UiA University of Agder

Bachelor

# Classification and Detection of Bearing Faults based on Vibration Signals using Transformers and Tsetlin Machines

HPR/D-24-006
SIMON MAKASSIOUK, KNUT SELSTAD and SIGURD SØBERG

SUPERVISOR
Lei Jiao

**University of Agder, 2024**
Department of Information and Communication Technology
Faculty of Engineering and Science

## Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

| | | |
|---|---|---|
| 1. | Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen. | Ja |
| 2. | **Vi erklærer videre at denne besvarelsen:**<br>• Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands.<br>• Ikke refererer til andres arbeid uten at det er oppgitt.<br>• Ikke refererer til eget tidligere arbeid uten at det er oppgitt.<br>• Har alle referansene oppgitt i litteraturlisten.<br>• Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse. | Ja |
| 3. | Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31. | Ja |
| 4. | Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert. | Ja |
| 5. | Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk. | Ja |
| 6. | Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider. | Ja |
| 7. | Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet. | Nei |

## Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

| | |
|---|---|
| Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering: | Ja |
| Er oppgaven båndlagt (konfidensiell)? | Nei |
| Er oppgaven unntatt offentlighet? | Nei |

# Acknowledgements

We would like to give a big thanks to our supervisor, Professor Lei Jiao, for his invaluable guidance throughout the course of this project.
Another thanks goes out to Lidia Kouzginova, Yuri Makassiouk, and Dag Søberg for their assistance in reviewing the thesis and providing advice on readability and language.

The AI tool ChatGPT by OpenAI was utilized for advice on text structure and assistance with code implementation.

# Abstract

This thesis is motivated by the requirement for efficient fault detection and classification in bearings within industrial machinery. Two advanced machine learning models, namely, the transformer-based deep learning model and the Tsetlin Machine (TM), are explored, with the focuses on optimizing and evaluating the performance of these models using two benchmark datasets: the National Aeronautics and Space Administration (NASA) Bearing Dataset and the Case Western Reserve University (CWRU) Bearing Data Center. The results show that both models achieved exceptionally high accuracy in both detecting and classifying bearing faults, proving their potential to be applied in the industry. In more detail, the transformer model is proficient at capturing long-range dependencies within time-series data, leading to high classification results of 100 % for the best single training run in the multiclass classification task using the CWRU dataset, and 99.86 % for the best single training run in the binary classification task using the NASA dataset. Similarly, the TM, in addition to its potential for interpretability, gave classification results of 99.83 % and 99.67 % for the single best training runs for the two aforementioned tasks, respectively.

The contribution of this thesis lies in its detailed evaluation of these advanced models and their potential for application to this real-world problem, providing a path for future studies to explore further implementations, as well as the addition to the small pool of research around the different applications of TMs.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

In the world of industrial machinery, a lot of downtime is often caused by bearing failures and maintenance of those failures. According to a representative from National Oilwell Varco (NOV) [50], prediction and detection of those faults are currently done manually by an expert through spectral analysis and time-domain monitoring. These manual methods can sometimes be insufficient as subtle changes in the signal can be hard to see, and a small mistake could bear significant unnecessary costs for a company. Early detection and classification of faults has the potential to reduce downtime, saving a lot of money. A report from Siemens [43] on "The True Cost of Downtime 2022" shows the dramatic financial impacts of unplanned downtime. It notes that among Fortune Global 500 companies, downtime now costs approximately 11 percent of their yearly turnover, which translates to almost 1.5 trillion US dollars annually [43]. This is an increase from two years earlier when the number was 864 billion US dollars.

Additionally, rolling bearing failures, which are known to account for 40 % to 90 % of all rotating machinery failures, significantly contribute to these economic losses by causing shutdowns and catastrophic accidents, thereby are the reason behind a big part of the financial burden on industries [38].

In addition to this downtime being a major cost for a lot of companies, the cases of bearing failure that are easy to see by eye can also be easy for a machine learning model to classify, suggesting a high potential for automation in the fault detection process. This is supported by insights from IBM, which describe how supervised learning models are particularly adept at recognizing and classifying patterns that are visibly discernible, supporting the potential for machine learning to automate complex identification tasks in industrial applications [26].

## 1.2 Problem Statement

Conventional manual diagnostics that are currently being used for the detection of bearing failures in industrial machinery [50] are increasingly turning obsolete with the advancement of Artificial Intelligence (AI). Inaccuracies and the lack of early detection in these manual processes can lead to considerable financial impacts. This thesis explores the performance of two machine learning models - the transformer, and the less explored TM - as potential enhancements to fault detection systems. The TM, with its unique proposition of interpretable machine learning, has not yet been fully researched or applied to signal analysis tasks, representing a gap in the ongoing research of its performance. There are two problems this thesis aims to address: one is classifying the type of bearing faults present in vibrational

data, and the other is the binary task of detecting whether a bearing is faulty or not.

## 1.3 Research Method

This is the structured approach employed in this thesis to take on the issue of bearing fault classification and detection presented by NOV. The research methodology was designed to address both practical and academic requirements, ultimately leading to the development of an innovative solution. The methodological framework of the work is organized as follows:

1. **Identification of the Problem:** Start the project by defining the practical problem presented by NOV, emphasizing the need for an automated solution to monitor bearing faults.

2. **Comprehensive Understanding:** Acquire a deep understanding of the domain through extensive literature review and analysis of existing datasets used in the field.

3. **Innovative Solution Development:** Propose a novel approach by employing an AI model to classify bearing faults.

4. **Implementation and Testing:** Implement the proposed models and evaluate their performance using available datasets, ensuring the solution meets the required benchmarks. This is done using an iterative approach, testing models and adjusting configurations based on intermediate results.

5. **Validation of the Solution:** Argue for the validity of the solution, presenting the results and explaining the performance.

6. **Theoretical Contributions and Applicability:** Lay out the theoretical advancements and the broader applicability of the solution in industrial settings.

## 1.4 Research Objectives

Solutions to the problems will be developed using both the transformer and the TM architectures. The primary aim is to look for the potential of these advanced models to outperform traditional methods in classifying and detecting bearing faults. The categorization of common bearing faults through vibrational signal analysis will be done using the CWRU Bearing Dataset [8], and the determination of bearing condition; either faulty or healthy, will be done using the NASA Bearing Dataset [49]. Applying a transformer or a TM to these problems presents two potentials: saving considerable resources for companies, and pioneering a new application area for these types of machine learning models.

## 1.5 Scope of Research

We have not found any information about transformers and TMs used for this application. The TM have been especially underexplored in practical applications so far, specifically in the context of fault detection and signal analysis. This is the innovative core of the research in this thesis. The performance of these models will be evaluated in a comprehensive way to investigate their applicability on these tasks. This will be done through an iterative development process, followed by thorough testing and benchmarking.

## 1.6 Significance of the Study

By investigating both cutting-edge and underexplored machine learning models, this thesis has a goal of advancing predictive maintenance strategies and establishing a new frontier in the potential application of TMs. The automation of fault detection processes stands to significantly benefit the field of industrial maintenance.

## 1.7 Thesis Organization

The thesis is structured as follows:

- **Chapter 2 - Background**: Reviews the relevant literature, providing context to the problem, the datasets used, and an overview of the machine learning models examined in this research.
- **Chapter 3 - Methods and System Design**: Details the system design, experimental setup, and methodology for evaluating model performance.
- **Chapter 4 - Results**: Presents the results of the research, highlighting the performance of different setups of the proposed machine learning approaches in fault detection and classification.
- **Chapter 5 - Discussion**: Discusses alternative configurations explored, limitations and potential for improvement, as well as future research paths.
- **Chapter 6 - Conclusion**: Concludes the thesis, summarizing the findings and contributions.
- **A - Appendix**: Contains additional information about the development environment and tools used in this study.

# Chapter 2

# Background

## 2.1 Problem Description

In the past, various machine learning models, including Convolutional Neural Networks (CNNs) using transfer learning [30], deep convolutional transfer learning networks [22], bidirectional long short-term memory (BiD-LSTM) [13] and EfficientNets [39] have been used to attempt to solve the problems discussed in this thesis with varying degrees of success. These models, however, may struggle with temporal complexities and detecting the subtle patterns present in vibration data. The challenge is particularly significant in contexts where sequential data integration is critical, as discussed in an article that explains the struggles of pattern recognition systems with complex data patterns that include temporal elements [20].

- **CNNs:** While they are robust in feature extraction from spatial and frequency-domain data, CNNs do not inherently take sequential order into account, making them potentially less effective for certain time-series analysis (or spectrogram analysis since it is also sequential).

- **BiD-LSTMs:** They are designed to handle sequence prediction problems better by remembering both past and future information, but they can be computationally intensive and prone to overfitting, especially with limited data as is typical in practical settings, a challenge documented in studies such as those published in "Physica D: Nonlinear Phenomena" [42].

- **EfficientNets:** EfficicientNets are known for their efficiency and scalability, which is important for handling large datasets. They are primarily used for image recognition tasks, but might not be as effective in capturing long-range dependencies across time, a task where transformers excel due to their self-attention mechanism which allows global context incorporation at every layer of the model.

Introducing a machine learning model such as the transformer or a TM could be a more reliable option for solving this problem. Transformers are designed to process sequential data, potentially improving the detection and classification accuracy of bearing faults by capturing temporal dependencies that traditional models may not detect. On the other hand, TMs are known for their interpretability and high efficiency. They utilize propositional logic and Boolean inputs to make decisions, providing a better understanding of their decision-making process. This attribute can be highly valuable in industrial settings where explainability can be as useful as accuracy.
There are two main objectives of this research:

- **Fault Classification:** To classify the type of fault present in bearings based on vibration signals, focusing on the common types of faults. The types of faults that are common are ball faults, inner race faults and outer race faults.

- **Fault Detection:** To determine whether a vibration signal represents a faulty or a functioning bearing. This is essentially a binary classification problem.

The bearing fault classification (multiclass classification) part will be done using the CWRU Bearing Data Center, and the fault detection (binary classification) will be done using the NASA Bearing Dataset, discussed in the subsequent sections.

## 2.2 Datasets

This thesis utilizes two datasets, the NASA Bearing Dataset [49] and CWRU dataset [8], which are widely used benchmark datasets for this type of problem. They will be used for training, validating and testing the proposed models. Given the limitations in availability of alternative datasets, these two datasets are the optimal options for this research. The CWRU dataset provides a broad range of fault types with multiple experiments for each type, and the NASA dataset consists of run-to-failure experiments with organic (instead of artificially induced) faults.

