-0.007024		
11	0.113393	0.113393	0.48781		
12	-0.134024	0.113393	-0.25444		
13	-0.38144	-0.134024	-0.007024		
14	-0.134024	-0.38144	-0.25444		
15	-0.134024	0.113393	0.48781		
16	-0.38144	-0.134024	0.240393		

Figure 14. Screenshot of raw data file containing acceleration signals

Selection of Statistical Features

Feature-level fusion was achieved by selecting meaningful statistical features from the raw data. The statistical features included in the dataset were carefully selected to capture critical aspects of the signal data. These features included maximum, minimum, mean, and standard deviation as detailed in Table 1. Each of these features provides valuable insights into the behavior of the machining process, allowing the models to learn more effectively.

Table 1. Selected statistical features

Force (X, Y and Z)	Vibration (X, Y and Z)	Acoustic emission
Max	Max	Max
Min	Min	Min
Mean	Mean	Mean
Standard deviation	Standard deviation	Standard Deviation

The experimental data files were compiled into a single file using the pandas library (The pandas development team, 2023) in Python 3.9. This was executed by iterating through a folder containing all the raw data files and iterating through each column of each file extracting statistical features for each signal (Code Block 1). In the combined file each row corresponds to a test containing 28 distinct statistical features alongside the associated tool flank wear value. Figure 15 is a screen shot of the compiled data file with the statistical features and associated tool flank wear values.

Code Block 1. Compiling the raw force and AE data with pandas

```
import pandas as pd
def calc_stats(folder_path, output_file):   #folder and output file location function
    files = os.listdir(folder_path)            #getting a list of files in the folder
    stats_data = []                            #creating a list to store statistics
    for csv_file in csv_files:                 #for file in folder, construct the path of each file
        file_path = os.path.join(folder_path, csv_file)
        df = pd.read_csv(file_path)            #read the file into a dataframe
        stats = {'file': csv_file, 'raw_signal_mean': df['raw_signal'].mean(), #dictionary
                                   'raw_signal_max': df['raw_signal'].max(),    #that stores
                                   'raw_signal_min': df['raw_signal'].min()     #stat info for
                                   'raw_signal_std': df['raw_signal'].std()}    #each signal
        stats_data.append(stats)               #add the stats to the list
    stats_df = pd.DataFrame(stats_data)    #convert the list to a dataframe
    stats_df.to_csv(output_file, index = False) #write the dataframe to a csv file
folder = '/Users/User/Documents/folder'        #folder path
combined_file = '/Users/User/Documents/combined_file.csv' #name and path of new file
calc_stats(folder, combined_file)
```

X	Y	Z	AA	AB	AC	AD
AccelerationY_std	AccelerationY_max	AccelerationZ_mean	AccelerationZ_min	AccelerationZ_std	AccelerationZ_max	Flank Wear
0.310090979	1.97231	0.023449836	-3.718274	0.353998349	3.209393	110.905
0.298326339	1.8168515	0.021809219	-3.409003	0.337439926	2.90012225	127.0625
0.291213635	1.730485667	0.02034795	-3.154713556	0.32765169	2.700814389	133.5625
0.27590096	1.578189222	0.016727015	-2.742352222	0.31146177	2.563360722	141.745
0.270779068	1.584513333	0.013549868	-2.591153111	0.309934417	2.742050556	153.72
0.272887775	1.735346778	0.010435927	-2.591153111	0.313178407	3.071939556	156.8675
0.278623328	1.872800556	0.009240348	-2.481190222	0.310397711	3.016958222	163.2225
0.287426723	2.120217222	0.00799175	-2.371227222	0.305824764	2.879504556	163.8675
0.296408254	2.202689444	0.007610014	-2.453699444	0.307589737	3.044448889	169.3475
0.298158743	2.230180222	0.007075467	-2.646134667	0.307155824	3.264374667	173.095
0.286401775	2.010254333	0.008529866	-2.646134778	0.300091212	3.346846889	175.605
0.273019106	1.859421	0.011623854	-2.481190444	0.290045119	2.852013667	178.2175
0.269526392	1.818550667	0.014992869	-2.426209	0.29204875	2.577106333	178.665

Figure 15. Screenshot of resulting combined data file

Feature Smoothing

To further enhance the quality of the inputs for the machine learning models, the rows in the combined data file were organized by flank wear magnitude. This arrangement helps to create a more meaningful relationship between the features and output variable. Additionally, each statistical feature column was smoothed using a moving average technique. This smoothing process is instrumental in reducing short-term fluctuations, thereby highlighting underlying trends and patterns within the data. A simple moving average was applied to smooth the data. A simple moving average is an average of data points in a certain window, for this case the window size was set to three. This means that the new value for a given data point is calculated as the average between the proceeding data point, data point, and following data point. It is calculated by taking the sum of a number of data points and dividing it by the number of data points in the sum. A simple moving average is calculated as follows:

$$S_t = \frac{1}{n} \sum_{i=t-n+1}^{t} x_i \qquad (Eq. 1)$$

where S_t is the moving average at step t, n is the window size, and x_i are the data points within the window. By ordering the dataset by flank wear magnitude and applying moving averages to the statistical features, several benefits are achieved. Firstly, this approach mitigates the impact of noise and variability in the measurements, leading to a more stable dataset. Secondly, it facilitates better organization during the training process, as the models can focus on underlying trends rather than reacting to short-term fluctuations. Ultimately, these preprocessing steps enhance efficiency and robustness of the training process, improving the predictive capabilities of the machine learning models. Figure's 16 and 17 illustrate an example of the original data versus the smoothed data for the average force in the Y direction. Code Block 2 shows how this was executed for each statistical feature.

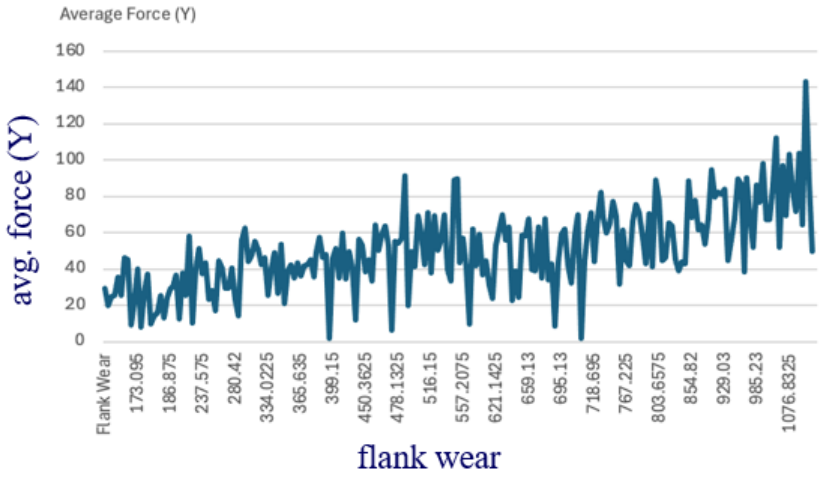


Figure 16. Statistical feature with no moving average

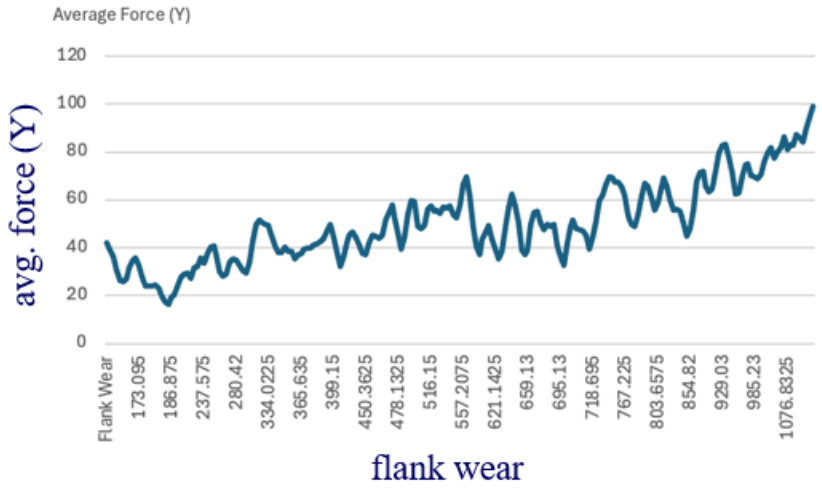


Figure 17. Statistical feature with moving average

Code Block 2. Applying a moving average to statistical features

```
1: import pandas as pd
2: data = pd.read_csv('/Users/User/Documents/combined_file.csv')
3: columns_to_exclude = ['Experiment Number', 'Flank Wear']
4: for column in data.columns:                    #iterate through file columns
5:       if column not in columns_to_exclude:   #apply moving average to column
6:            data[column] = data[column].rolling(window = 3, min periods = 1).mean()
7: data.to_csv('/Users/User/Documents/smoothed_file.csv', index = False)  #new file
```

CHAPTER FOUR

MACHINE LEARNING ALGORITHMS

Model Selection

Preliminary Data

Prior to the experimental phase, selecting three high-performing machine learning algorithms was essential to streamline the experimental process. Testing algorithms beforehand allowed for early insights into each model's behavior and performance, helping to identify potential issues and ensure model readiness. To facilitate this selection, data from Li et al. (2009) was used to assess model performance. Li et al. (2009) gathered force (X, Y, Z), acceleration (X, Y, Z), and acoustic emission signals from 315 high-speed CNC milling operations to estimate tool wear. This dataset provided time-series data across seven signal channels, with each experiment recorded in a CSV file alongside tool wear values. The data was provided to the public and has been widely referenced for model evaluation in previous research. For instance, Wu et al. (2017) previously transformed the raw-time series data from each experiment into various statistical features for each signal channel. The purpose of the study was to compare the performance of RF, ANN, and SVR models when estimating tool wear. Each model achieved coefficient of determination (R^2) values above 0.95 across various training sizes. Given the proven compatibility with machine learning models, the obtained data by Li et al. (2009) and data transformation method by Wu et al. (2017) were selected to evaluate the performance of various algorithms for the current research.

Preliminary Data Transformation

To prepare the data, each of Li et al.'s (2009) raw data files were converted into the 28 statistical features as stated in Chapter Three, using Python libraires pandas and NumPy. Each file was compiled into a single comma separated values (CSV) file, with each row representing one experiment and each column corresponding to statistical features and tool wear values. This transformation applied to the preliminary dataset was planned to be applied to the experimental data in this research, because it was proven to be a good fit for machine learning models, and it was known that the experimental data would consist of seven raw signal channels, particularly force (X, Y, and Z), acceleration (X, Y, and Z), and acoustic emission.

Initial Model Performance Assessment

Five machine learning models were initially selected for performance assessment: (1) RF; (2) ANN; (3) MLR; (4) SVR; and (5) XGBoost. Each model was trained and tested on the combined dataset, with R^2 used as the performance metric for its applicability to regression problems. Following this evaluation, RF, XGBoost, and ANN models demonstrated the highest R^2 values when tested on the preliminary data obtained by Li et al. (2009). Because the RF, XGBoost, and ANN models demonstrated high performance with similar preliminary data, these models were selected for this research.

Machine Learning Model Architecture and Key Hyperparameters

With the models constructed and tested on preliminary data, the next step is to understand and leverage the strengths of each algorithm by breaking down key components in each model. Certain components play a critical role in driving the predictive power of RF, ANN, and XGBoost

models. The following sections discuss these components in detail, highlighting the architecture and key hyperparameters for each model.

Artificial Neural Networks

ANNs are a class of machine learning models inspired by the behavior of biological neural networks. In this research, the ANN was developed using the Keras library in Python 3.9, allowing for efficient design and training of the model. As shown in Figure 16, the structure of the ANN consists of five core components: (1) the input layer; (2) hidden layers; (3) the output layer; (4) weights and biases; and (5) the training process. A key hyperparameter for ANNs is the number of nodes in the hidden layers. This architecture is illustrated below in Figure 18.

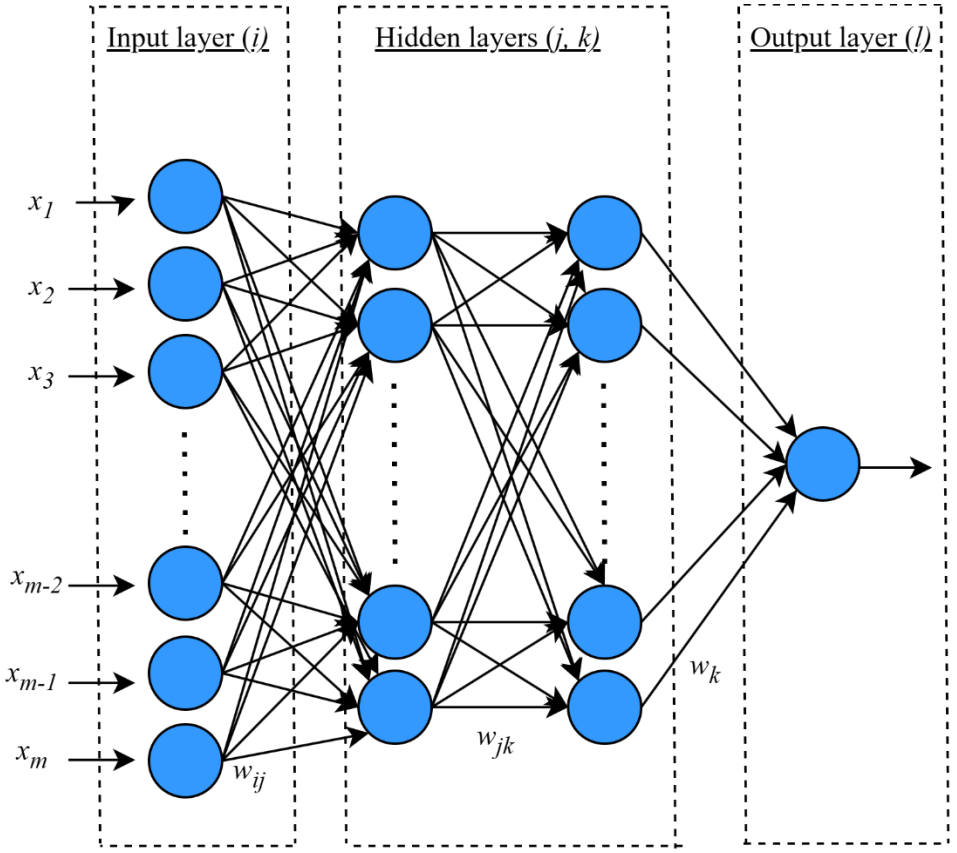


Figure 18. ANN architecture

Input Layer: The input layer of the ANN is responsible for receiving the input data, denoted as x_i. In this case, the inputs were the extracted statistical features derived from the raw signal data. The input layer comprises nodes, or neurons, with the number of nodes corresponding to the number of input features (m = 28).

Hidden Layers: The primary function of the hidden layers is to process the input data through multiple neurons and pass the results to the output layer. Each neuron in the hidden layers computes a weighted sum z_j based on the inputs from the previous layer, followed by an activation function a_j. The mathematical formulation for the output of a neuron j in a given layer is:

$$z_j = \sum_{i=1}^{m} w_{ij}\, x_i + b_j \qquad (Eq. 2)$$

where x_i are the inputs to the neuron, w_{ij} are the weights for the inputs, b_j is the bias term for neuron j, and m is the number of inputs to the neuron. The result z_j is then passed through a rectified linear unit (ReLU) activation function, defined as:

$$a_j = ReLU(z_j) = \max(0, z_j) \qquad (Eq. 3)$$

The $ReLU$ function is particularly effective for regression tasks, as it helps ANNs learn complex relationships in data and mitigates the vanishing gradient problem, which can arise during the training of deep networks.

Output Layer: The output layer plays the role in generating the final predictions $\hat{Y}_i$ based on the computations from the hidden layers. Because the tool flank wear value is continuous, there is only one neuron in the output layer. The output $\hat{Y}_i$ is computed as:

$$\hat{Y}_i = \sum_{k=1}^{p} \left(w_k \sum_{j=1}^{n} (w_{jk} a_j) + b_k \right) + b \qquad (Eq. 4)$$

where b is the bias term for the output neuron. The connections between neurons in one layer to the next has an associated weight (w_{ij}). During the training process, the weights in the connections are adjusted to minimize the prediction error. Each neuron in each layer also has an associated bias b_j, which shifts the activation function to improve the flexibility in learning.

ANN Training: ANNs are trained by minimizing a loss function. The loss function selected was mean squared error (MSE), which is typical for regression problems. The loss function can be computed as:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2 \qquad (Eq. 5)$$

where N is the number of samples and Y_i is true value for sample i. The training process employs backpropagation and gradient descent to adjust weights and biases iteratively, aiming to reduce the loss function. The weights and biases are calculated as follows:

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \left(\frac{\partial MSE}{\partial w_{ij}} \right) \qquad (Eq. 6)$$

where $w_{ij}^{t}$is the weight of the *t-th* iteration, ɳ is the learning rate and $\partial MSE\ /\partial w_{ij}$ is the gradient of the loss function with respect to weight. The adam optimizer was chosen for this study, which is a stochastic gradient descent method that adapts the learning rate based on first and second moment estimates of the gradients. This optimization technique helps in achieving faster convergence and better performance.

Random Forests

RFs are an ensemble learning method that combines multiple decision trees for classification and regression problems. In RFs, classification is based on a majority vote, while regression takes the average of all decision tree's predictions. This ensemble method reduces the likelihood of overfitting, making RFs more robust than a single decision tree. The structure of a decision tree is comprised of three key features: (1) decision nodes, decision points where data is split, often determined by the MSE method for regression problems; (2) leaf nodes, or the final predictions; and (3) the branches, which are the connections showing decision outcomes. This research implemented a RF using Scikit-Learn's RandomForestRegressor, with key hyperparameters such as number of trees, bootstrap sampling, max tree depth and minimum samples per node. Figure 19 illustrates the general architecture of a RF.

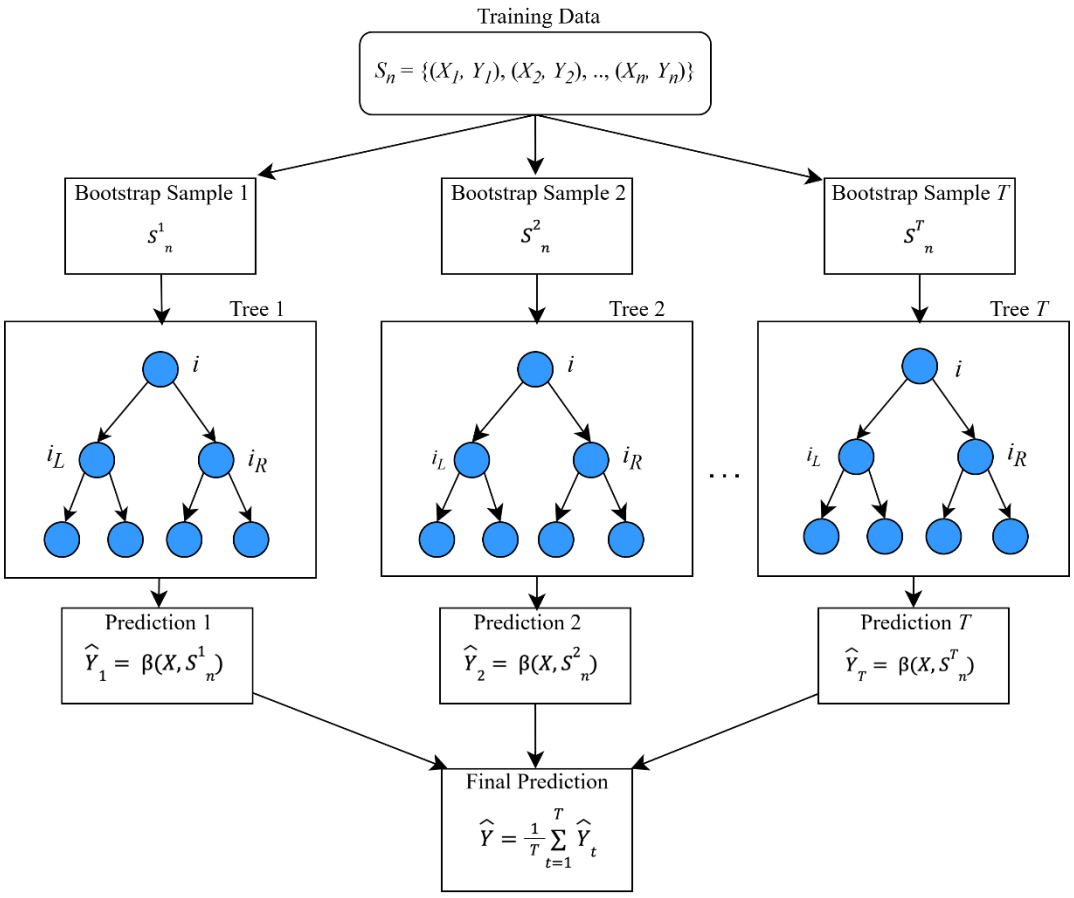


Figure 19. RF architecture

Bootstrap Sampling: Bootstrap sampling helps introduce variability amongst all the decision trees. Bootstrapping refers to randomly sampling the training data with replacement. This ensures that each tree is trained on a slightly different dataset, which reduces the likelihood of overfitting and helps improve the generalization ability of the model. Let X be the input vector with m features, $X = \{x_1, x_2, .., x_m\}$, Y the target variable, and S_n the training dataset containing n data points:

$$S_n = \{(X_1, Y_1), (X_2, Y_2), .., (X_n, Y_n)\} \qquad (Eq. 7)$$

In a RF, a random subset of features is considered at each decision node. This subset ensures that each split is based on only a fraction of the available features, introducing further availability between trees. During the training process a bootstrap sample is gathered by randomly selecting observations with replacement from S_n, by default the number of observations is typically 63.2% of the training dataset. Then for each split within a tree, only a certain number of features from X are evaluated as candidates for the best split. The number of features considered at each split is the square root of m, which is the default in RandomForestRegressor. In the bootstrap process, the algorithm selects several bootstrap samples for each tree in the ensemble with T trees:

$$Boostrap\ samples = (S_n^1, .., S_n^T) \qquad (Eq. 8)$$

Limiting the maximum depth of a tree prevents overfitting. Restricting depth ensures that each tree captures general trends rather than noise.

Splitting Criteria: Each tree grows by splitting data at decision nodes. In regression trees the splitting criterion is based on minimizing the MSE as shown in Eq. 9. The tree continues to split nodes until the specified maximum depth or other specified stopping criterion is met (minimum samples per leaf). The minimum samples per leaf node ensures that at least a certain number of samples must be present. If the leaf node contains too few samples, it has the possibility of representing an outlier or noise. Minimum samples per split dictates that at least a certain number of samples are required to consider splitting a node. The impurity criterion for splitting nodes in regression trees is based on minimizing the total MSE after the split. For a split at node i into two child nodes i_L and i_R, the objective is to minimize:

$$\Delta MSE = MSE(i) - \left(\frac{N_L}{N} MSE(i_L) + \frac{N_R}{N} MSE(i_R)\right) \qquad (Eq. 9)$$

where N represents the total number of samples at parent node i, N_L and N_R are the number of samples in the left and right child nodes. MSE(i_L) and MSE(i_R) are the MSE of the child nodes.

Splitting Criteria Example: The splitting process for a RF is best understood visually. For a bootstrap sample S_n^t for a single tree t, each sample in the bootstrap is comprised of $\sqrt{m}$ randomly selected statistical features x_i with an associated Y_i value, which can be visually represented as a single red dot (Figure 20).

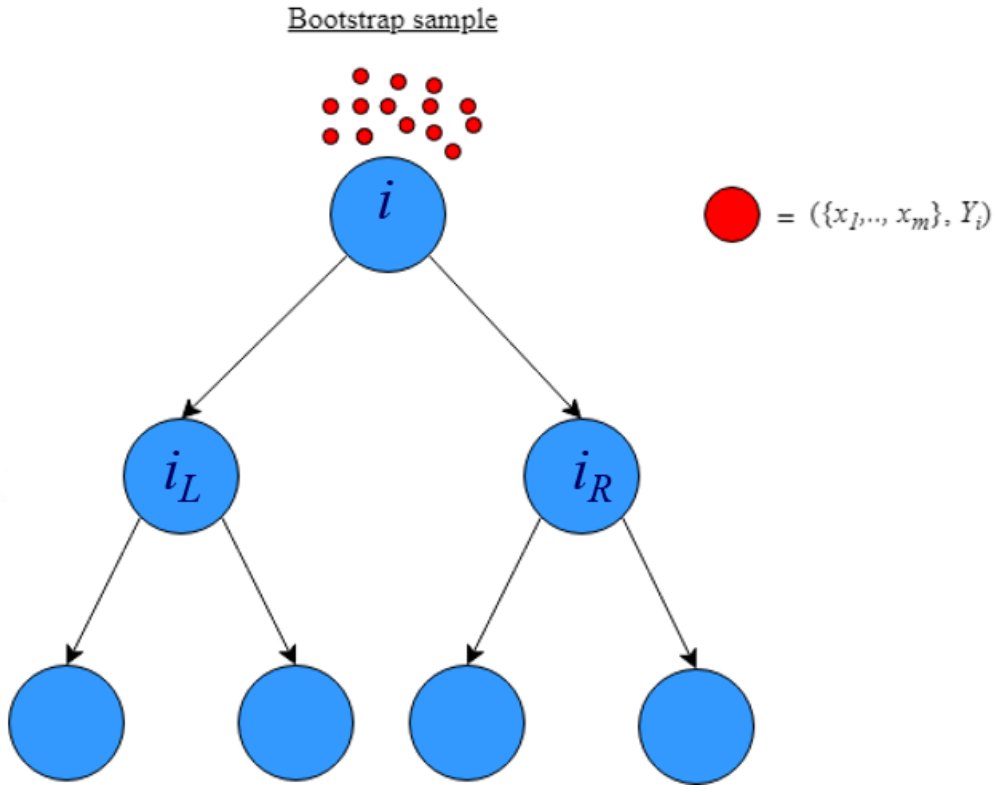


Figure 20. RF splitting criteria first iteration

Initially, the algorithm is looking at each x_i along with the value of each x_i to split on (e.g. $x_7 = 50$ newtons of average force in the Y direction), that minimizes the total variance amongst the Y_i's. For this example, the algorithm found the best value to split on was $x_7 = 50$ newtons of average force in the Y direction (Figure 21).