### 2.2.1 NASA Bearing Dataset

In a comprehensive vibration analysis experiment conducted by NASA, PCB 353B33 High Sensitivity Quartz ICP accelerometers were mounted on a rotating machine to monitor vibration data [49]. This setup facilitated a precise extraction of time series vibration information for assessing bearing integrity and performance over time. The experiment was first set up with accelerometers configured to track both the x-axis and y-axis vibrations, but this was only done on the first of the three experiments. Subsequent tests were modified to monitor x-axis vibrations only [49]. All data in this dataset is in the form of a time-series, meaning that no pre-processing has been done on the data and it is the raw signal output from the accelerometers used.

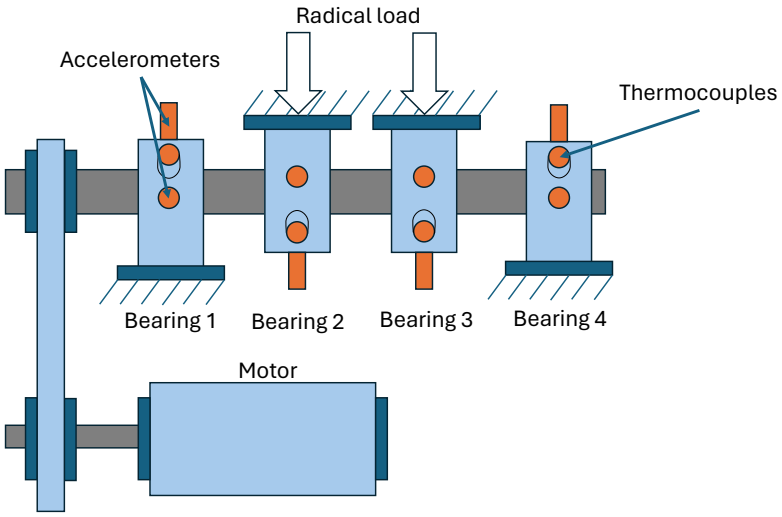



Figure 2.1: NASA's bearing test rig and sensor placement illustrated.

#### Experimental Setup and Procedure

The experimental apparatus consisted of four bearings mounted on a shaft, shown in Figure 2.1, driven by an AC motor to maintain a constant rotation speed of 2000 RPM. To avoid potential bearing faults, a force lubrication method was used, enhancing the longevity and reliability of the bearings under test. The shaft was subjected to a substantial radial force of 6000 lbs (2721.554 kilograms), applied via a spring mechanism, to simulate operational stress conditions. The tests were designed to continue until a bearing failure occurred, in

order to obtain a timeline of bearing performance from healthy to failure states, with each bearing ultimately surpassing its designed lifetime before failing [49]. This timeline is useful for training a model that identifies when exactly a bearing fails and generalizes across different stages of the bearing lifetime.

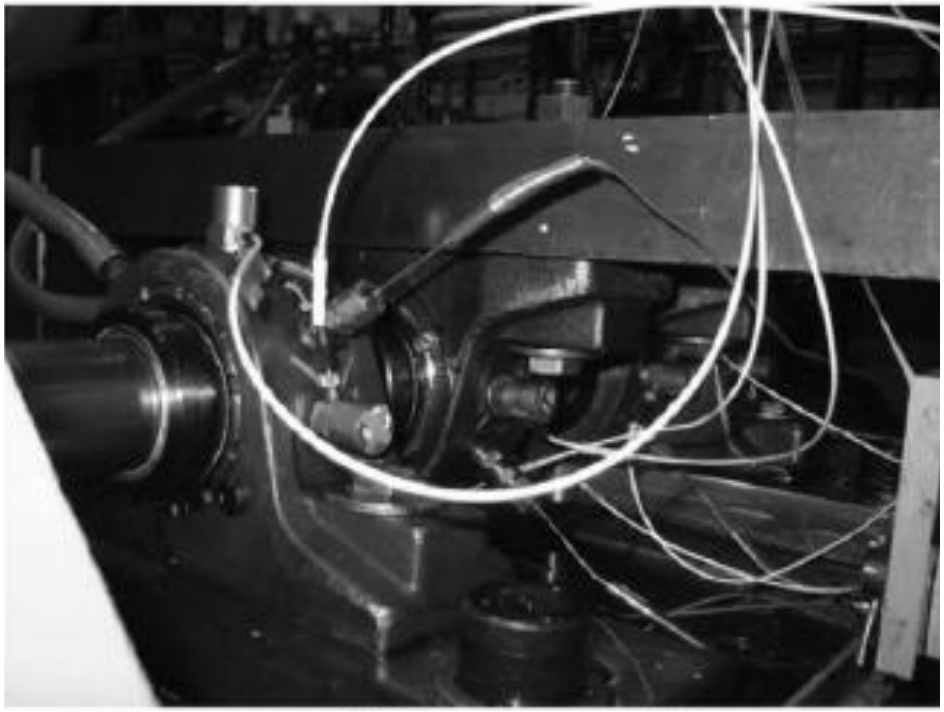


Figure 2.2: Photo of the NASA's bearing test rig. Retrieved from [49].

**Dataset Structure and Findings**

The experiments generated three distinct series, each corresponding to a specific test-to-failure scenario:

- **Experiment 1:** Featured eight channels, recording vibrations across two axes for four bearings. The initial 43 files were captured every 5 minutes, transitioning to a 10-minute interval for subsequent recordings, culminating in 2156 files. This test concluded with an inner race defect in bearing 3 and a roller element defect in bearing 4.

- **Experiment 2:** Simplified to monitor only the x-axis vibrations, this dataset includes 984 files recorded at 10-minute intervals. The test ended with an outer race failure in bearing 1, demonstrating the variance in failure types common in different bearing components.

- **Experiment 3:** Consisting of 6324 files, also recorded every 10 minutes and focusing solely on x-axis vibrations. This test ended with an outer race failure in bearing 3.

This dataset has a comprehensive variation in fault evolution over time with different types of faults, making it particularly suitable for training the proposed transformer and TM models.

Note that for experiment 1 and 3 there are multiple gaps in the capturing of the data. In dataset 1 there are a total of 24 gaps, the longest being 149 hours long. For the third dataset

the number of gaps is only 4 and the longest gap in time is 7 hours. The reason for this is that experiments were set on pause at the end of workdays and resumed the next workday.

### 2.2.2 CWRU Bearing Data Center

The CWRU dataset [8] is a collection of data specifically designed for the study of bearing faults within mechanical systems. Unlike the NASA dataset, the CWRU dataset was made using a 2 hp Reliance Electric motor, with data measurements captured both near to and further from the bearings. The dataset consists of bearing faults artificially induced through electro-discharge machining, presenting faults of varying depths:

- 7 mils,
- 14 mils,
- 21 mils,
- 28 mils,
- 40 mils.

One mil equates to 0.001 inches.

The dataset further categorizes faults across two different types of bearings, with SKF bearings hosting the three smallest faults and NTN bearings the two largest. However, the dataset consists of two main parts as data was collected both using 12 000 Hz sampling frequency, and using 48 000 Hz sampling frequency. On the 48 000 Hz part, data from 28 mils and 40 mils fault depth is not included, so all the bearings in this part are from SKF. The variation in fault size in the 48 000 Hz part of the dataset still provides a detailed spectrum of training data as there are 36 fault conditions in total.

The 36 fault conditions are evenly distributed across three primary fault types:

- outer raceway faults,
- inner raceway faults,
- ball faults.

There are 12 instances for each of these three fault categories. These 12 are further evenly divided into three of the five total types of fault depths: 7 mils, 14 mils and 21 mils. This is specifically in the 48 000 Hz part, while the layout of the 12 000 Hz part is different. For training the models in this thesis, however, only the 48 000 Hz part of the dataset was used. We used this data to align with the version of the dataset used in a recent paper on using AI for bearing fault classification [39] (see [10] for actual usage), a thesis that we have used as a baseline for comparisons when working towards developing our models. Like the NASA dataset, the CWRU dataset is also raw time-series data with no preprocessing done to it.

#### Data Acquisition and Relevance

Data acquisition involved mounting accelerometers on the housing's top and, in some setups, on the motor support plate, capturing vibration signals with a 16 channel DAT recorder. Meanwhile, operational parameters such as speed and horsepower were recorded by hand using a torque transducer/encoder. These measurements were done under varied load conditions on the motor, ranging from 0 hp to 4 hp, although the experiments with 0 hp load weren't captured the same way as the others in the 48 000 Hz part of the dataset (they have only half the number of datapoints with unspecified reason), and was therefore excluded

when training and testing the proposed models.

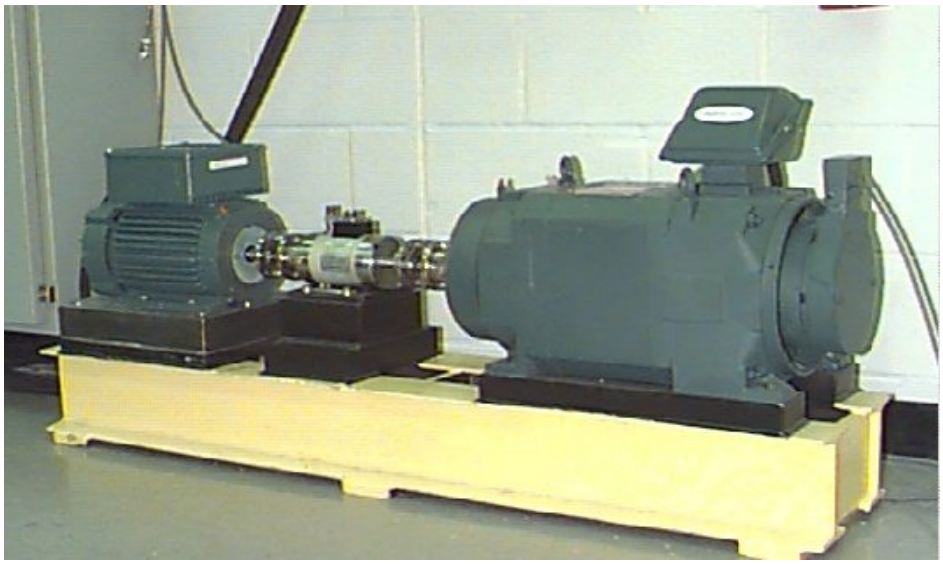


Figure 2.3: Photo of CWRU's bearing test rig. Retrieved from [8].