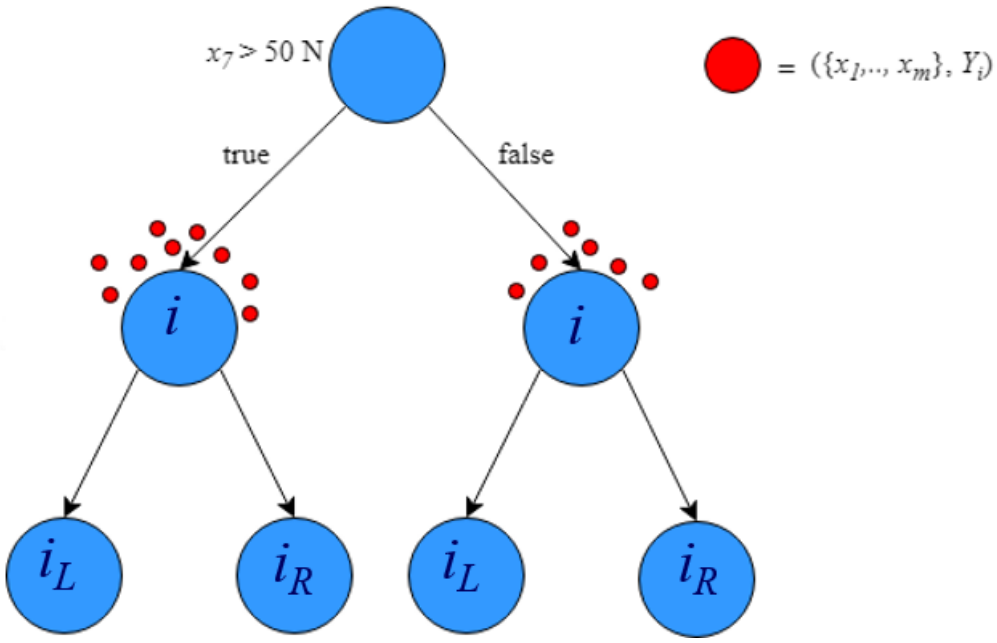


Figure 21. RF splitting criteria second iteration

The splitting at each parent node i is repeated recursively until a stopping criterion is met (e.g. max depth). When a stopping criterion is met, each leaf node in each tree provides a value Y_t, which is the average of the Y_i's in each sample that ended up in a leaf node (Figure 22).

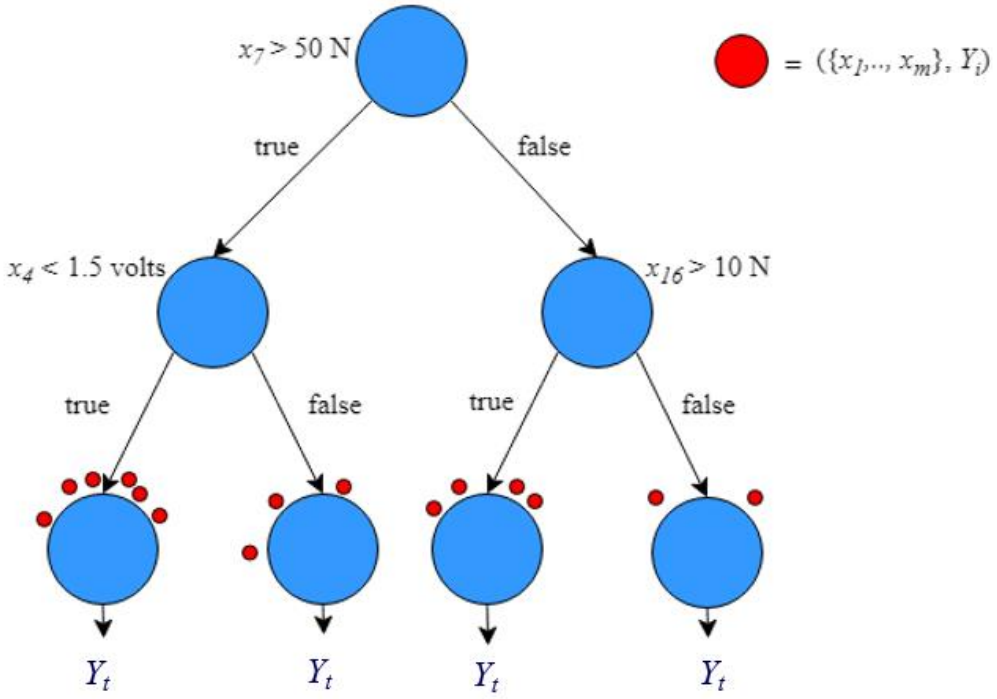


Figure 22. RF splitting criteria final iteration

RF Testing Process Example: For the testing process, the decision nodes for a tree *t* evaluate a test sample *X* with the and pass it to different branches based on threshold values specified in the nodes. As an example, the input vector $X = \{x_1, x_2, .., x_m\}$ comprised of feature values $\{x_7 = 100$ N, $x_4 = 0.5$ volts$\}$ is evaluated at each decision node. *X* (the green dot in Figures 23, 24, and 25) is directed down the branches according to the values of its features.

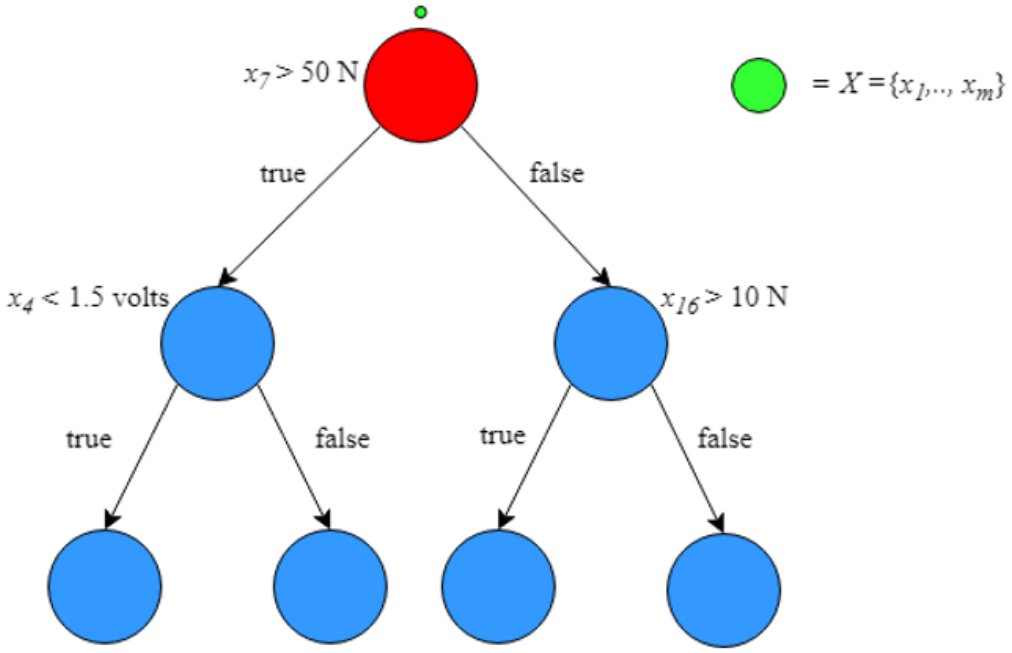


Figure 23. RF testing process first iteration

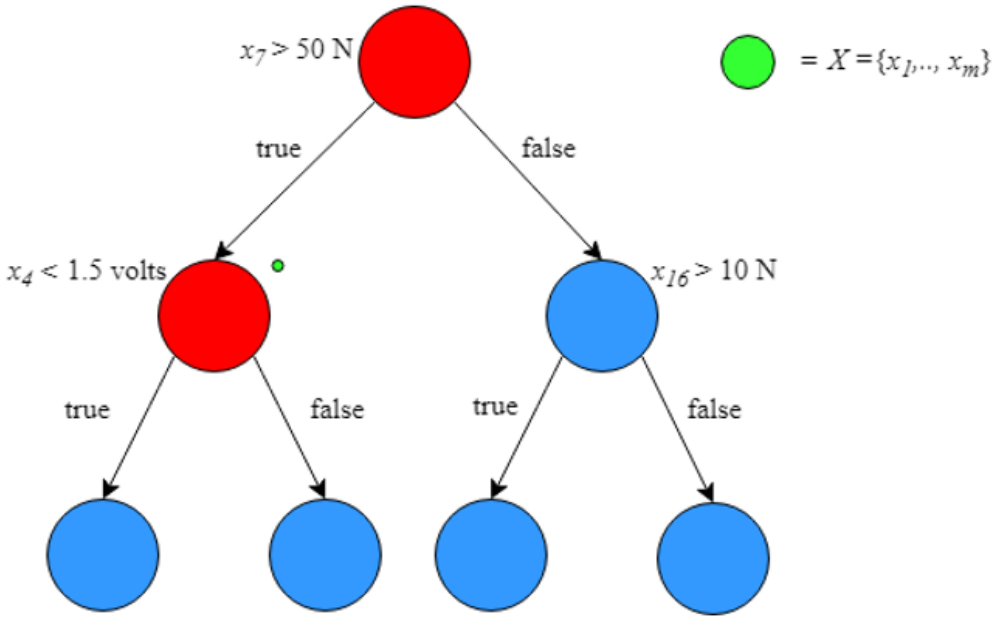


Figure 24. RF testing process second iteration

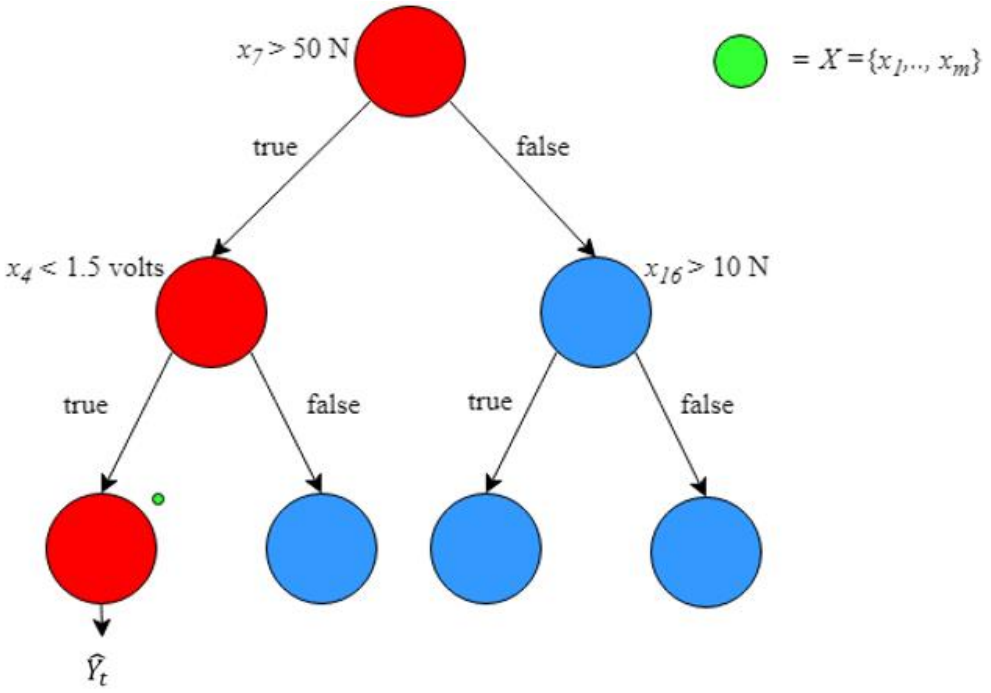


Figure 25. RF testing process final iteration

The test sample X ends up in a single leaf node based on the values of its features. The predicted value for a test sample X in tree t is $\hat{Y}_t$. This process is performed by all trees in a RF for a test sample X, producing T outputs. The final prediction for a single test sample $\hat{Y}$ is obtained by averaging the outputs of all trees:

$$\hat{Y} = \frac{1}{T}\sum_{t=1}^{T} \hat{Y}_t \qquad (Eq. 11)$$

where $\hat{Y}_t$ is the output of the *t-th* tree (t = 1, 2, .., T).

Extreme Gradient Boosting

XGBoost is a powerful decision tree ensemble method commonly used for classification and regression tasks. XGBoost is similar to a RF, however while a RF builds independent trees,

XGBoost builds trees sequentially with each new tree correcting the errors of the previous ones. This research implemented XGBoost using Python's xgboost library. Some key hyperparameters include number of trees, maximum tree depth, learning rate, subsample, and column sample size. Figure 26 illustrates the general architecture of a XGBoost model with *T* trees. With XGBoost each tree corrects the errors from the previous tree based on a loss function:

$$\hat{Y}_n = \sum_{t=1}^{T} f_t\,(X_n) \qquad (Eq.12)$$

where $\hat{Y}_n$ is the prediction for sample n, and f_t is the prediction from the *t-th* tree.

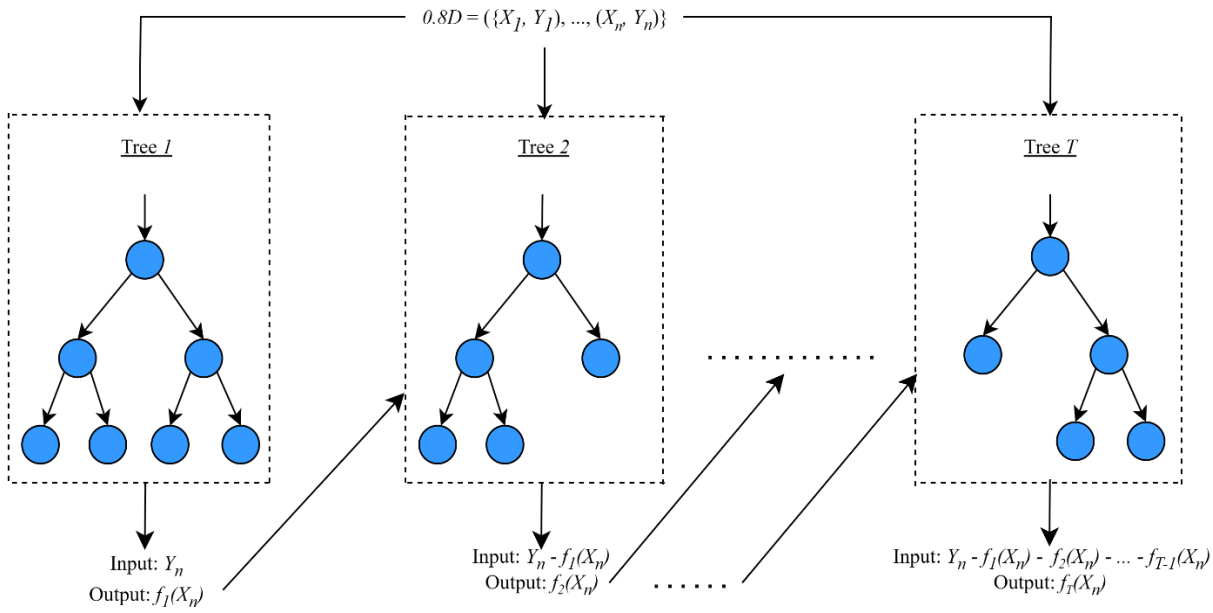


Figure 26. XGBoost architecture

Tree Depth: The max depth per tree is an important hyperparameter for decision trees in an XGBoost algorithm. Deeper trees can learn more complex patterns but can introduce overfitting if the max depth is too high. The max depth can be expressed as:

$$d_{max} = \max\big(depth(f_t)\big) \qquad (Eq. 13)$$

where d_{max} is the maximum depth of any tree f_t in the model.

Learning Rate: The learning rate (γ) controls the learning rate in each iteration. A smaller learning rate typically requires more trees to converge, which might make the training process longer but can lead to a better generalization. The updated prediction $\hat{Y}_n$ at step *t* based on the learning rate γ can be represented as:

$$\hat{Y}_n^t = \hat{Y}_n^{t-1} + \gamma\big(f_t(X_n)\big) \qquad (Eq. 14)$$

where $\hat{Y}_n^t$ is the prediction after adding the *t-th* tree.

Subsample Size: The subsample hyperparameter controls the percentage of the training data used to build each tree, which is a crucial hyperparameter when trying to prevent overfitting. Lower subsample values make the algorithm more robust, preventing trees from being too closely fit to the data. Subsampling is a form of stochastic gradient boosting, which can be expressed as:

$$N_{tree} = s(N_{total}) \qquad (Eq. 15)$$

where N_{tree} is the number of samples in the data used to train each tree and N_{total} is the total number of training samples in the data. The subsample fraction *s* is a specified value between 0 and 1.

Number of Features Per Tree: The column sample hyperparameter controls the fraction of features to be used by each tree. The mitigation of overfitting is heavily taken care of with the

subsample hyperparameter, which is the fraction of features each tree can be exposed to. For a dataset with *M* features the number of features used by each tree is:

$$M_{tree} = c(M_{total}) \qquad (Eq. 16)$$

where M_{tree} is the number of features from the dataset used for training each tree and M_{total} is the total number of features in the data. The column sampling rate *c* is a specified value between 0 and 1.

Stacking Ensemble

The third level of data fusion is at the decision level (Hall and Llinas, 1999). In order to achieve this, an ensemble method which fuses the predictions of various machine learning algorithms was selected. An article Bajaj (2023) published outlined various ensemble learning methods to combine predictions from various algorithms, highlighting "stacking" as a powerful ensemble method for regression problems. A stacking ensemble is a machine learning technique that combines the predictions produced by multiple base models to produce a more robust, accurate prediction. Stacking ensembles can be used for classification and regression tasks. In a stacking ensemble, different base models are trained on the same dataset. Once the base models are trained, the base models generate predictions for both a testing set, and a validation set. The test set is used for meta-model training, where the meta-model sees the predictions generated by the base models, alongside the true values. Once the meta model is trained, it is tested on the validation set, using the predictions generated by the base models on the validation set as input features, with the overarching goal of generating a more accurate fused prediction. A figure of a stacking ensemble can be seen in Figure 27.

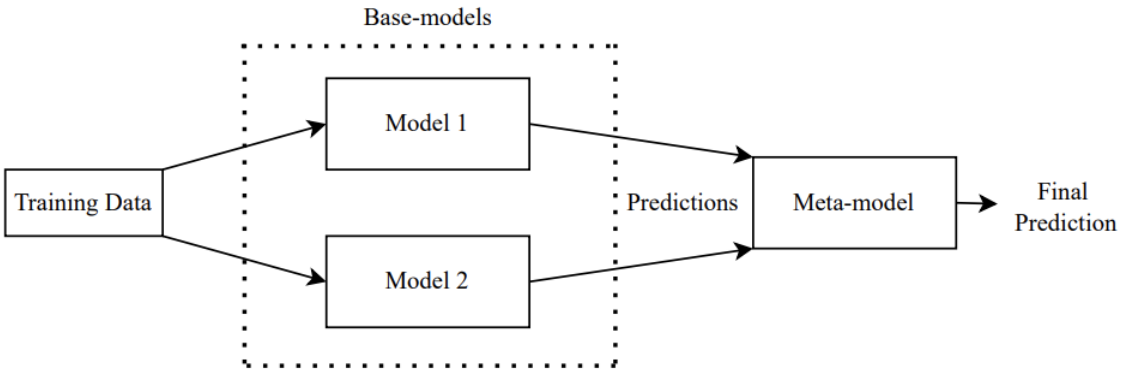


Figure 27. Stacking ensemble architecture

Model Construction

The RF, ANN, and XGBoost models were built in Python 3.9. Key libraries included Scikit-Learn (Pedregosa et al., 2011) and NumPy (Harris et al., 2020), particularly for functions like "train_test_split", and regression metrics.

RF and XGBoost Model Construction

The RF and XGBoost models are very similar in terms of code. The RF algorithm was constructed with the RandomForestRegressor function from Scikit-Learn. The XGBoost algorithm was made using xgboost's (Chen and Guestrin, 2016) XGBRegressor. To learn how to make a RF in Python, a tutorial by Dhiraj (2023) was referenced. The XGBoost model was also made following a tutorial made by Brownlee (2021). The following steps are how the RF and XGBoost models were built, using the RF as an example.

Data Loading and Preprocessing: The preliminary data was loaded using the pandas library (The pandas development team, 2023) (Code Block 3). Features (line 3) and the target variable (line 4) were defined, including the 28 statistical features and the associated tool wear value.

Code Block 3. Data loading and preprocessing

```
1: import pandas as pd
2: data = pd.read_csv('/Users/User/Documents/file.csv')
3: features = data[('name of each feature column')]        #statistical feature columns
4: target = data['Flank Wear']                             #target value columns
```

Train-Test Split: The preliminary data was split into training and testing sets (Code Block 4), 70% of the data used for training, 30% used for testing (line 2). This was done using Scikit-Learn's "train_test_split". In "test_train_split", the features and target variables are defined for both the testing and training sets. The test or train size is a specified as a fraction of the dataset between 0 and 1.

Code Block 4. Train-test split

```
1: from sklearn.model_selection import train_test_split     #splitting data into train, test sets
2: x_train, x_test, y_train, y_test = train_test_split(features, target, test_size = 0.3)
```

Model Initialization: The model was initialized by importing RandomForestRegressor from the Scikit-Learn library (Code Block 5). The hyperparameters for both RF and XGBoost models are entered into the RandomForestRegressor and XGBRegressor functions, respectively. Some hyperparameters include tree depth, number of trees, and learning rates. For now, the only hyperparameter used was "bootstrap". "bootstrap" was set to "True"; this randomly selects data points with replacement from the training set. This ensures that each decision tree in the forest is slightly different, leading to a diverse set of trees. A function known as "random_state" was also used, "random_state" is a vehicle for reproducibility. "random_state" was set to the max value of 42, this is used for reproducibility in performance.

Code Block 5. Model initialization

```
1: from sklearn.ensemble import RandomForestRegressor
2: model = RandomForestRegressor(random_state = 42, bootstrap = True)    #defining model
```

Model Training and Testing: The next step is to train the model and generate predictions (Code Block 6). The model is trained on the training data (line 1), which is done by fitting the model to learn the features and the target variable. Once the model is trained, it generates predictions on the test set with the features as input variables (line 2). The model makes predictions based the relationships it learned between the features and the target variables during the training process.

Code Block 6. Model training and testing

```
1: model.fit(x_train, y_train)          #training the model
2: y_pred = model.predict(x_test)       #testing the model
```

Model Evaluation: The metrics used to evaluate the model's prediction accuracy was root mean squared error and R^2 (coefficient of determination), using a metrics function provided by Scikit-Learn (Code Block 7). R^2 represents the proportion of the variance in the target variable that is explained by the input variables (line 3). Root mean squared error measures the average difference between the models' predicted values and the true values, expressed in the same units as the target variable (line 2).

Code Block 7. Model evaluation and performance metrics

```
1: from sklearn.metrics import mean_squared_error, r2_score
2: mse = mean_squared_error(y_test, y_pred, squared = False)    #root mean squared error
3: r2 = r2_score(y_test, y_pred)
4: print('Root Mean Squared Error:' , mse, 'micrometers')
5: print('R-Squared:', r2)
```

ANN Model Construction

The ANN was constructed using some of the same libraries used for the RF and XGBoost models, such as pandas, NumPy, and Scikit-Learn. However, the construction of the ANN required some additional libraries, such as Keras, a high-level open-source neural network library that runs on top of a backend framework called TensorFlow (Abadi et al, 2016). The structure of the ANN was made using Sequential, Dense, and Input, all functions in the Keras library. A step-by-step guide provided by Dutta (2024) was referenced to code the ANN. The following steps outline how the ANN was made.

Data Loading and Preprocessing: Like the RF and XGBoost construction, the data was loaded and read using pandas. The features selected represent the 28 statistical features, the target variable selected represents the associated tool wear values.

Standardizing the Data: Data standardization is an important step for neural networks (Code Block 8). Standardizing ensures that the data has a mean of 0 and a standard deviation of 1, this helps with improving model convergence during the training process. The data was scaled using StandardScaler, a function in the Scikit-Learn library.

Code Block 8. Data standardization

```
x = data[features].values
y = data[target].values.reshsape(-1,1)
from sklearn.preprocessing import StandardScaler
feature_scaler = StandardScaler()                    #standardizing feature values
target_scaler = StandardScaler()                     #standardizing target values
feature_scaler_fit = feature_scaler.fit(x)
target_scaler_fit = target_scaler.fit(y)
x = feature_scaler_fit.transform(x)
y = target_scaler_fit.transform(y)
```

Train-Test Split: The data was split into training and testing sets, 70% used for training, 30% used for testing.

Building and Compiling the ANN Model: The Keras library was used to build a sequential neural network model, using Sequential, Dense, and Input functions (Code Block 9). The input layer was made using Input, which defines the shape of the input data to determine the number of input nodes (line 5). The hidden layers and output layer was made with Dense (lines 7 and 9). The two hidden layers were set to have 10 nodes each, and an activation function "ReLU", for non-linearity. The output layer is a single node, playing the role of predicting the target value (line 11). The ANN was compiled with "mean_absolute_error" loss function, typically used for regression (line 13). The optimizer used was "adamw", which adapts learning rates during training to improve convergence.

Code Block 9. Building and compiling the ANN

```
1: from keras.models import Sequential
2: from keras.layers import Dense, Input
3: model = Sequential()
4: #defining the number of nodes in the input layer, the dimensions of the input data
5: model.add(Input(shape = (x_train.shape[1],)))
6: #defining the number of nodes, weights, and activation function in hidden layer 1
7: model.add(Dense(units = 10, kernel_initializer = 'random_normal', activation = 'relu'))
8: #defining the number of nodes, weights, and activation function in hidden layer 2
9: model.add(Dense(units = 10, kernel_initializer = 'random_normal', activation = 'relu'))
10: #defining the number of nodes and weights in the output layer
11: model.add(Dense(1, kernel_initializer = 'random_normal'))
12: #compliling the model, specifying the loss function, and optimizer for weight updates
13: model.compile(loss = 'mean_squared_error', optimizer = 'adam')
```

Training the Model: The ANN was trained on the training data. The batch size was set to 5, which is the number of samples fed to the network for each iteration. This helps the training be

more manageable and efficient. The epoch size was set to 100, which is the number of times the ANN will see each sample in the training data.

Predicting and Transforming Back: The trained ANN model was used to predict the target values on the test set (Code Block 10). The predictions were also transformed back to the original scale using "inverse_transform". This was done to provide interpretable results in the target units (e.g., micrometers).

Code Block 10. Predicting and transforming the data back

```
1: y_pred_original = target_scaler.inverse_transform(y_pred)   #transforming the predicted
2: y_test_original  = target_scaler.inverse_transform(y_test)   #and true values back
```

Model Evaluation: The predictive performance of the ANN was evaluated using root mean squared error and R^2, higher R^2 values corresponding to higher model accuracy.