Out of the three types of faults in the dataset; inner raceway, outer raceway and ball faults, the outer raceway faults are described as "stationary", since the outer raceway doesn't rotate in relation to the accelerometers [8]. The experimental design considered the fault's position relative to the bearing's load zone, conducting tests with faults at 3 o'clock (directly in the load zone), 6 o'clock (orthogonal to the load zone), and 12 o'clock positions. The bearings with 14 mils outer race fault only had data from the 3 o'clock position, so for this thesis, only the 3 o'clock positions were used for all bearings with outer race faults.

## 2.3 Preprocessing

### 2.3.1 Importance and Methods of Data Splitting for AI Model Validation

Creating distinct training and testing datasets is essential in the development of AI models. This process is important for evaluating a model's performance, generalization ability, and for preventing overfitting. By splitting the data, the model is exposed to completely unseen data during testing, which simulates real-world application. This approach not only tests how the model would perform in practical scenarios but also checks for overfitting, a condition where the model learns not just the underlying patterns but also noise and irrelevant details from the training data. Consequently, while the model may perform well on the training data, its performance often degrades on the test data. By maintaining a separate test set, it is possible to ensure that the model is genuinely learning and not merely memorizing the data.

#### Strategy for Data Splitting

Effective data splitting strategies are important for a robust model evaluation process. One common method is the random split, where data points are randomly assigned to either the training set or the test set. Typically, a split ratio of 70 % to 80 % for training and 20 % to 30 % for testing is used.

$$\text{Training set size} = 0.8 \times \text{Total data size.} \tag{2.1}$$

$$\text{Testing set size} = 0.2 \times \text{Total data size.} \tag{2.2}$$

#### Benchmarking and Model Validation

Ultimately, the goal of any AI model is to perform well on new, previously unseen data. Therefore, a reliable benchmark that can be trusted is necessary to evaluate the AI's performance fairly. For various AI applications, especially in areas involving large language models like GPT-4 [4], Claude 3 [6], and Llama 3 [31], standardized benchmark tests are utilized to compare performance objectively. An example of such a platform is Promptbase [32].

### 2.3.2 Data Normalization Techniques

Data normalization is an essential preprocessing step in data analytics and machine learning. It enhances the quality of each data point and ensures uniformity across the dataset. One common approach to normalization is standardization, which often involves rescaling the data to follow a standard normal distribution [27].

#### Standardization

Standardization typically employs the Z-score normalization method, where $\mu$ represents the mean and $\sigma$ denotes the standard deviation of the data points, $X$. This method adjusts each data point based on the overall distribution, as shown in Equation (2.3):

$$X_{\text{standardized}} = \frac{X - \mu}{\sigma}. \tag{2.3}$$

This technique is particularly valuable as it reduces the impact of outliers, which are data points significantly different from others. In their raw form, outliers can skew the analysis, leading to potentially biased results. Standardization mitigates this by centering the data around the mean and scaling according to the standard deviation, thus bringing outliers closer to the norm.

#### Min-Max Scaling

Another prevalent technique is min-max scaling, often simply referred to as normalization [27]. This method rescales the data to a specified range, typically from 0 to 1, ensuring that no values fall below 0. It is executed by subtracting the minimum value from all data points and then dividing by the range of the data, which is the difference between the maximum and minimum values, as detailed in Equation (2.4):

$$X_{\text{normalization}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}. \tag{2.4}$$

Both standardization and min-max scaling are often beneficial for preparing data for machine learning models, as they standardize the range of independent variables or features of data by ensuring that features have a similar scale.

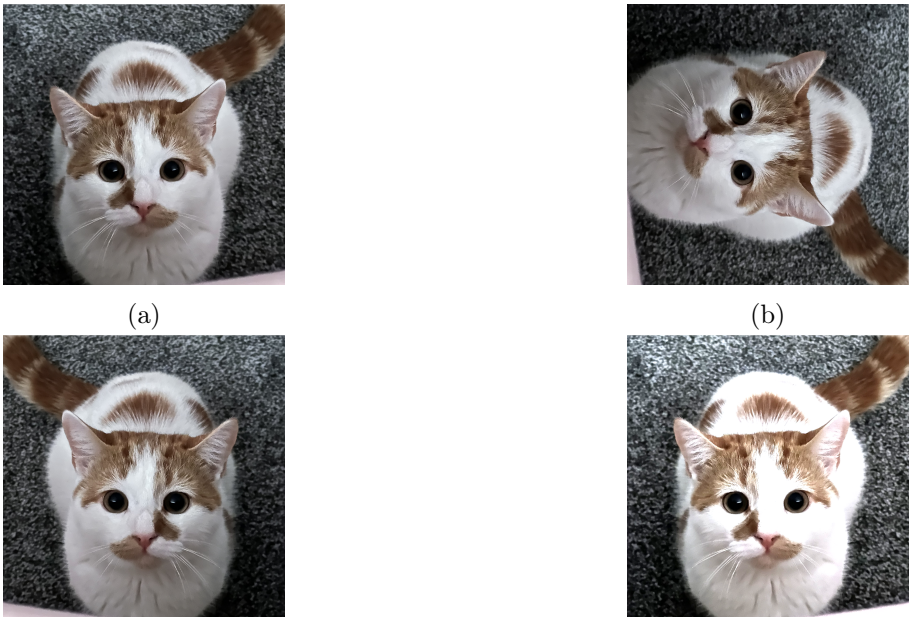
### 2.3.3 Data Augmentation

The foundation of AI is built upon data. Data serves as the critical input from which models learn and derive patterns that are necessary for making predictions or decisions without explicit programming. The quality and quantity of this data directly influence a model's performance; richer datasets can lead to more accurate and robust AI systems, as D. Douglas Miller discusses in "The medical AI insurgency: what physicians must know about data to practice with intelligent machines" [33]. However, acquiring a large amount of high-quality data can be challenging due to constraints such as privacy issues, resource limitations, and the time required for collection and labeling.

To mitigate the limitations associated with insufficient training data, data augmentation techniques are employed. Data augmentation involves artificially expanding the size and diversity of datasets by creating modified versions of data instances. This process helps in preventing overfitting, a scenario where a model learns the training data too well, including its noise and errors, and performs poorly on unseen data. By introducing variability through augmentation, models can generalize better to new, unseen data, which enhances their predictive performance and robustness.

Data augmentation techniques vary widely depending on the type of data and the specific application. In the context of image processing, common methods include transformations such as rotation, scaling, cropping, flipping, and altering lighting conditions [9]. For textual data, techniques like synonym replacement [25], back translation [47], and sentence shuffling [11] are used to create linguistically diverse expressions with the same underlying meaning. For signal possessing some of the common techniques are time stretching, sliding window, pitch shifting and time shifting to name a few.

For instance, by rotating an image of a cat by various angles, or changing the brightness and flipping the cat, a model training to recognize cats learns to recognize them regardless of their orientation or color in new images. This flexibility can be crucial for developing AI systems capable of operating in dynamic real-world environments. In Figure 2.4, Theo the cat has undergone several data augmentations. In doing so the image of Theo has now turned into three more data instances that can be used in the training of the AI.



(a) (b) (c) (d)

Figure 2.4: The various transformations applied to Theo the cat. (a) Original, (b) Rotated, (c) Mirrored, (d) Brightened.

### 2.3.4 Data Representation

In the discussions within the sections about the NASA Bearing Dataset 2.2.1 and the CWRU Bearing Data Center 2.2.2, it is highlighted that the datasets under consideration are represented in the time domain. The time domain generally refers to any variable that changes over time (or potentially space), characterizing a wide range of physical phenomena. Examples in everyday observations can be, temperature fluctuations, variations in sea levels, power consumption patterns, and voltage signals. Essentially, any sequence of data points

recorded over successive time intervals can be construed as time-domain data.

Time-domain data is inherently rich in information, potentially capturing significant insights about underlying phenomena. To unveil these hidden insights, particularly in the context of signal processing, the application of a Fast Fourier Transform (FFT) is common practice. The FFT is an algorithmically optimized implementation of the Discrete Fourier Transform (DFT), designed to efficiently transform time-domain signals into a frequency-domain representation. When FFT is applied to a time series signal $x$, the resulting output represents the signal's frequency components in terms of amplitude $\alpha$ and phase $\phi$, as shown in Figure 2.5.


Figure 2.5: Output from applying FFT to a signal X, consisting of n data points. The outputs are the amplitudes ($\alpha$) and phases ($\phi$).

The FFT is a transformation that follows the following equation:

$$\text{FFT}\{x(n)\} = X(f) = \int_{-\infty}^{\infty} x(n)e^{-j2\pi ft}\,dt. \tag{2.5}$$

Here, $X(f)$ represents the frequency domain representation of the time-domain signal $x(n)$, with $f$ denoting frequency. FFT thereby lets the input transition from time-domain observations to a more comprehensive frequency-domain analysis.

Figure 2.6 shows a visual representation of how DFT extracts the frequencies present in a time-domain signal to the more easily understandable frequency domain.

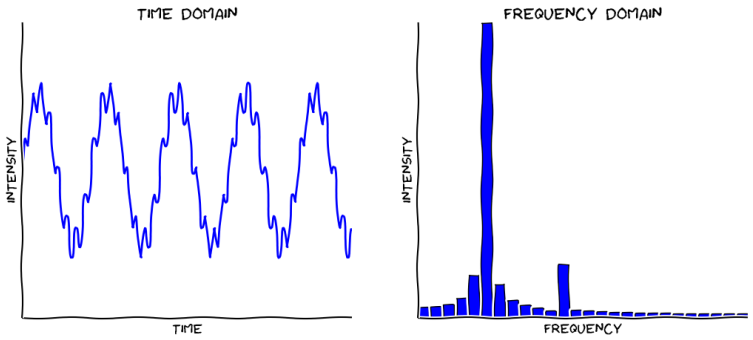



Figure 2.6: Input of a signal X(n) that goes into a DFT filter. The output is the different frequencies with amplitude $\alpha$ and phase $\phi$. Image adapted from Adafruit [3].

It is important to note that the output of FFT is a mirrored image of the frequency components. This occurs because the FFT of a time signal yields a symmetric spectrum, as is shown in an example on the SciPy documentation on Fourier transforms [16]. Therefore, only half of the FFT output is unique, while the other half is a mirrored duplicate, containing redundant information.