Stacking Ensemble Construction

A stacking ensemble requires a unique training and testing process as opposed to training and testing a single machine learning model. To train and test a single machine learning model, the total data is divided into two sets, one for learning, and one for performance evaluation. The traditional two-way split in the data is not valid in a stacking ensemble approach, because it does not set aside a fraction of data to evaluate the performance of a higher-level model. To ensure reliable results from a stacking ensemble, the total data is split into three sets, to have training data for base models, testing data for base models (training for the meta model), and a final set to assess the performance of the higher-level model. The three-way split is essential in stacking because it ensures that the higher-level model is tested on data that it has not been exposed to. This unique training process can be visualized below in Figure 28.

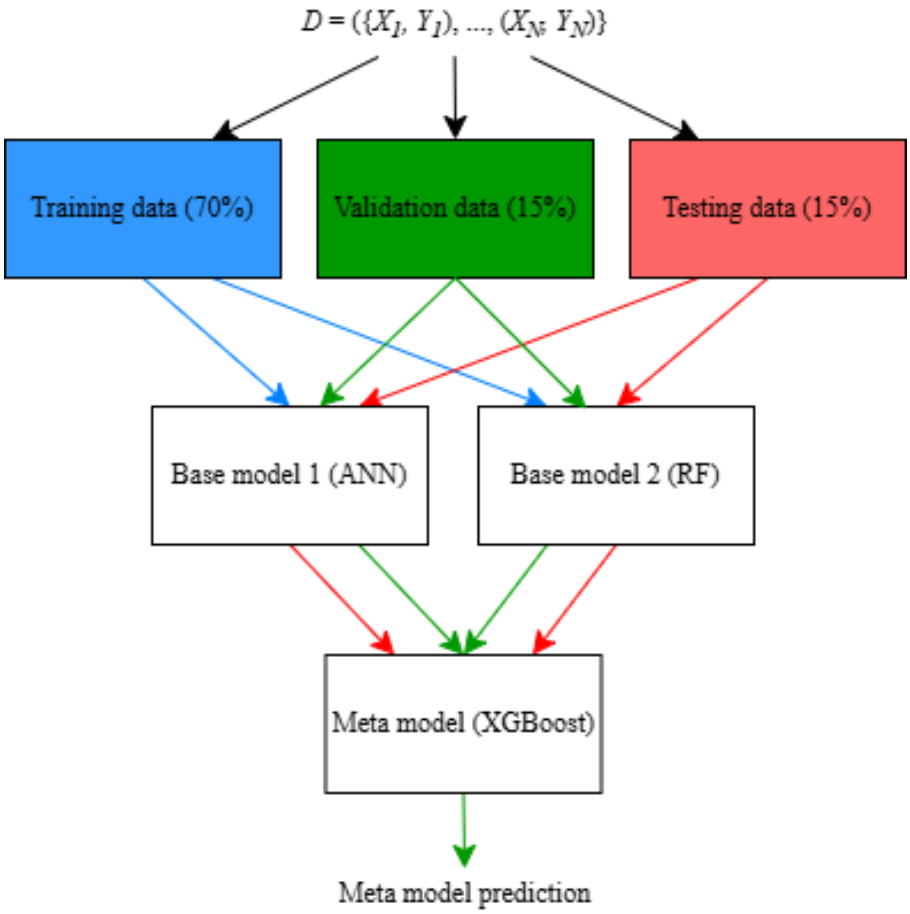


Figure 28. Stacking ensemble training and testing process

Stacking Ensemble Training and Testing: An example of the training and testing process for a stacking ensemble with a RF and XGBoost as base models, and another RF as a meta-model is provided below in Code Block 11. In a stacking ensemble, different base models (such as a RF and XGBoost) are trained on the same dataset, which is 70% of the total data (lines 9 and 10). Once the base models learn the relationships between the features and the true target values, the base models generate predictions on two testing sets, test set 1 and test set 2 (lines 11, 12, 13, and 14). Both sets are 15% of the data. This three-way split in the data is crucial in stacking, because the higher-level model needs a training set to learn the relationships between the features (base

model predictions on test set 1) and the true values, along with a testing set comprised of features (base model predictions on test set 2) it has not been exposed to. In this case, the meta model is trained on the predictions generated for test set 1, which are the base model predictions stacked in an array (line 17), alongside the true values for each sample (line 21). Based on the relationships the meta-model learned between the base model predictions on test set 1 and the true values, it makes predictions on a validation set (line 22), which is a stacked array of predictions the base models generated for test set 2 (line 19).

Code Block 11. Stacking ensemble training and testing process

```
1: #splitting the data into training and testing sets (70%, 30%)
2: x_train, x_test, y_train, y_test = train_test_split(features, target, test_size = 0.3)
3: #splitting the testing data into two sets (30%*50% = 15%)
4: x_train_meta, x_test_meta, y_train_meta, y_test_meta = train_test_split(x_test,
5:         y_test, test size = 0.5)
6: bm1 = RandomForestRegressor()                          #base model 1
7: from xgboost import XGBRegressor
8: bm2 = XGBRegressor()                                   #base model 2
9: bm1.fit(x_train, y_train)                              #train base model 1
10: bm2.fit(x_train, y_train)                             #train base model 2
11: y_pred_train_bm1 = bm1.predict(x_train_meta)          #base model 1 predictions on test set 1
12: y_pred_train_bm2 = bm2.predict(x_train_meta)          #base model 2 predictions on test set 1
13: y_pred_test_bm1 = bm1.predict(x_test_meta)            #base model 1 predictions on test set 2
14: y_pred_test_bm2 = bm2.predict(x_test_meta)            #base model 2 predictions on test set 2
15: import numpy as np
16: #feature array for meta model training, features are the base model predictions on test set 1
17: meta_train_data = np.column_stack((y_pred_train_bm1, y_pred_train_bm2))
18: #feature array for meta model testing, features are the base model predictions on test set 2
19: meta_test_data = np.column_stack((y_pred_test_bm1, y_pred_test_bm2))
20: meta_model = RandomForestRegressor()                  #meta model
21: meta_model.fit(meta_train_data, y_train_meta)         #train meta model
22: final_predictions = meta_model.predict(meta_test_data)  #generate meta model predictions
23: r2 = r2_score(y_test_meta, final_predictions)         #meta model performance
```

Hyperparameter Tuning

Since the performance of the models were initially assessed with the data by Li et all (2009), hyperparameter tuning for each model was left for the experimental data. Hyperparameter tuning is meant to maximize model performance on the specific data it's applied to. The preliminary data each model was initially assessed with might differ in important ways (e.g., noise levels, signal frequency) from the actual experimental data, which can make tuning on preliminary data less effective and misleading.

CHAPTER FIVE

RESULTS

Performance Metrics

A total of six predictive machine learning models were developed in Python 3.9, three solo base models, and three different stacking models consisted of the base models. 70% of the data was used for model training, while the remaining 30% was used for model validation (testing). One metric chosen to measure model performance was the coefficient of determination (R^2). The coefficient of determination is a statistical measure, which represents the proportion of the variance in the target variable that is explained by the input variables. R^2 can be calculated as shown below:

$$R^2 = 1 - \frac{RSS}{TSS} \qquad (Eq. 17)$$

where *RSS* is the sum of squared of the residuals and *TSS* is the total sum of squares. When a coefficient of determination is close to 0, the model explains none of the variance in the target variable. When a coefficient of determination value is 1, the model can perfectly explain the variance in the target variable, meaning a higher R^2 value indicates higher model accuracy. For regression problems, R^2 is a good metric to measure model accuracy, it quantifies how well models fit the data.

Model performance was also measured using the root mean squared error metric (RMSE). RMSE measures the average difference between a statistical model's predicted values and actual values. The resulting RMSE for each model is the average difference between the model's predicted tool wear values and actual tool wear values, in micrometers (μm). RMSE was chosen

as an accuracy metric because it's easily interpretable, as it is expressed in the same units as a dependent variable. RMSE is calculated as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2} \qquad (Eq. 18)$$

where n is the number of observations, Y_i is the actual value for observation i, and $\hat{Y}_i$ is the predicted value for observation i.

Optimal Hyperparameters

Optimal hyperparameters for each base and meta-model were chosen based on model performance. Hyperparameter tuning was conducted using GridSearchCV, a hyperparameter tuning method used in machine learning to find the optimal combination of hyperparameters. It works by training and evaluating a model based on possible values for each hyperparameter, typically using R^2 as a metric. GridSearchCV also uses k-folds cross validation, a cross validation method that evaluates each combination of hyperparameters. Cross validation is important because it helps assess the generalizability of each model, evaluating performance on multiple subsets of the data, which reduces the risk of overfitting. GridSearchCV iterates through a vector of potential hyperparameters, and for each hyperparameter combination, trains the model on a subset of the data and validates it on a different subset. For this case, 5 different subsets of data were chosen in the cross validation. R^2 values are calculated for each combination on each subset, the performance of the combination is determined by the average R^2 the combination produced on the subsets. The best hyperparameters for a given model are chosen based on the combination that produces the

highest average R^2. Code Block 12 below shows a chunk of code using GridSearchCV to determine optimal hyperparameters for a RF and XGBoost.

Code Block 12. GridSearchCV for a random forest

```
from sklearn.model_selection import GridSearchCV
model = RandomForestRegressor(random_state = 42, bootstrap = True)
param_grid = {
        'n_estimators': [100, 200, 300],      #number of trees
        'max_depth': [5, 10, 15],             #maximum depth of trees
        'min_samples_leaf': [1, 2, 4].        #minimum number of samples per leaf
        'min_samples_split': [2, 5, 10]}      #minimum number of samples per split
grid_search = GridSearchCV(estimator = model, param_grid = param_grid,
        cv = 5, n_jobs = -1, verbose = 2, scoring = 'r2')
grid_search.fit(x_train, y_train)    #training model on every hyperparameter combination
```

Determining the optimal hyperparameters for the ANNs required a different approach. GridSearchCV could not be used to find optimal hyperparameters for the ANN based on how the models were built. Another issue is Keras models are not inherently compatible with Scikit-Learn's estimator application programming interface. To address this, an algorithm with a nested loop was constructed to find the optimal number of nodes per hidden layer, along with the number of epochs (number of times the ANN sees the training data), and batch size (number of samples per training iteration). These were the only hyperparameters considered, because for regression problems, it is standard to use a random normal weight initializer, ReLU activation function, mean squared error loss function, and a back propagation function "adam" or "adamw". Code Block 13 shows a simplified example on how the optimal number of nodes per hidden layer were determined. The same process was used for batch size and epochs. Each combination was evaluated with R^2. Tables 2-7 show the best hyperparameters for each model.

Code Block 13. Optimal ANN nodes per hidden layer with nested loop

```
def best_parameters(x_train, y_train, x_test, y_test):
    results = pd.DataFrame(columns = ['r2', 'nodeslayer1', 'nodeslayer2'])
    num_nodes_layer1 = [5, 10, 15, 20]
    num_nodes_layer2 = [5, 10, 15, 20]
    for node_num1 in num_nodes_layer1:      #for number in hidden layer 1 node list
        for node_num2 in num_nodes_layer2:#for number in hidden layer 2 node list
            model = Sequential()                #build an ANN using the numbers
            model.add(Input(shape = (x_train.shape[1],)))
            model.add(Dense(units = node_num1))         #nodes = node_num1
            model.add(Dense(units = node_num2))         #nodes = node_num2
            model.add(Dense(1))
            model.compile()
            ann.fit(x_train, y_train)
            y_pred = ann.predict(x_test)
            r2 = r2_score(y_test, y_pred)    #performance metric for node combination
            results = pd.concat([results, pd.DataFrame(data = [r2,  #store results
                                                               node_num1, node_num2])])
    return results
best results = best_parameters(x_train, y_train, x_test, y_test) #call function with data
```

Table 2. ANN optimal hyperparameters (base model)

Hyperparameter	Optimal Hyperparameter
Hidden layers	2
Nodes per hidden layer	10
Epochs	100
Batch size	5
Weight initializer	random_normal
Activation function	ReLU
Loss function	MSE
Back propagation function	Adam

Table 3. RF Optimal hyperparameters (base model)

Hyperparameter	Optimal Hyperparameter
Number of trees	100
Maximum depth per tree	10
Minimum samples per leaf node	2
Minimum samples per split	5
Bootstrap sampling	True

Table 4. XGBoost optimal hyperparameters (base model)

Hyperparameter	Optimal Hyperparameter
Number of trees	200
Learning rate	0.1
Maximum depth per tree	5
Subsample per tree	80%
Column sample per tree	80%

Table 5. ANN optimal hyperparameters (meta-model)

Hyperparameter	Optimal Hyperparameter
Hidden layers	2
Nodes per hidden layer	Layer 1(10), Layer 2(15)
Epochs	100
Batch size	5
Weight initializer	random_normal
Activation function	ReLU
Loss function	MSE
Back propagation function	Adam

Table 6. RF optimal hyperparameters (meta-model)

Hyperparameter	Optimal Hyperparameter
Number of trees	100
Maximum depth per tree	10
Minimum samples per leaf node	2
Minimum samples per split	5
Bootstrap sampling	True

Table 7. XGBoost optimal hyperparameters (meta-model)

Hyperparameter	Optimal Hyperparameter
Number of trees	50
Learning rate	0.1
Maximum depth per tree	3
Subsample per tree	80%
Column sample per tree	80%

The best performing model has the highest R^2 value and the lowest RMSE value. Performance metrics were obtained by running each algorithm 10 times and taking the average of the metrics. The solo base model with the best performance was XGBoost, producing an R^2 value of 0.929 and RMSE of 73.948 µm. The stacking combination with the best performance had a RF and XGBoost as base-models, and an ANN as the meta-model. This stacking combination produced an impressive R^2 value of 0.955 and RMSE of 58.237 µm. In the stacking ensemble, the ANN machine learning method works particularly well as a meta-model for a variety of reasons. ANNs have the flexibility and ability to capture complex, non-linear relationships. The ANN can model highly nonlinear and multidimensional relationships between the base model predictions

and the target variable, making it a powerful meta-model. The performance of each base model can be seen in Table 8. The performance of each stacking model configuration can be seen in Table 9. Additionally, Figures 29-34 show each model's predicted tool wear values versus the actual tool wear values. The plots were made using Matplotlib (Hunter, 2007) (Code Block 14).

Code Block 14. Matplotlib performance plot for XGBoost

```
import matplotlib.pyplot as plt
plt.figure(figsize = (6, 6))                                    #setting the plot size
plt.scatter(y_test, y_pred, color = 'blue', alpha = 0.7)        #scatter plot true vs. predicted
plt.plot(y_test, y_test, color = 'red')                         #red diagonal reference line
plt.title('Tool Wear Prediction (XGBoost)')
plt.xlabel('Actual Flank Wear (µm)')
plt.ylabel('Predicted Flank Wear (µm)')
plt.grid(True)
plt.show()
```

Table 8. Base model performance

Model	R^2	RMSE (µm)
ANN	0.900	88.66
RF	0.906	85.585
XGBoost	**0.929**	**73.948**

Table 9. Stacked model performance

Meta-model	Base-model 1	Base-model 2	R^2	RMSE (µm)
XGB	RF	ANN	0.947	63.243
RF	XGB	ANN	0.938	68.732
ANN	**XGB**	**RF**	**0.955**	**58.237**

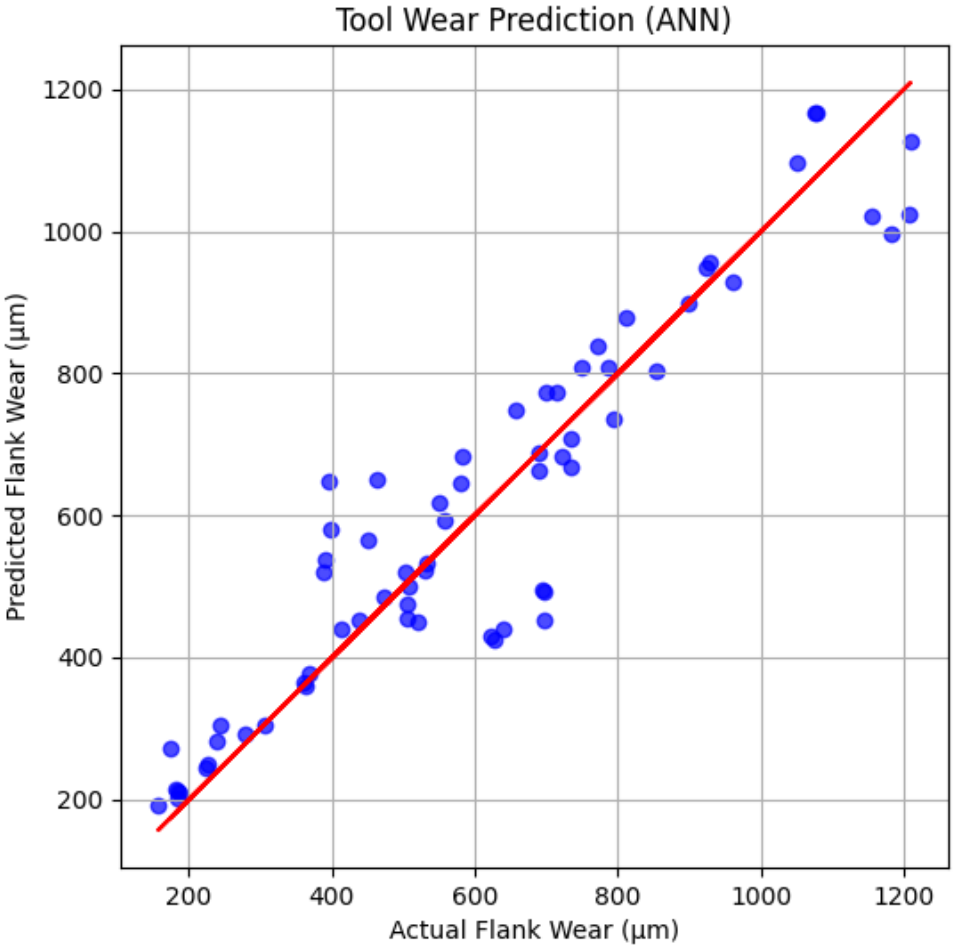


Figure 29. ANN predicted vs. actual tool wear values

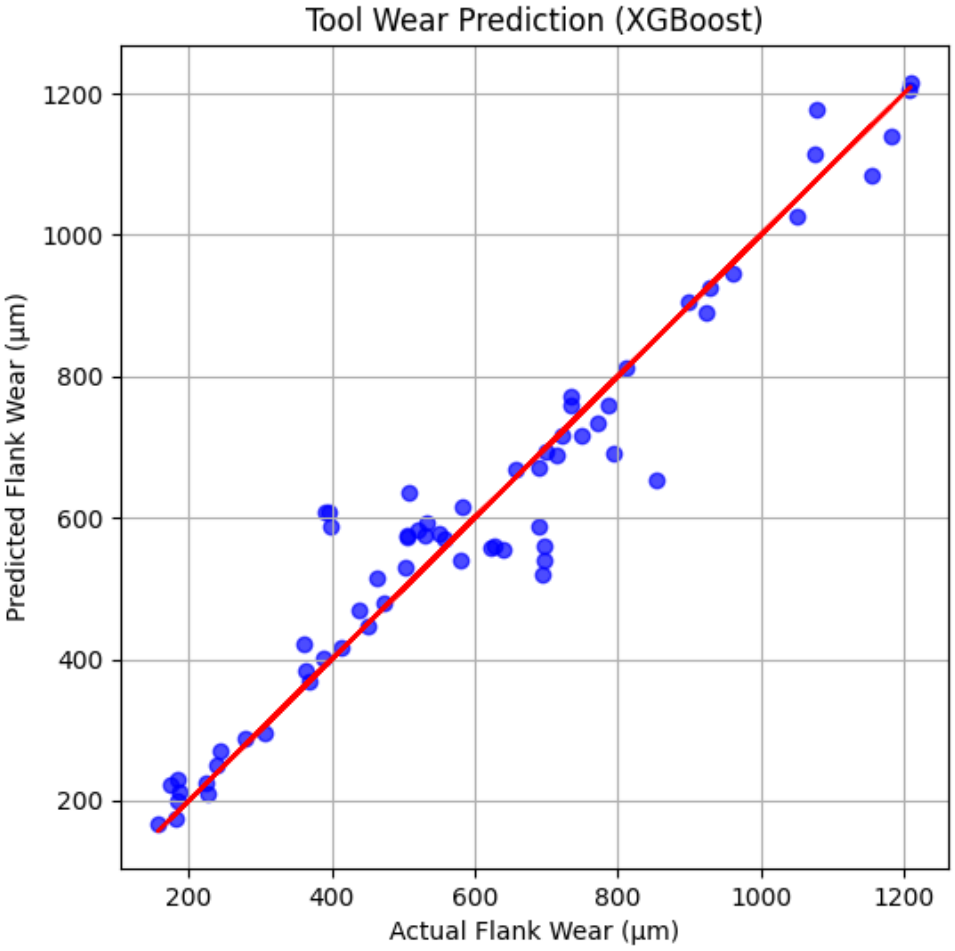


Figure 30. XGBoost predicted vs. actual tool wear values

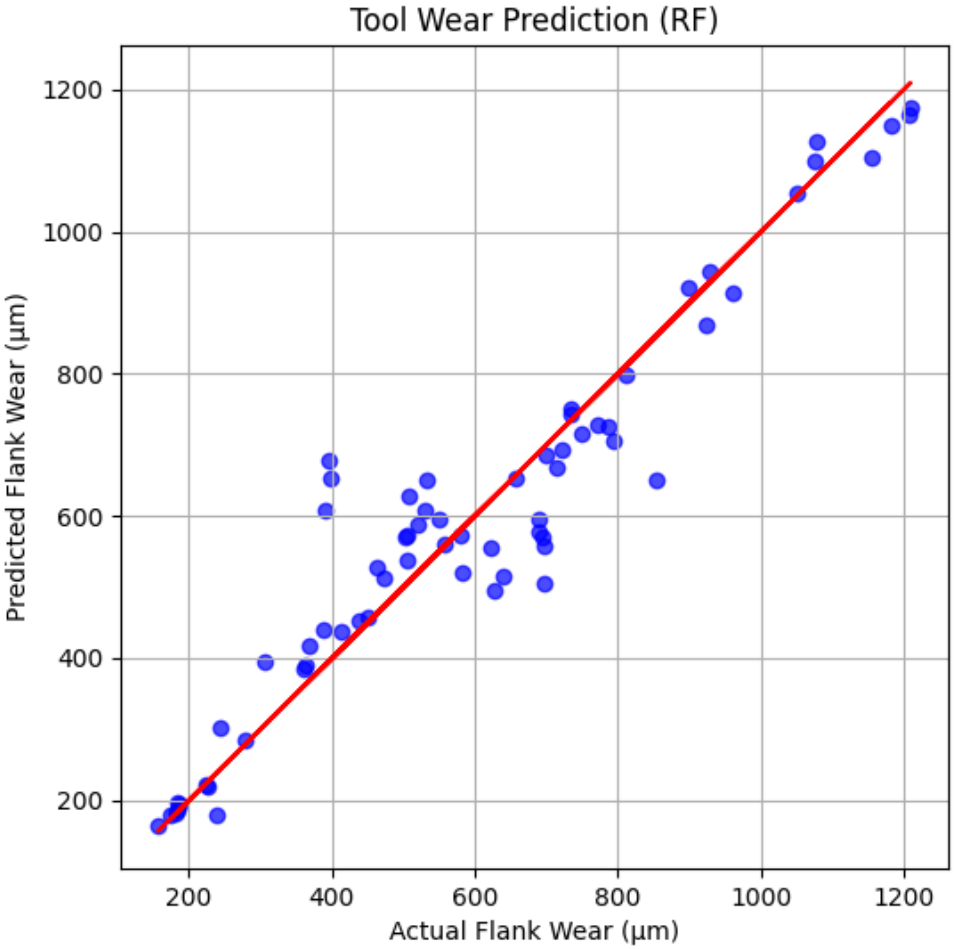


Figure 31. RF predicted vs. actual tool wear values

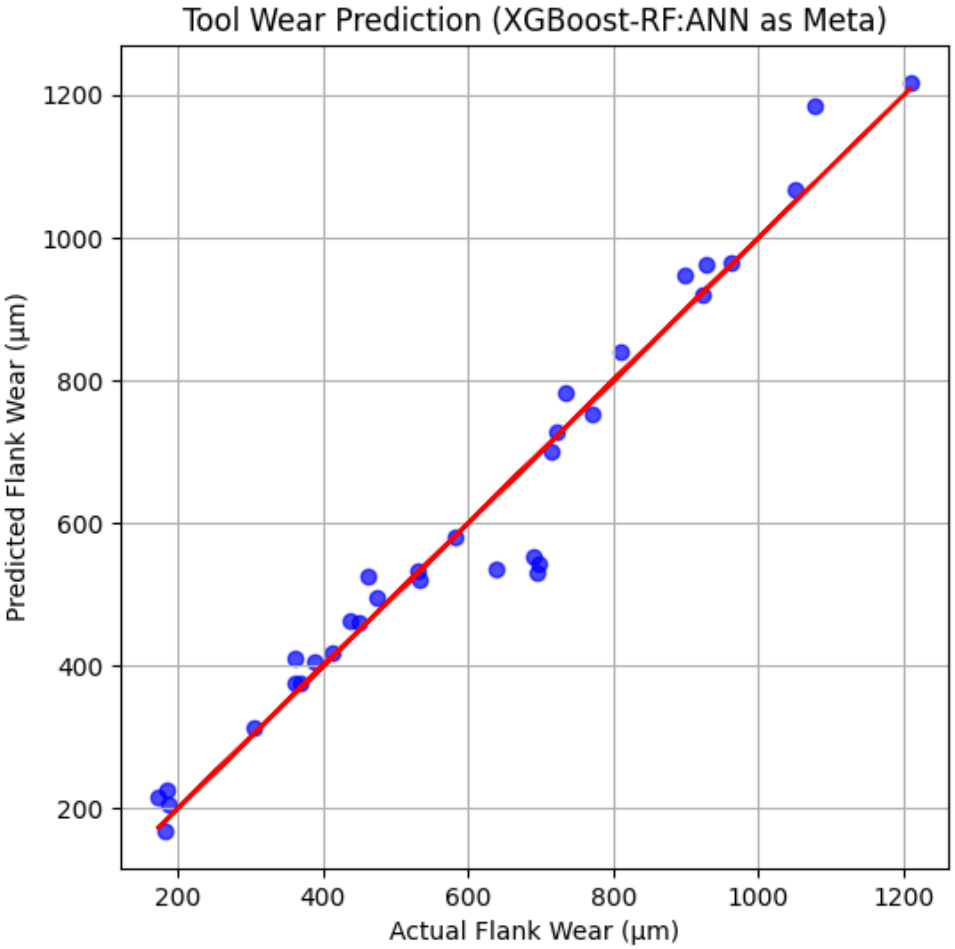


Figure 32. ANN meta-model predicted vs. actual tool wear values

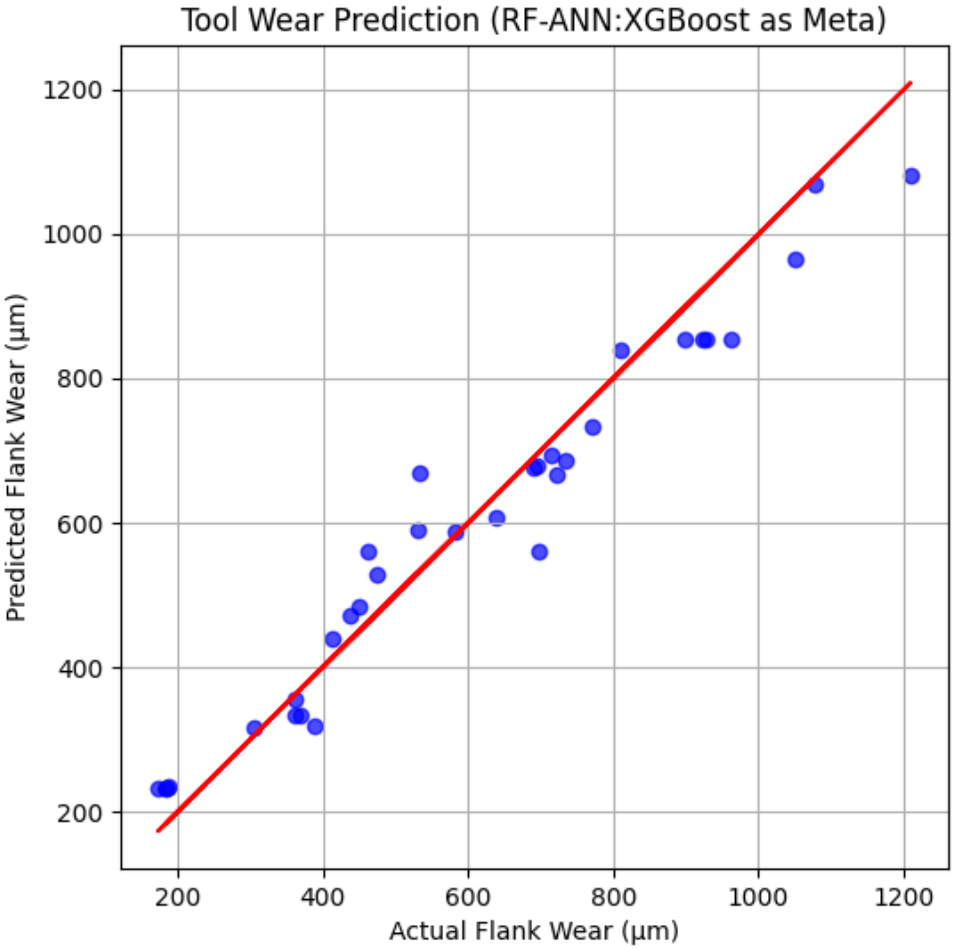


Figure 33. XGBoost meta-model predicted vs. actual tool wear values

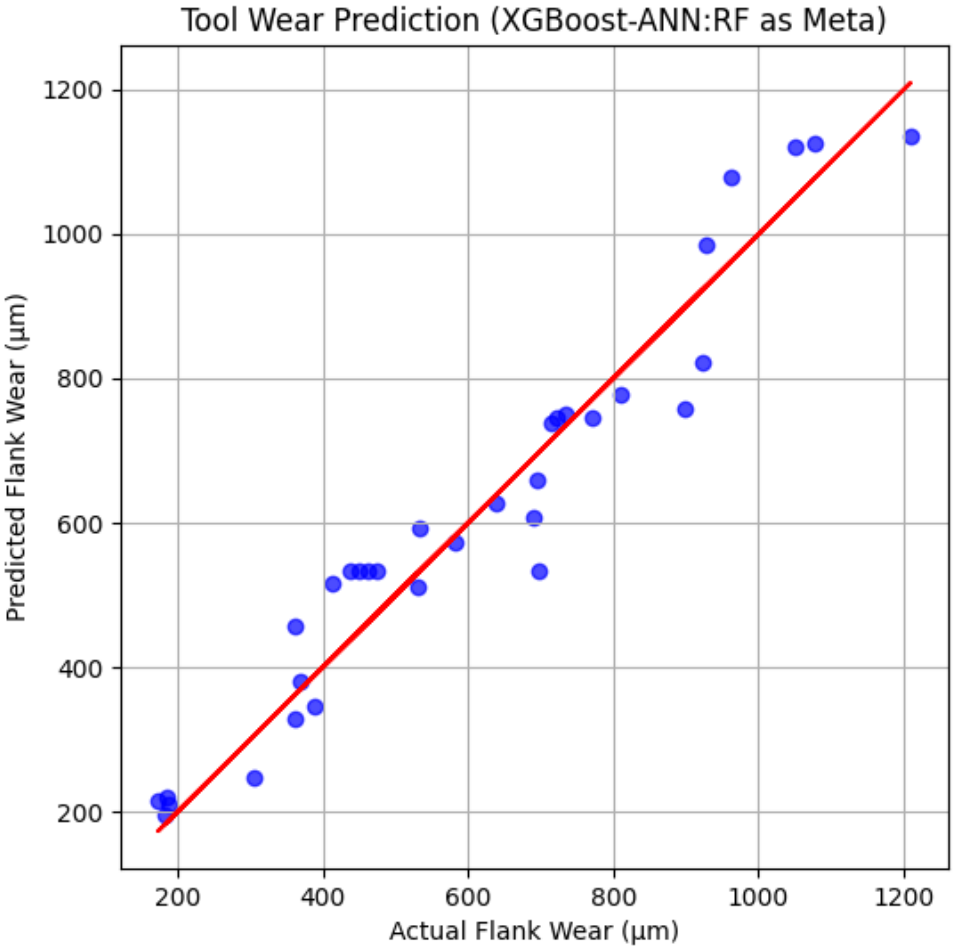


Figure 34. RF meta-model predicted vs. actual tool wear values

CHAPTER SIX

DISCUSSION AND CONCLUSION

Research Contribution

In this research, a three-level data fusion approach is developed to estimate tool flank wear for CNC face milling operations. Raw data was collected using three sensors, including a load cell, an accelerometer, and an AE sensor. Statistical features are then extracted from the raw data for machine learning model training. Previous research in TCM generally has used the two-level data fusion approach, i.e., raw data fusion and feature-level fusion. This research achieves fusion at the decision level, specifically using a machine learning stacking ensemble to fuse the predictions from various trained machine learning models. The experimental results have shown that the integration of decision-level fusion in a predictive model is highly effective. The main contribution of this research is summarized below.

- Raw data fusion: a load cell, an accelerometer, and an AE sensor were used to collect the cutting forces, vibrations, and sound waves during CNC face milling operations. Sensing signals from multiple sensors are collected and fused by a DAQ system.
- Feature-level fusion: 28 statistical features were extracted from the raw data. The statistical features play a vital role in the training process for the machine learning models.
- Decision level fusion: achieved by using a stacking ensemble, consisting of two base-level machine learning models trained on the selected statistical features, and one meta-model trained on the predictions from the base-models. The meta-model fuses the

predictions of the base models, producing a more robust, accurate estimation of tool wear.

- Results: the multisensory and multilevel data fusion approach achieved high accuracy in predicting tool wear, producing an R^2 value and root mean squared error (RMSE) of 0.955 and 58.237 μm, respectively. With appropriate modifications, this multilevel data fusion architecture can be used as a high performing predictive approach in various fields of prognostics and health management (PHM), such as tool condition monitoring (TCM).

The described three-level fusion approach demonstrates significant real-world applications, particularly in the PHM and TCM domains. Listed are some ways it can be applied:

- Advanced TCM in CNC machining: this approach could be implemented in industrial machining environments to monitor tool wear in real-time. By accurately predicting tool wear, manufacturers can schedule timely tool replacements, avoiding tool breakage or suboptimal machining quality. This approach could help with product quality, minimizing downtime, and reducing maintenance costs.
- Predictive maintenance in manufacturing: this framework could be extended to other critical components such as spindles, motors, and bearings. The approach could help in predicting the remaining useful life of components, enabling proactive maintenance before failures occur. This would help in enhancing the reliability of machines as well as operational efficiency.
- General PHM across various industries: the multi-level fusion approach could be applied to other PHM problems, such as fault detection in electric vehicles, robotics, or

renewable energy systems. This approach would enable high-accuracy predictions for complex systems where multiple signals influence performance. The multi-level fusion approach would help in reducing failure rates and improving reliability in various engineering domains.

Limitations

Quality of Cutting Tools

One limitation in this research involved inconsistencies in the quality and durability of the cutting tools. Notably, five tools experienced rapid wear, with significant chipping and degradation after only one or two passes, which likely introduced variability and outliers in the dataset. Conversely, other tools displayed higher resilience, one tool in particular lasting for more than twenty passes. A potential cause of this could be due to work hardening. Work hardening is caused by improper speeds and feeds during machining. It causes the workpiece to harden to the point where it becomes the same hardness as the cutting tool, which can lead to excessive tool wear and damage. This phenomenon is especially common in aluminum alloys. This variation in tool performance may have impacted model accuracy by introducing unpredictable wear patterns. Despite these inconsistencies and outliers, the models maintained high accuracy, demonstrating the robustness of machine learning algorithms. This suggests that even with interruptions, inconsistencies, or anomalies in data, predictive models can still deliver reliable results, underscoring the adaptability to real-world variability.