The FFT conversion yields a novel dataset: frequency-domain data. Unlike its time-domain counterpart, frequency-domain data is inherently three-dimensional, time, frequency, and

the magnitude of each frequency component. This multidimensional nature of frequency-domain data is often visualized through a spectrogram (Figure 2.8).

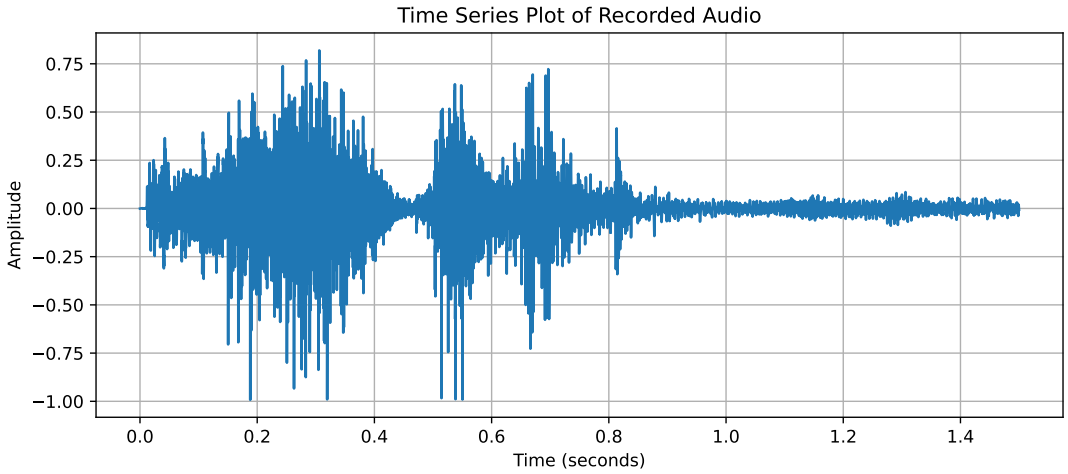



Figure 2.7: Time series representation of a sound signal recorded through a laptop microphone.

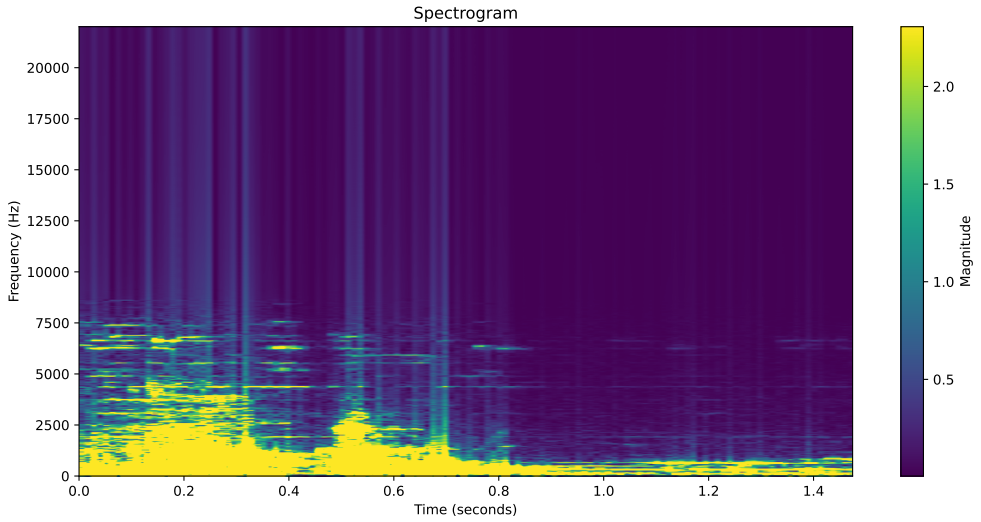



Figure 2.8: Spectrogram representation of the same sound signal as shown in Figure 2.7.

A spectrogram provides a comprehensive depiction of how the frequency components of a signal vary with time, contrasting the two-dimensional nature of time series plots, where time is plotted against signal strength, with the three-dimensional perspective of frequency analysis. In a spectrogram, time and frequency constitute the axes, while the strength, or amplitude, of each frequency component is represented through varying colors. This visualization not only facilitates a deeper understanding of the signal's characteristics but also enables the identification of patterns that are not discernible in the time domain.
Spectrograms are the form of data which the transformer model in our research is ultimately trained on. This approach leverages the transformer's capacity to interpret complex data by recognizing patterns within the spectral-temporal domain, providing a method to classify bearing faults from vibrational data. The TM, however, needs some more processing done to the data before it can be trained on it, as described in Section 2.3.5.

### 2.3.5 Boolean Data Representation

Boolean data is a data type that represents two possible values, often represented as true / false, positive / negative and 1 / 0. The TM and its component Tsetlin Automata (TA), later discussed in section 2.4.2 use Booleanized input data. Because of this, the frequency-domain data needs to be transformed into some sort of Boolean data.

There are various methods for converting data into Boolean form, each with specific considerations and strengths. One of the simplest yet highly effective methods is thresholding. This approach can be implemented in several ways, each suited to different scenarios. To illustrate, common thresholding techniques include global thresholding, adaptive mean thresholding, and adaptive Gaussian thresholding (AGT), as detailed in [35]. These methods provide flexibility in handling diverse data characteristics, thereby enhancing the Booleanization process.

#### Global Thresholding

Global thresholding is a straightforward yet powerful method for converting data into Boolean form. In this technique, the user specifies a threshold value $v$. Each data point in the dataset is then compared against this threshold. If a data point's value is less than the threshold, it is assigned a minimum value, typically 0. The opposite happens if the data point's value meets or exceeds the threshold, it is assigned a maximum value, often 1 or 255. This binary conversion process is depicted in Figure 2.9b, demonstrating the application of global thresholding on a image.

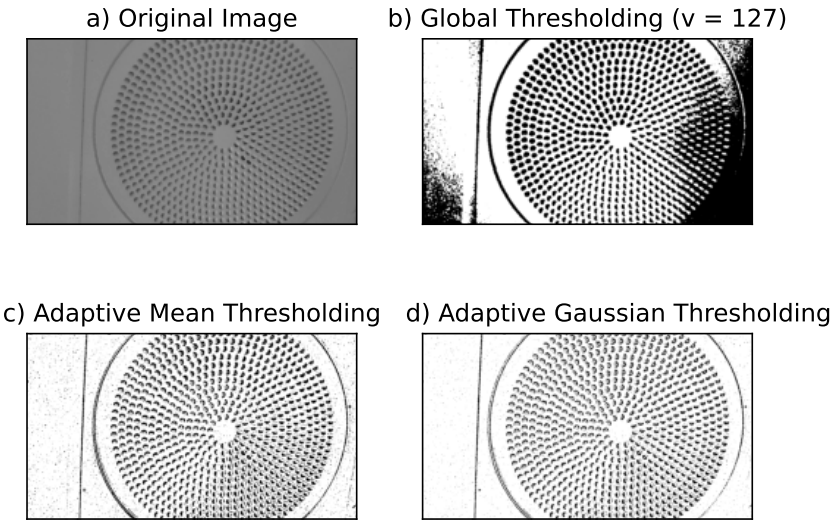



Figure 2.9: Image of an air vent. a) Grayscaled and salt and pepper noise added to the original image. b) Global thresholding where all values lower than a value $v$ becomes 0 and all other values becomes 255. c) Adaptive mean thresholding with block size 9 and constant 2. d) AGT with block size 9 and constant 2.

#### Adaptive Mean Thresholding

Adaptive mean thresholding [35] calculates the threshold for a pixel as the mean value of a specified neighborhood around that pixel, called "block", and then subtracting a constant

$C$. This method is effective for images with varying illumination conditions. An example of adaptive mean thresholding can be seen in Figure 2.9c, illustrating the localized thresholding effect on an image.

$$\text{Threshold}_{\text{mean}} = \text{Mean}_{\text{local}} - C. \tag{2.6}$$

**Adaptive Gaussian Thresholding**

In contrast, AGT [35] sets the threshold by computing a Gaussian-weighted sum of the block values around each pixel, also subtracting a constant $C$. This approach gives more weight to pixels near the center of the window and is particularly useful for preserving edge details in areas of significant color variation. Figure 2.9d demonstrates this method.

$$\text{Threshold}_{\text{gaussian}} = \text{Sum}_{\text{gaussian weighted}}(\text{neighborhood}) - C. \tag{2.7}$$

**Binary Embedding**

Binary embedding offers an efficient method for representing data in a Boolean format. This method involves converting the original data into a binary code. For instance, consider the number 54; when converted into binary, it becomes 110110. This approach is beneficial because it allows the encoding of a substantial amount of information within a relatively small number of bits. A single bit, which can be either 1 or 0, represents two distinct data values. Similarly, two bits can represent four distinct data values: 00, 01, 10, and 11. The capacity for data values increases exponentially with the number of bits used. This relationship is quantified in the following equation:

$$\text{Number of Data Values} = 2^n, \tag{2.8}$$

where $n$ is the number of bits. This formula demonstrates the potential for binary embedding to store a large volume of information efficiently, making it a valuable technique for data processing and storage in machine learning applications.

**Thermometer Embedding**

Thermometer embedding is a method used to convert scalar inputs into binary vectors while preserving their ordinal nature. This property makes it particularly useful for categorical data with inherent order or scale. For example, encoding a value of 3 on a scale of 0 to 5 with thermometer embedding would result in the binary vector:

$$3 \rightarrow [1, 1, 1, 0, 0]. \tag{2.9}$$

This vector indicates that the first three positions are filled to represent the value "3", reflecting its ordinal position relative to the maximum scale of 5. The number of bits required depends on the maximum value that needs to be encoded. For any maximum value $N$, $N-1$ bits are required, ensuring that each step up to $N$ can be uniquely represented.

$$\text{Number of Bits} = N - 1. \tag{2.10}$$

One key advantage of thermometer encoding over binary encoding is that it inherently preserves the scale information. This prevents the loss of ordinal relationships between values, which can occur with standard binary encoding. Thermometer embedding is advantageous in scenarios where the order of magnitude can be critical, providing a straightforward representation that maintains the inherent hierarchy of data values.

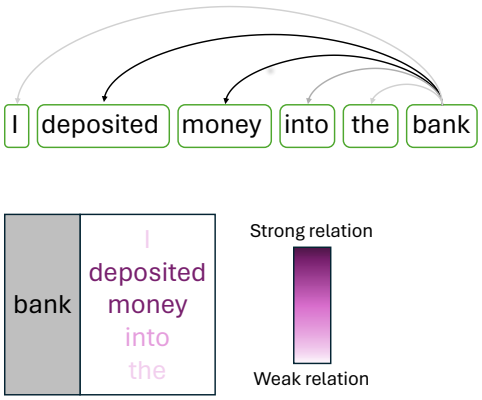



Figure 2.10: Self-attention visualized as the strength of relations between the word "bank" and different words in a sentence.

## 2.4 AI Algorithms

### 2.4.1 The Transformer

Transformers, introduced in 2017 [51], are a type of deep learning models that have revolutionized the field of natural language processing (NLP) [14] and are now making significant progress in other areas of machine learning, such as computer vision [12] and time-series analysis [15]. They are designed to process sequential data without the need for a recurrent architecture, giving them the ability to process data in parallel. This significantly improves the computational efficiency and performance over other models.

#### Self-Attention

The main idea of the transformer architecture is the self-attention mechanism. This mechanism allows the model to put emphasis on different parts of the input data differently. In the context of language models, this means that a transformer can focus more on certain words or phrases when predicting the next word in a sentence. For instance, when processing the word "bank" in the sentence "I deposited money into the bank", the self-attention mechanism allows the model to associate the word "bank" more with the word "deposited" and the word "money" than with other words like "I" or "into", which aren't as relevant to the word "bank." This also provides the context that "bank" refers to the financial institution and not the side of a river (a river bank). In the context of time-series analysis, this translates to the model being able to look at a specific part of a signal within the full context, relating it more to parts it should relate it to, and less to others.

#### Query, Key and Value Vectors

The self-attention mechanism uses three vectors to operate: query ($Q$), key ($K$) and value ($V$). These vectors derive from the input embeddings and are transformed through separate weight matrices learned during training. The query vector corresponds to the current datapoint being processed, while the key vectors correspond to all the datapoints in the input sequence. The value vectors contain the actual content of the datapoints. During self-attention, each query vector interacts with each key vector to create a score, which is then used to weight the corresponding value vectors. The scores indicate how much focus or "attention" the model should place on other parts of the input when processing a particular

datapoint. The transformation of input data into query, key and value vectors is represented mathemathically as follows:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V. \tag{2.11}$$

Here, $X$ represents the input embeddings, and $W^Q$, $W^K$, and $W^V$ are the weight matrices that are learned during the training process. Each matrix projects the input into a new space that contains the queries, keys and values, which are used in calculating the attention scores.

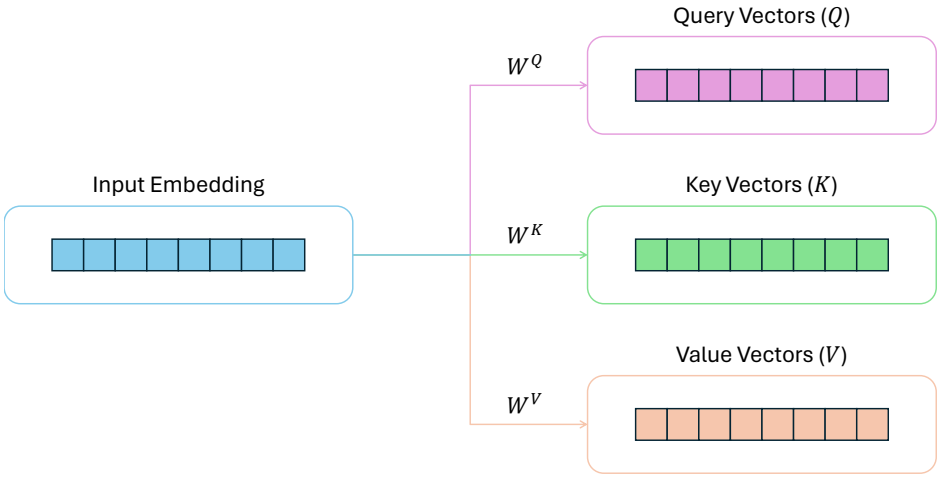



Figure 2.11: Transformation of input embedding into query, key, and value vectors.

**Attention Score and Weights**

The attention weight determines how much each part of the input contributes to each part of the output. It is calculated by taking the dot product of the query vector with all key vectors, followed by a softmax operation. This obtains the weights for the value vectors. The higher the weight for one part of the input, the more "attention" the model pays to that part of the input.

The attention scores that are used to obtain the weights, are computed using the dot product of the query and key vectors, adjusted by the square root of the dimension of the key vectors, to control the scale of the dot product:

$$\text{score}_{ij} = \frac{Q_i K_j^T}{\sqrt{d_k}}. \tag{2.12}$$

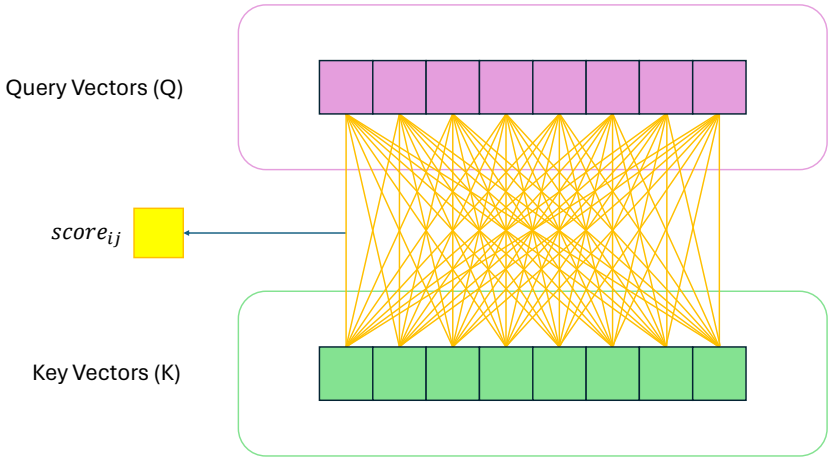



Figure 2.12: Dot product being performed between each query vector and each key vector to make the scores.

This score represents the alignment between query $i$ and key $j$, which influences how much each value vector will contribute to the final output.
The attention weights, which determine the final output, are derived from the attention scores using the softmax function:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.13}$$

where $d_k$ is the dimensionality of the key vectors, which scales the dot products to avoid extremely small gradients during training. Softmax ensures that the weights sum up to one, allowing the model to perform a weighted sum of the value vectors based on the attention scores. Figure 2.13 shows the attention scores resulting from each dot product of query vectors and key vectors, going through the softmax operation. These become the attention weights used to find the output vectors.

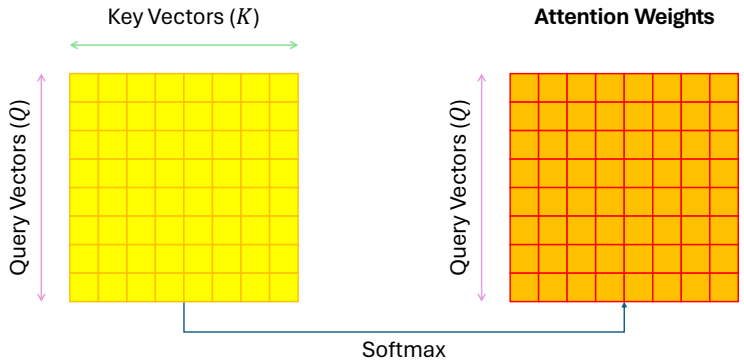



Figure 2.13: The softmax operation being applied to the scores matrix to make the matrix of weighted scores.

Once the attention weights are obtained through the softmax function, each value vector is then scaled by its corresponding weight. This ensures that the most relevant features are weighted more, and the less relevant are weighted less. The final step in the attention

mechanism is to sum these weighted value vectors to make the output vector for each position in the input sequence. This output vector is a representation rich in information, that takes into account both the content of the input and the significance of each part of the input as derived from the self-attention process. Figure 2.14 illustrates this transformation from the value vectors and attention weights into the final attention output vectors.

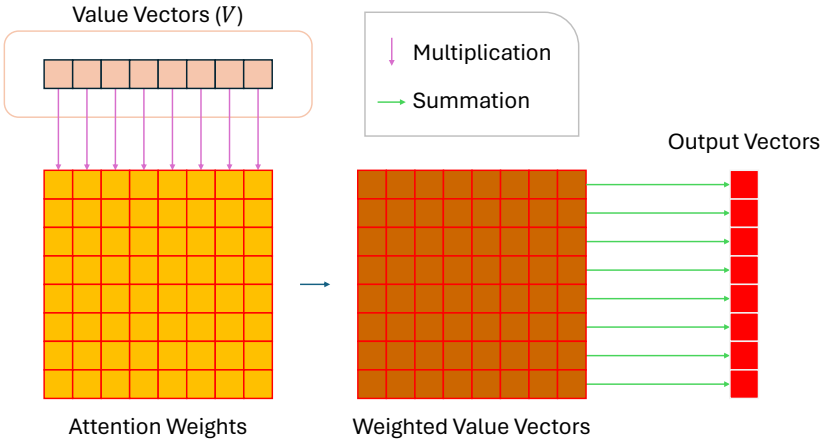



Figure 2.14: The summation of weighted value vectors to produce the final output vectors in the self-attention mechanism. Each row in the attention weights matrix is multiplied with the value vectors (V), resulting in the weighted value vectors, which are then summed row-wise to generate the output vector for each position in the sequence.

This attention mechanism is what allows the model to focus on relevant parts of the input sequence more than others, regardless of their position, when making predictions. This also enables the model to capture long-range dependencies.

### Positional Encoding

Since transformers process input data in parallel and do not inherently model the order of the sequence, positional encodings are added to the input embeddings to provide the model with information about the order of the sequence. There are various ways of making these encodings, but they typically involve sine and cosine functions of different frequencies. The formulas for positional encodings using sine and cosine functions can be:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \tag{2.14}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right), \tag{2.15}$$

where $pos$ is the position of the word in the sequence, and $2i$ and $2i+1$ correspond to each dimension of the positional encoding vector. These encodings help the model to understand the order of the sequence, which is crucial for processing time-dependent data like a time-series.