Number of Data Points

Another limitation in this research was time constraints for experimentation. Conducting a large number of trials to capture sufficient data on tool wear requires time and careful monitoring. The number of data points collected was sufficient, however more data points were desired. A larger dataset could have been beneficial for several reasons:

1) Improved model generalization – more data helps models learn broader ranges of patterns, making it less likely to overfit to certain observations. With a larger dataset, models can capture a more diverse set of tool wear patterns and operating conditions.
2) Enhanced accuracy – with more data, models can better understand the underlying relationships between dependent and independent variables. This leads to more accurate predictions, as models are provided a richer basis. More data would have also benefited the performance of the stacking ensemble. Because of the three-way split required for the stacking training and testing, more data would have enhanced the learning and generalization for the higher-level model, leading to more robust, accurate predictions.
3) Reduced noise impact – noise and outliers in data can skew model performance, especially with smaller datasets. By increasing the number of data points, models have more chances to learn the true signals rather than anomalies.
4) Increased statistical power for validation – a larger dataset allows for more bust validation and testing, leading to more reliable performance metrics.

Fixed Operating Parameters

Another limitation to this study was fixed machining parameters during the experimental procedures. While the developed models demonstrated high performance on an offline analysis with consistent machine operating parameters, the models are only applicable to certain conditions, meaning the performance would not be the same with different operating parameters. If the dataset the models learn from consists of specific wear levels or machine conditions, the models will struggle to predict wear levels when the machine is operating with different speeds, feeds, and cutting depths. Fixed parameters reduce the diversity of the training set, leading to models that might not learn certain patterns associated with different cutting conditions, which are critical for robust predictions. Varying the cutting conditions during experiments could help the applicability of the models in the real world in several ways. Introducing variability in the spindle speed, feed rate, and cutting depths during data collection allows the model to learn patterns across a wider range of machining conditions, enhancing abilities to predict tool wear in different scenarios. With data covering a range of machining conditions, the model can be applied to different machines, work piece materials, tool types, increasing usability in real-world scenarios. Real-world machining often involves parameter optimization to balance tool life, productivity, and surface finish. Training models with variable parameters simulates these complexities, making it more realistic for industrial applications.

Future Work

Real-Time Monitoring Systems

For future work, the development of a real-time monitoring system using multi-level fusion and machine learning techniques presents a promising avenue for further research. This would aim

to integrate the predictive approach developed in this study with live sensor data from an industrial environment to continuously monitor the health and wear of CNC tools and other critical machinery components. By continuously monitoring tool wear and the health of other machine components in real time, the system could identify early signs of deterioration, allowing for optimal interventions such as tool replacement or machine maintenance. In addition to monitoring the health of equipment, a real-time system could also support dynamic decision making in response to the health of equipment, such as wear on a milling tool. For a case involving monitoring tool wear, this approach could help optimize tool usage and prolong tool life without sacrificing product quality, thus creating a more resilient and economic analysis of the production process. For real-time monitoring systems to be feasible, an appropriate hyperparameter tuning method for machine learning algorithms must be explored. GridSearchCV has the drawback of assigning hyperparameters based on fixed data, which would not work in a case involving new data being introduced to machine learning models. Some research has been done to address these limitations of fixed hyperparameters, such as Bayesian optimization. Snoek et al. (2012) utilized Bayesian optimization for hyperparameter selection for machine learning algorithms, for cases that involved algorithms dealing with variable regimens. For real-time monitoring systems with machine learning to be effective, a proper hyperparameter tuning method that can deal with dynamic databases must be explored.

Adaptive Machine Learning

Another avenue for future work is the exploration of adaptive machine learning models. Traditional machine learning models are typically trained on a static dataset and can struggle to maintain accuracy when new data or different conditions are present. By contrast, adaptive models

are designed to self-update and adjust parameters as new data becomes available. This capability could improve the accuracy and robustness of tool wear prediction models by allowing models to evolve with dynamic conditions present in an industrial environment. This would be particularly beneficial in cases where the same model needs to adapt to different materials or production processes, making it suitable for a broader range of applications without the need for extensive training.