### Performance on Spectrograms and Non-NLP Tasks

Transformers were originally designed for NLP tasks, but their ability to process sequential data in parallel and to capture long-range dependencies in general, has led to them being applied in other domains. One of these domains is for analyzing spectrograms in audio processing. The Audio Spectrogram Transformer (AST) exemplifies this capability by effectively capturing the temporal dynamics in spectrograms. The self-attention mechanism

of the AST model lets it focus on specific parts of the time dimension of the spectrogram, making it perform well on audio classification tasks[17].
This example shows that researchers have adapted the transformer architecture for non-NLP tasks, showing promising results.

**Comparison with Other Neural Networks**

EfficientNets and CNNs are highly effective at processing spatial information due to their convolutional nature, which allows them to excel in tasks like image recognition where spatial hierarchies are key. However, these models typically require additional mechanisms, such as recurrence or attention layers, to handle sequence prediction tasks effectively.

In contrast, BiD-LSTMs excel in sequence processing due to their ability to capture information from both past and future contexts within a sequence. They are therefore especially useful for tasks where context sensitivity is essential. However, while BiD-LSTMs are powerful for their understanding of context, they may not process data as efficiently as transformer models, which handle sequences in parallel. This architectural difference allows transformers to train more effectively on large datasets or complex sequence tasks.

Transformers address some of the limitations of other models by using the self-attention mechanism that allows the model to weigh the importance of different parts of the input data no matter their sequential order. This lets transformers process data in parallel, significantly speeding up training times. Moreover, unlike CNNs and EfficientNets, transformers can inherently capture long-range dependencies across entire sequences, making them especially well suited for tasks where understanding the global context is important, such as in language modeling or complex time-series analysis.

The flexibility and efficiency of transformers, therefore, allow them to excel in a wide range of tasks, surpassing traditional models in environments where both spatial and temporal dynamics are significant, such as audio processing and other signal analysis tasks such as spectrogram analysis.

**Transformer Architecture**

The transformer model typically consists of an encoder and a decoder, with multiple layers in each. The encoder maps an input sequence to a sequence of continuous representations, which the decoder then uses to generate an output sequence. Each encoder layer has two sub-layers: the self-attention layer and a position-wise fully connected feed-forward network.

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2. \tag{2.16}$$

This network is applied to each position separately and identically, allowing the model to integrate information from the self-attention layer further.
Residual connections around each of the two sub-layers, followed by normalization, help to avoid the vanishing gradient problem and encourage smooth training.
The decoder also includes a similar sub-layer structure, but adds a third sub-layer that performs multi-head attention over the output of the encoder stack. This structure allows each position in the decoder to attend to all positions in the input sequence, integrating information from across the whole sequence. For classification, however, as generation is unnecessary, so is the decoder. When the goal is classification, for instance to identify faulty machinery from vibration data, the decoder is therefore often omitted, for example in the Bidirectional Encoder Representations from Transformers (BERT) [14]. The encoder captures the necessary contextual information within the input, which is then passed directly into a classifier. This simplification of the architecture is possible because the task does not

require the generation of new sequence data but rather the interpretation and categorization of the input data.

$$\text{Classifier Output} = \text{Softmax}(\text{FFN}(\text{Encoder Output})). \tag{2.17}$$

Here, Encoder Output refers to the contextualized representation of the input data produced by the transformer encoder, and FNN represents a feed-forward network that processes this output and feeds it into a softmax function for classification. This architecture is not only more efficient but also reduces the complexity of the model.

#### Multi-Head Attention

Multi-head attention is one of the defining characteristics of the transformer model. Instead of performing one single attention operation, the model projects the queries, keys and values h different times with different learned linear projections, where h is the number of multi-head attention heads. These projections are then fed into attention mechanisms in parallel, allowing the model to capture information from different representation subspaces at different positions.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O, \tag{2.18}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \tag{2.19}$$

Each head captures different features from the input, and the outputs are combined to form the final attention output, which is passed through another learned linear transformation. This approach provides a more nuanced understanding of the input.

#### Transformers for Spectrograms

Applying the transformer architecture to spectrograms involves treating the time-frequency representation as a sequence of vectors, each vector representing a slice of time with its corresponding frequency content. The transformer can attend to different slices to identify temporal patterns, such as the evolution of harmonic structures or the presence of transient noises that could indicate mechanical failures or speech phonemes. This application of transformers to audio signal processing has been exemplified by the Audio Spectrogram Transformer (AST) model, which demonstrates the ability of transformers to analyze spectrogram data effectively for detecting and analyzing complex audio events [17].

#### Previous Performances

In practice, transformers have set new standards of performance in a wide variety of fields. In image recognition tasks, Vision Transformers (ViTs) treat parts of images as sequences, achieving results competitive with state-of-the-art CNNs [12]. In signal processing, transformers have recently been used to classify heartbeats from ECG data with high accuracy, as demonstrated by the ECGformer model, which effectively classifies various arrhythmias based on time-signal data [5]. These successes show the model's adaptability and potential for innovation in machine learning.

#### Closing Remarks on Transformers

The versatility of the transformer architecture has made it a fundamental building block for contemporary AI systems. Its success in handling spectrogram data promises further advancements in fields like audio signal processing and condition monitoring, where understanding complex, time-varying signals is crucial. As research continues to evolve, we can expect transformers to play a bigger role in advancing the field of AI.

### 2.4.2 Tsetlin Machine

The TM is a novel approach to machine learning that operates on the principles of propositional logic [18] and learning automatas [34]. Unlike conventional machine learning models, which typically rely on statistical methods or deep learning techniques, the TM harnesses the power of propositional logic to solve classification and regression problems in a transparent and interpretable manner. Developed by Ole-Christoffer Granmo in 2018 [18], the TM is based on the TA, a type of learning automaton named after the Soviet mathematician M. L. Tsetlin, who in the early 1960s introduced it. These TAs (Figure 2.15) are simple learning machines designed to make optimal decisions through trial and error, guided by a reinforcement mechanism called feedback.

#### Explainability in Artificial Intelligence

AI explainability refers to the methods and processes that allow humans to understand the decisions or outputs generated by AI systems. This aspect of AI development is sought after because it helps clarify how algorithms arrive at their conclusions, especially in complex models like deep learning, where decisions are often made in what is referred to as a "black box." Explainable AI aims to make the operations of AI systems transparent, providing insights into their strengths and weaknesses, and clarifying their decision-making processes. This is particularly important in high-stakes areas such as healthcare, finance, and legal systems [36] [24], where understanding the basis of an AI's decision can impact human lives and legal responsibilities. Explainability also enables debugging and improvement of AI models, enhances user trust, and ensures compliance with increasing regulatory requirements that demand transparency in AI operations.

In recent years, many nations have been establishing rules and guidelines to govern the development of AI. A significant portion of these regulatory efforts can be addressed by developing AI models that are explainable and interpretable. The European Parliament introduced the AI Act in June 2023 [37], marking the European Union (EU)'s first comprehensive regulation of AI technologies. The AI Act emphasizes that:

> "AI systems should be overseen by people, rather than by automation, to prevent harmful outcomes."

This directive underscores the need for AI systems that humans can understand and manage effectively, which encourages the development of explainable and interpretable AI models. Such models enhance transparency and accountability, allowing human overseers to comprehend and justify the AI's decisions. This is crucial not only for building trust in AI systems but also for ensuring they align with ethical standards and legal requirements.

The executive order issued by United States (US) President Joe Biden on October 30, 2023 [24], aims to position the US as a leader in managing the potential risks while seizing the opportunities presented by AI. This directive sets new standards for AI safety and security, enhances privacy protections, advances civil rights and equity, and promotes innovation and competition. Key measures include requiring developers of significant AI systems to disclose safety test results to the US government, creating rigorous testing standards through various agencies, and addressing cybersecurity risks linked to AI [24].

#### Explainable Tsetlin Machine

The TM is particularly celebrated for its inherent interpretability [18] [41] [1], a feature that stands out in the landscape of machine learning models. This attribute stems from its use of propositional logic to make decisions, which facilitates a clear understanding of how outputs are derived. A TM operates using a collection of TAs. The decisions made by these

automatas are combined to form clauses that collectively decide the output of the machine.

This structured approach allows for each output decision of the TM to be traced back through the active clauses and their corresponding TAs. By examining which clauses contributed to a decision and the state of the TAs within those clauses, one can directly map out the reasoning process of the TM. This traceability is not just a byproduct of its architecture but a core feature, making the TM exceptionally transparent compared to many of its counterparts in AI, where decision paths are often obscured within complex network layers. This makes the use of propositional logic not only the underlying operational mechanics of the TM but also provides it with a level of clarity that supports applications where understanding AI decision-making is crucial.

**Propositional Logic**

Propositional logic is a branch of mathematical logic that deals with propositions and their interrelations through logical connectives [21]. In its simplest form, propositional logic uses propositions that are statements that can either be true or false. Some of the fundamental connections used in propositional logic can be: and, or, not, implies and equivalent.

- AND (Conjunction): This connective results in true if both propositions it joins are true. AND is denoted by $\wedge$.
- OR (Disjunction): Yields true if at least one of the propositions it connects is true. OR is denoted by $\vee$.
- NOT (Negation): Inverts the truth value of the proposition it precedes. NOT is denoted by $\neg$.
- IMPLIES (Implication): Is true if either the first proposition is false or the second proposition is true. IMPLIES is denoted by $\rightarrow$.
- EQUIVALENT (Biconditional): Results in true if both propositions are either true or false simultaneously. EQUIVALENT is denoted by $\leftrightarrow$.

The power of propositional logic in computational contexts lies in its ability to simplify decision-making processes and problem-solving tasks [44]. Logical expressions or formulas composed of these connectives and propositions can be systematically evaluated to determine their truth values under various scenarios. The simplicity of propositional logic allows for the design of very efficient algorithms to tackle practical problems. This simplicity also aids in making the algorithms more interpretable, an advantage in scenarios where understanding the decision-making process of AI systems is critical.

| **Operation** | **Expression** | **Input (p, q)** | **Output** |
|---|---|---|---|
| AND (Conjunction) | $p \wedge q$ | (T, T) | T |
| AND (Conjunction) | $p \wedge q$ | (T, F) | F |
| OR (Disjunction) | $p \vee q$ | (F, T) | T |
| OR (Disjunction) | $p \vee q$ | (F, F) | F |
| NOT (Negation) | $\neg p$ | (F) | T |
| NOT (Negation) | $\neg p$ | (T) | F |
| IMPLIES (Implication) | $p \rightarrow q$ | (F, T) | T |
| IMPLIES (Implication) | $p \rightarrow q$ | (T, F) | F |
| EQUIVALENT (Biconditional) | $p \leftrightarrow q$ | (T, T) | T |
| EQUIVALENT (Biconditional) | $p \leftrightarrow q$ | (T, F) | F |

Table 2.1: Examples of propositional logic operations.

### Tsetlin Automata

Within the intricate framework of the TM, the TA plays a central role, acting as the engine that drives the TM's decision-making process and the TM's learning process through a meticulously designed system of reinforcement. This system operates on the principles of penalizing or rewarding actions based on their outcomes. The TA is characterized by its straightforward, linear architecture, comprising $2N$ states. Its operational mechanism involves transitioning between these states in response to a series of rewards and penalties. The set of states is divided into two halves: the first half spanning from state 1 to $N$, representing the first action zone, and the second half from state $N+1$ to $2N$, representing the second action zone. Within the first action zone, receiving a penalty nudges the TA towards state $N+1$, whereas a reward moves it towards state 1. Conversely, in the second action zone, this dynamic is inverted: penalties push the TA towards state $N$, and rewards towards state $2N$ as shown in Figure 2.15. This design ensures a balanced and effective learning process through the systematic application of reinforcement principles.

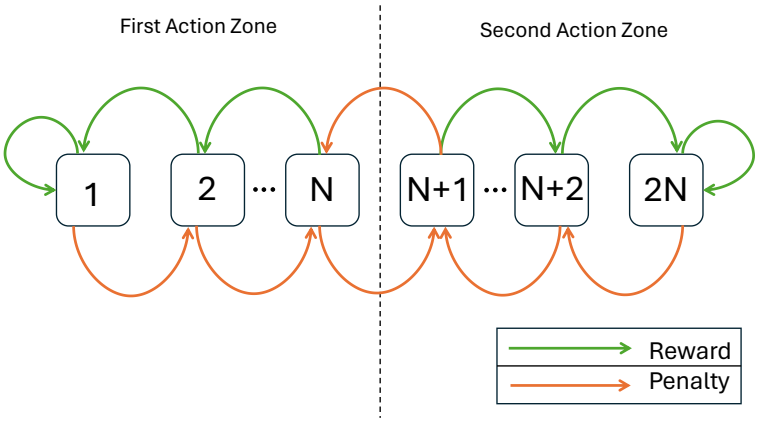



Figure 2.15: The two action zones and its rewards and penalties for different states.

### Clauses and Literals

At the heart of the TM lies its foundational reliance on propositional logic [18]. This reliance forms the bedrock upon which TM operates, utilizing a set of basic building blocks: clauses, TAs, and literals. Literals are the data the TM uses. One data point creates two literals, positive literal or a negated literal. If we have the data $[x_1, x_2, x_3]$, this will correspond to $[x_1, \neg x_1, x_2, \neg x_2, x_3, \neg x_3]$. If we create an example where we have $x_1$ and $x_2$ but no $x_3$ it would look like this: [True, False, True, False, False, True] becoming $[1, 0, 1, 0, 0, 1]$. This approach ensures that every aspect of the input data is mirrored, allowing the TM to effectively capture and process both the presence and absence of features, which is vital for robust decision-making. The introduction of Boolean data as input to the clauses, underscores the TM's reliance on propositional logic. This logical framework ensures that for every positive literal, there is a corresponding negated literal. Consequently, if there are $o$ positive literals, the total count of literals would be $2o$.

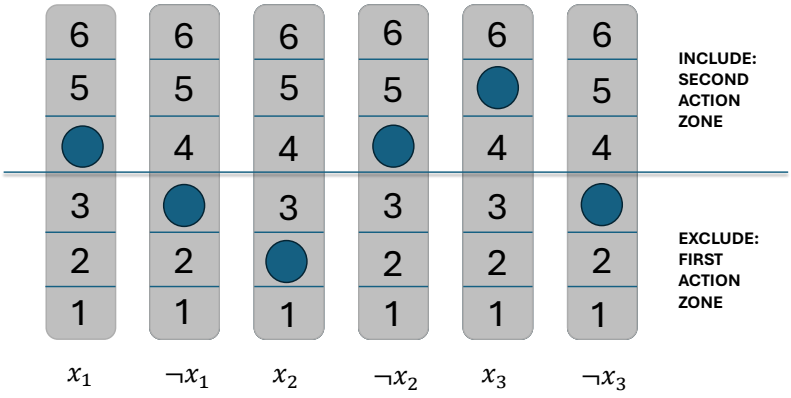



Figure 2.16: A clause where $x_1 \wedge \neg x_2 \wedge x_3$ are in the first action zone, and $\neg x_1 \wedge x_2 \wedge \neg x_3$ are in the second action zone.

One clause consists of $2o$ TAs, one TA for each positive and negative literal. This is demonstrated in Figure 2.16, which shows one clause where $o$ is equal to 3. The blue circles represent the state of each of the $2o$ TAs. When a TA is in the first action zone they are excluded from the clause vote, and when they are in the second action zone they are included into the clause vote. Clause vote is the mechanism that clauses use to either vote for their corresponding class or not.

In a Classical TM, clauses play a crucial role in distinguishing between two classes, labeled as class 0 and class 1. Each clause is associated with a specific polarity, aligning it in favor of either class 0 or class 1. This polarity determines how the clause votes based on incoming inputs by selectively including or excluding certain TAs based on their state and action zone. The cumulative effect of these inclusions or exclusions influences the clause's vote. Ultimately, the majority vote across all clauses, considering their polarities, decides the class selection. This mechanism enables the TM to classify input data by considering the collective behavior of clauses and their polarities in a decision-making process.

When the TM is made, a user inputs a number $n$. That number represents the amount of clauses. For a TM the number of clauses is then divided by the amount of classes to predict. For the Classical TM, the amount of clauses are divided by two, one for each polarity. This is then shown as $C_j^1$ and $C_j^0$, where the upper index 1 represents the positive polarity and upper index 0 represents the negative polarity. The lower index denoted $j$ is a list spanning from 1 to $n/2$. This represents each clause in the given polarity. The clause vote is calculated by summing all the TAs inside the clause. This can be seen in the Equation below:

$$v_p = \sum_{j=1}^{\frac{n}{2}} C_j^p(X), \tag{2.20}$$

where $p$ represents the two different polarities. With an equation for retrieving votes from clauses, it can be used to make the equation for the majority vote. To derive the output of the classical TM, which is binary (either 1 or 0), a threshold function $u(v)$ is applied:

$$u(v) = 1 \textbf{ if } v \geq 0 \textbf{ else } 0. \tag{2.21}$$

This thresholding mechanism facilitates the determination of the majority vote from the clauses, encapsulated by the equation:

$$\hat{y} = u\left(\sum_{j=1}^{\frac{n}{2}} C_j^1(X) - \sum_{j=1}^{\frac{n}{2}} C_j^0(X)\right). \tag{2.22}$$

The output of Equation (2.22) is the output of the Classical TM. Because the clauses can include and exclude different literals into its majority vote it can use the Boolean data and clauses to create its own propositional logic solutions.

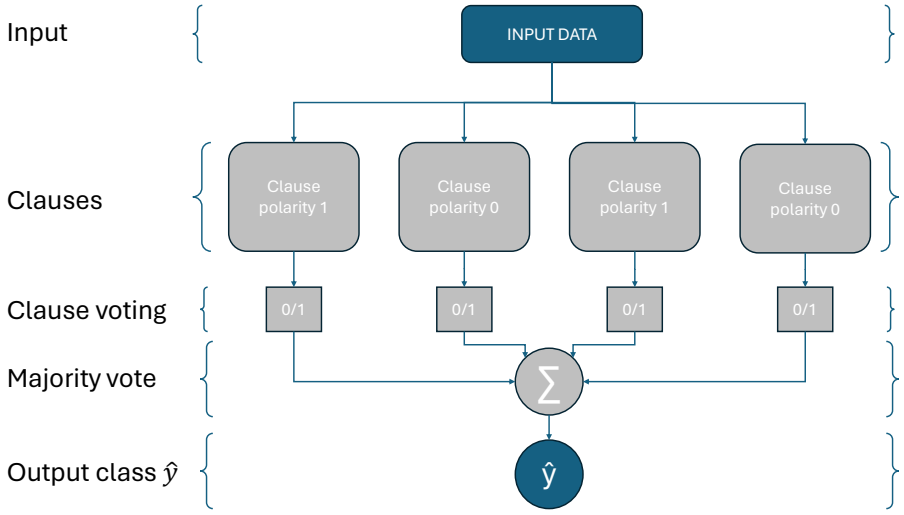



Figure 2.17: Pipeline of a TM with 4 clauses. The majority vote predicts the output $\hat{y}$.

With the foundational understanding of TA and clauses, the operation of the TM can be articulated from the basics shown in the Figure 2.17. Initially, input data are fed into the system, where each clause evaluates them. Clauses consist of literals and their negations, allowing for a diverse interpretation of inputs. The inclusion or exclusion of specific TA within a clause is determined by their state, influencing the clause's contribution to the decision-making process. The collective output from all clauses undergoes a majority vote to produce the final classification. Then the TM undergoes a feedback loop step, which guides the learning and adaptation of the TM, ensuring its decisions evolve over time to reflect the complexities of the input data accurately.

**Feedback**

The TM employs a feedback mechanism during training to refine the behavior of its clauses and the underlying TAs, addressing both false negatives and false positives through two distinct types of feedback: type I and type II.

Type I feedback aims to mitigate false negatives and reinforce true positive outcomes. A false negative occurs when the TM incorrectly labels an actual positive case as negative, missing a true detection. Conversely, a true positive is an accurate identification of a positive case by the TM. Type I feedback operates on two fronts that can be divided into type Ia and type Ib [19]. For true positives, type Ia feedback is used. It aims to further increase the clause's accuracy by promoting the inclusion of relevant literals, making the clause denser and more specific to positive patterns. For false negatives, type Ib feedback is used. It encourages clauses to become more exclusive by encouraging literals to move towards the exclusion zone, thereby decreasing the clause's sensitivity to false negatives and becoming more sparse.

Type II feedback, on the other hand, is designed to counteract false positives, where the TM labels a negative case as positive. This feedback type is activated only when the TM's output

is falsely positive. It specifically targets literals that contribute to this incorrect output by encouraging their exclusion from the decision-making process, thus making the clause less likely to erroneously activate for similar cases in the future.

This feedback loop allows the TM to dynamically adjust its clauses, enhancing its ability to accurately classify input data by learning from its errors and reinforcing its successes, thereby improving its overall decision-making capability over time.