Scheduled Maintenance

The tool wear prediction models can be applied to integrate scheduled maintenance by transitioning from fixed interval maintenance to predictive maintenance. This involves using the models to predict the remaining useful life of tools or critical wear thresholds, allowing maintenance to only be performed when necessary. By defining tool wear thresholds, maintenance can be triggered when predicted tool wear approaches defined critical levels. The models can be integrated with maintenance systems to automate alerts and dynamically adjust maintenance schedules based on real time predictions. This approach not only eliminates unnecessary maintenance but also extends tool life and reduces downtime, making maintenance intervals more flexible and closely aligned with actual tool conditions. The integration of real-time predictions on the remaining useful life of tools into maintenance systems allows for continuous adjustment of intervals, ensuring that maintenance is neither premature or delayed, optimizing both efficiency and operating costs in manufacturing environments.

Predictive Modeling in a Variety of Industries

The final area for future research involves expanding the monitoring of health of systems and components across various industries. This research has demonstrated that the combination of

data fusion and machine learning results in a highly accurate predictive model for assessing the health of machine tooling. This robust framework can be adapted to monitor the health of a wide range of engineering systems, including but not limited to:

1) Manufacturing equipment – similar predictive models could be applied to other manufacturing machinery, such as lathes and injection molding machines, to monitor tool wear and other maintenance needs.
2) Aerospace systems – the integration of predictive analytics could enhance the monitoring of critical components in aircraft, such as engines and control systems, allowing for timely maintenance and improving safety and reliability.
3) Automotive systems – in the automotive industry, data fusion techniques could be used to assess the condition of various vehicle components, including engines, brakes, and other components, helping to prevent failures and optimize maintenance schedules.
4) Healthcare systems – personal medical devices are widely used and adopted by the population. In these devices, data fusion and machine learning can be leveraged to monitor the physical state of a patient. Data fusion and machine learning can also be leveraged to monitor the health of various medical equipment, such as MRI machines or ventilators, ensuring optimal performance and patient safety.

REFERENCES CITED

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., & Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (pp. 265—283). USENIX Association. https://www.tensorflow.org/

Akbari, F., Taghizadeh, S., Shvydka, D., Sperling, N., & Parsai, E. (2023) Predicting electronic stopping powers using stacking ensemble machine learning method. Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms, 538, 8—16. https://doi.org/10.1016/j.nimb.2023.02.023

Alajmi, M. S., & Almeshal, A. M. (2020). Predicting the Tool Wear of a Drilling Process Using Novel Machine Learning XGBoost-SDA. *Materials, 13(21).* https://doi.org/10.3390/ma13214952

Atlas, L., Ostendorf, M., & Bernard, G. D. (2002). Hidden Markov models for monitoring machining tool-wear. *2002 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE Xplore.* https://ieeexplore.ieee.org/document/860252/authors#authors

Bagga, P., Chavda, B., Modi, V., Makesana, M. A., & Patel, K. M. (2021). Indirect tool wear measurement and prediction using multi-sensor data fusion and neural network during machining. *Materials Today: Proceedings, 56(1), 51–55*. https://doi.org/10.1016/j.matpr.2021.12.131

Bajaj, A. (2023, April 26). Ensemble Models: How to Make Better Predictions by Combining Multiple Models with Python Codes (Explained). Medium. https://aryanbajaj13.medium.com/ensemble-models-how-to-make- better predictions-by combining-multiple-models-with-python-codes-6ac54403414e

Bhattacharyya, P., Sengupta, D., & Mukhopadhyay, S. (2007). Cutting force-based real-time estimation of tool wear in face milling using a combination of signal processing techniques. *Mechanical Systems and Signal Processing, 21(6), 2665–2683*.https://doi.org/10.1016/j.ymssp.2007.01.004

Boud, F., & Gindy, N. (2008). Application of multi-sensor signals for monitoring tool/workpiece condition in broaching. *International Journal of Computer Integrated Manufacturing, 21(6), 715–729.* https://doi.org/10.1080/09511920701253337

Breiman, L. (2001). Random Forests. *Machine Learning, 45, 5–32.* https://doi.org/10.1023/A:1010933404324

Brownlee, J. (2021, March 7). *XGBoost for Regression*. Machine Learning Mastery. https://machinelearningmastery.com/xgboost-for-regression/

Chen, J. C., & Chen, J. C. (2004). An artificial-neural-networks-based in-process tool wear prediction system in milling operations. *The International Journal of Advanced Manufacturing Technology, 25, 427–434.* https://link.springer.com/article/10.1007/s00170-003-1848-y

Chen, J. C., & Chen, W.-L. (1999). A tool breakage detection system using an accelerometer sensor. *Journal of Intelligent Manufacturing, 10, 187–197* https://doi.org/10.1023/A:1008980821787

Chen, S., Yin, Z., Zheng, L., & Yuan, J. (2024). Study of an ISSA-XGBoost model for milling tool wear prediction under variable working conditions. *The International Journal of Advanced Manufacturing Technology, 133, 2761–2774.* https://doi.org/10.1007/s00170-024-13811-5

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Association for Computing Machinery, New York, NY, USA, 785—794. https://doi.org/10.1145/2939672.2939785

Dhiraj, K. (2023, July 24). *Random Forest Regression in Python Using Scikit-Learn*. Comet. https://www.comet.com/site/blog/random-forest-regression-in-python-using-scikit-learn/

Dong, M., & He, D. (2007). Hidden semi-Markov model-based methodology for multi-sensor equipment health diagnosis and prognosis. *European Journal of Operational Research, 178(3), 858–878.* https://doi.org/10.1016/j.ejor.2006.01.041

Dutta, S. (2024, April 22). *Hyperparameter Tuning with Keras and TensorFlow*. Medium. https://medium.com/@sanjay_dutta/hyperparameter-tuning-with-keras-tuner-and-ensorflow-48ab5ea69cc5

Elangovan, M., Devasenapati, S. B., Sakthivel, N. R., & Ramachandran, K. I. (2011). Evaluation of expert system for condition monitoring of a single point cutting tool using principal component analysis and decision tree algorithm. *Expert Systems with Applications, 38(4), 4450–4459.* https://doi.org/10.1016/j.eswa.2010.09.116

Haber, R. E., Jimenez, J. E., Peres, C. Ronei., & Alique, J. R. (2004). An investigation of tool wear monitoring in a high-speed machining process. *Sensors and Actuators A: Physical, 116(3), 539–545.* https://doi.org/10.1016/j.sna.2004.05.017

Hall, D. L., & Llinas, J. (1997). An introduction to multisensor data fusion. *Proceedings of the IEEE, 85(1), 6—23.* doi: 10.1109/5.554205

Harris, C. R., Milman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., & Oliphant, T. E. (2020). *Array programming with NumPy. Nature. 585(7825), 357—362*. https://doi.org/10.1038/s41586-020-2649-2

Huang, Z., Zhu, J., Lei, J., Li, X., & Tian, F. (2019). Tool Wear Predicting Based on Multisensory Raw Signals Fusion by Reshaped Time Series Convolutional Neural Network in Manufacturing. *IEEE Access, 7*. https://ieeexplore.ieee.org/abstract/document/8928491

Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment. Computing in Science and Engineering, 9(3), 90—95*. https://doi.org/10.1109/MCSE.2007.55

Jiaa, C., & Dornfeld, D. (1998). A self-organizing approach to the prediction and detection of tool wear. *ISA Transactions, 37(4), 239–255.* https://doi.org/10.1016/S0019-0578(98)00035-4

Karayel, D. (2009). Prediction and control of surface roughness in CNC lathe using artificial neural network. *Journal of Materials Processing Technology, 209(7), 3125–3137*. https://doi.org/10.1016/j.jmatprotec.2008.07.023

Karna, S., & Sahai, R. (2012). An Overview on Taguchi Method. *International Journal of Engineering and Mathematical Sciences, 1, 11—18.*

Kepczak, N., Zgorniak, P., Lajmert, P., Roslik, R., & Sikora, M. (2020). Influence of machining parameters on the Polymer concrete milling process. *The International Journal of Advanced Manufacturing Technology, 106, 3017—3032*. https://doi.org/10.1007/s00170019-04811-x

Lau, W. S., Venuvinod, P. K., & Rubenstein, C. (1980). The relation between tool geometry and the Taylor Tool Life Constant. International Journal of Machine Tool Design and Research, 20(1), 29—44. https://doi.org/10.1016/0020-7357(80)90016-5

Lee, B., & Tarng, Y. (1999). Milling cutter breakage detection by the discrete wavelet transform fn1. *Mechatronics, 9(3), 225–234.* https://doi.org/10.1016/S0957-4158(98)00049-X

Lee, K., Park, S., Sung, S., & Park, D. (2019). Prediction of the CNC Tool Wear Using the Machine Learning Technique. *2019 International Conference on Computational Science and Computational Intelligence (CSCI).* https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9070992

Li, X., Lim, B., Zhou, J., Huang, S., Phua, S., Shaw, K., & Er, M. (2009). Fuzzy Neural Network Modelling for Tool Wear Estimation in Dry Milling Operation. *Annual Conference of the Prognostics and Health Management Society (PHM), San Diego, CA, 1—11.*

Liang, S., & Dornfeld, D. (1989). Tool Wear Detection Using Time Series Analysis of Acoustic Emission. *Journal of Manufacturing Science and Engineering, 111(3), 199-205.* https://doi.org/10.1115/1.3188750

Morshed-Bozorgdel, A., Kadkhodazadeh, M., Anaraki, M., & Farzin, S. (2020). A Novel Framework Based on the Stacking Ensemble Machine Learning (SEML) Method: Application in Wind Speed Modeling. Atmosphere, 13(5). https://doi.org/10.3390/atmos13050758

Niaki, F., Michel, M., & Mears, L. (2016). State of health monitoring in machining: Extended Kalman filter for tool wear assessment in tuning of IN718 hard-to-machine alloy. *Journal of Manufacturing Processes, 24(2), 361-369.* http://dx.doi.org/10.1016/j.jmapro.2016.06.015

Ozel, T., & Karpat, Y. (2005). Predictive modelling of surface roughness and tool wear in hard turning using regression and neural networks. *International Journal of Machine Tools and Manufacture, 45(5), 467–479.* https://doi.org/10.1016/j.ijmachtools.2004.09.007

Palanisamy, P., Rajendran, I., & Shanmugasundaram, S. (2007). Prediction of tool wear using regression and ANN models in end-milling operation. *The International Journal of Advanced Manufacturing Technology, 37, 29-41.* https://doi.org/10.1007/s00170-007 0934-y

Pedregosa, F., Varoquax, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825—2830. https://arxiv.org/abs/1201.0490

Rathod, N., Chorpra, M. K., Chaurasiya, P., & Vidhate, U. S. (2022). Optimization of Tool Life, Surface Roughness and Production Time in CNC Turning Using Taguchi Method and ANOVA. Annals of Data Science, 10(2). http://dx.doi.org/10.1007/s40745-022-00423-7

Schwabacher, M., & Goebel, K. (2007). A survey for artificial intelligence for prognostics. ResearchGate. https://www.researchgate.net/publication/228346641_A_survey_of_artificial_intelligence _for_prognostics

Si, X., Wang, W., Hu, C., Chen, M., & Zhou, D. (2013). A Wiener-process-based degredation model with a recursive filter algorithm for remaining useful life estimation. Mechanical Systems and Signal Processing, 35(2), 219—237. https://doi.org/10.1016/j.ymssp.2012.08.016

Siddhpura, A., & Paurobally, R. (2013). A review of flank wear prediction methods for tool condition monitoring in a turning process. *International Journal of Advanced Manufacturing Technology, 65, 371--393.* https://doi.org/10.1007/s00170-012-4177-1

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. *Advances in Neural Information Processing Systems (NIPS 2012), 25*.https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd Abstract.html

Teti, R., Jemielniak, K., O'Donnell, G., & Dornfled, D. (2010). Advanced monitoring of machining operations. *CIRP Annals, 59(2), 717—739*. https://doi.org/10.1016/j.cirp.2010.05.010

The pandas development team. (2023). *pandas: Powerful Python data analysis toolkit (version 2.2.3)* [Computer software]. https://pandas.pydata.org/

Wang, H., Wang, S., Sun, W., Xiang, J. (2024). Multi-sensor signal fusion for tool wear condition monitoring using denoising transformer auto-encoder Resnet. *Journal of Manufacturing Processes, 124, 1054- 1064.* https://doi.org/10.1016/j.jmapro.2024.07.002

Wu, D., Jennings, C., Terpenny, J., Gao, & R., Kumara, S. (2017). A Comparative Study on Machine Learning Algorithms for Smart Manufacturing: Tool Wear Prediction Using Random Forests. Journal of Manufacturing Science and Engineering, 139(7). https://doi.org/10.1115/1.4036350

Yesilyurt, I., & Ozturk, H. (2006). Tool condition monitoring in milling using vibration analysis. International Journal of Production Research, 45(4), 1013-1028. http://dx.doi.org/10.1080/00207540600677781

Zhang, C., Yao, X., Zhang, J., & Jin, H. (2016). Tool Condition Monitoring and Remaining Useful Life Prognostic Based on a Wireless Sensor in Dry Milling Operations. *Sensors, 16(6), 795.* https://doi.org/10.3390/s16060795