The hyper-parameter $T$, initialized alongside the TM, serves a pivotal role. It is designed to facilitate the fair distribution of clauses across the various patterns present in the data. This approach ensures that approximately $T$ clauses of the correct polarity evaluate to 1 for any given input $X$. Moreover, the introduction of the target summation $T$ encourages a dynamic interaction among clauses, allowing for the adjustment and rectification of special cases.

For scaling the learning rate of the TM in a correct way, Granmo found a fantastic way to do that [18]. He constructed four equations that have its stepping stones in Equation (2.20) and the majority vote from Equation (2.22). The four equations are divided onto each of the two different types of feedback, where they are counterparts to each other. First a clause is selected at random. The result of each equation is the chance for a corresponding feedback type to act upon the clause. If the class $y$ equals the polarity $p$, the probability for retrieving type I feedback is:

$$\frac{T - clip\left(\sum_{j=i}^{n/2} C_j^1(X) - \sum_{j=i}^{n/2} C_j^0(X), -T, T\right)}{2T}, \tag{2.23}$$

where $X$ is the input. This is true only when $y = 1$. When $y = 0$, the equation becomes:

$$\frac{T + clip\left(\sum_{j=i}^{n/2} C_j^1(X) - \sum_{j=i}^{n/2} C_j^0(X), -T, T\right)}{2T}. \tag{2.24}$$

When a clause that has a polarity $p$ that is not equal to the class $y$, the probability for the selected clause to retrieve type II feedback when $y = 1$ is:

$$\frac{T + clip\left(\sum_{j=i}^{n/2} C_j^1(X) - \sum_{j=i}^{n/2} C_j^0(X), -T, T\right)}{2T}. \tag{2.25}$$

When $y = 0$, it becomes:

$$\frac{T - clip\left(\sum_{j=i}^{n/2} C_j^1(X) - \sum_{j=i}^{n/2} C_j^0(X), -T, T\right)}{2T}. \tag{2.26}$$

Also make note that Equation (2.23) and Equation (2.26) are equal. The same holds true for Equation (2.24) and Equation (2.25). These equations are visualized in Figure 2.18.

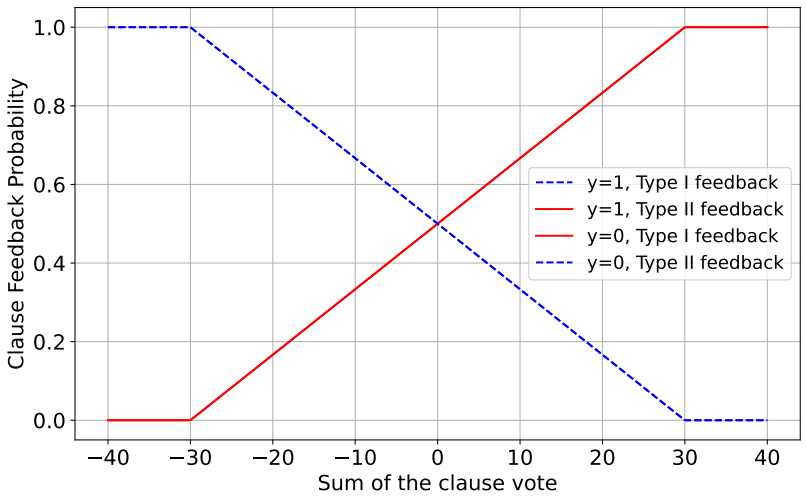



Figure 2.18: Clause probability where $T = 30$.

The clip function is defined as follows:

$$\text{clip}(x, \text{min}, \text{max}) = \begin{cases} \text{min} & \text{if } x \leq \text{min}, \\ x & \text{if min} < x < \text{max}, \\ \text{max} & \text{if } x \geq \text{max}. \end{cases}$$

Once a clause is selected for feedback within the TM, each TA within the clause is subject to a probability of being selected for learning. This probability varies depending on the type of feedback being applied, as illustrated in Table 2.2 for type I feedback. An analysis of type I feedback probabilities reveals a strong bias toward inclusion actions over exclusion actions [18]. This preference is evidenced by the probabilities favoring rewards or inactions over penalties, where $\frac{s-1}{s}$ is greater than $\frac{1}{s}$ when $s > 2$ shown in Figure 2.19, indicating a higher likelihood of rewarding or maintaining the status quo rather than penalizing.

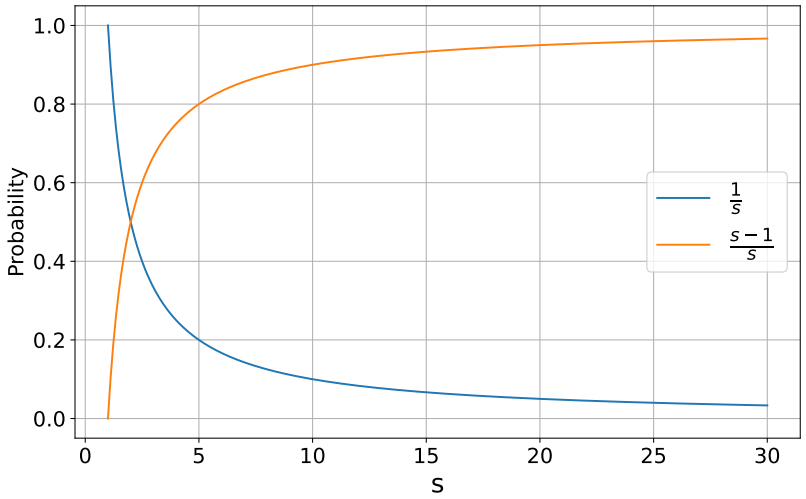



Figure 2.19: Probability difference between two functions: $\frac{s-1}{s}$, indicated by an orange line, and $\frac{1}{s}$, indicated by a blue line.

In contrast, as shown in Table 2.3 for type II feedback, this feedback type uniquely penalizes the exclusion of literals while opting for inaction in other scenarios. This delineation

in feedback strategies highlights the mechanism's adaptive approach to optimizing clause configurations based on the behavior of each individual TA within the system.

| **Truth Value of Clause $C_j^\omega$** | | **1** | | **0** | |
|---|---|---|---|---|---|
| **Truth Value of Literal $l_k$** | | **1** | **0** | **1** | **0** |
| Include Literal ($\alpha_2 \to l_k \in L_j^\omega$) | $P$(Reward) | $\frac{s-1}{s}$ | NA | 0 | 0 |
| | $P$(Inaction) | $\frac{1}{s}$ | NA | $\frac{s-1}{s}$ | $\frac{s-1}{s}$ |
| | $P$(Penalty) | 0 | NA | $\frac{1}{s}$ | $\frac{1}{s}$ |
| Exclude Literal ($\alpha_1 \to l_k \notin L_j^\omega$) | $P$(Reward) | 0 | $\frac{1}{s}$ | $\frac{s-1}{s}$ | $\frac{s-1}{s}$ |
| | $P$(Inaction) | $\frac{s-1}{s}$ | $\frac{s-1}{s}$ | $\frac{1}{s}$ | $\frac{1}{s}$ |
| | $P$(Penalty) | $\frac{1}{s}$ | 0 | 0 | 0 |

Table 2.2: Type I feedback. Table showing the probability of a single TA updating its state. $\alpha_1$ is action zone one, $\alpha_2$ is action zone two. $l_k$ indicates the literal to be evaluated for feedback in the clause $C_j^\omega$. $L_j^\omega$ is the set of included literals.

| **Truth Value of Clause $C_j^\omega$** | | **1** | | **0** | |
|---|---|---|---|---|---|
| **Truth Value of Literal $l_k$** | | **1** | **0** | **1** | **0** |
| Include Literal ($\alpha_2 \to l_k \in L_j^\omega$) | $P$(Reward) | 0 | NA | 0 | 0 |
| | $P$(Inaction) | 1 | NA | 1 | 1 |
| | $P$(Penalty) | 0 | NA | 0 | 0 |
| Exclude Literal ($\alpha_1 \to l_k \notin L_j^\omega$) | $P$(Reward) | 0 | 0 | 0 | 0 |
| | $P$(Inaction) | 1 | 0 | 1 | 1 |
| | $P$(Penalty) | 0 | 1 | 0 | 0 |

Table 2.3: Type II feedback. Table showing the probability of a single TA updating its state. $\alpha_1$ is action zone one (exclude), $\alpha_2$ is action zone two (include). $l_k$ indicates the literal to be evaluated for feedback in the clause $C_j^\omega$. $L_j^\omega$ is the set of included literals.

The hyper-parameter $s$ is also initialized alongside the TM. This hyper-parameter can control just how much the TM favors the include over the exclude action. $s$ decides just how "fine-grained" the patterns in the clauses are going to be [18]. Choosing a greater $s$ will increase probability for any TA to move its state from the first action zone to the second action zone.

#### Hyper-Parameter Tuning for $T$ and $s$

Hyper-parameters play a critical role in machine learning, influencing the speed of model training and significantly impacting performance [48]. Various tools exist for identifying optimal hyper-parameters, such as the widely used Adam optimizer [28], which has gained significant recognition in the field.

For the TM, hyper-parameter optimization is important due to its simpler hyper-parameter space compared to more complex models like deep neural networks. The primary hyper-parameters in TM are the number of clauses $C$, the target sum $T$ of the clauses, and the

learning sensitivity $s$ for each TA [18].

While the optimal setting for $C$ is generally to increase it for improved accuracy, the parameters $T$ and $s$ are more nuanced and require careful tuning. Recent studies, including Tarasyuk et al. [48], have focused on these parameters, demonstrating that their optimal values are crucial for achieving high performance on tasks like digit recognition using the MNIST dataset [29].

In their study they found that choosing a $T$ and $s$ depending on the $C$ was generating good results. They created Equation (2.27) for optimizing the $T$ and also the Equation (2.28) for optimizing the $s$.

$$T_{\text{optimal}}(C) = \left(\frac{C}{0.8377}\right)^{\frac{1}{2.2099}} \approx \sqrt{\frac{C}{2}} + 2. \tag{2.27}$$

$$s_{\text{optimal}}(C) = 2.534 \cdot \ln\left(\frac{C}{3.7579}\right). \tag{2.28}$$

The interplay between $T$ and $s$ significantly affects the TM's performance, where $T$ manages the clause voting threshold and $s$ adjusts the sensitivity of state transitions within TAs. Finding the balance between these parameters is key to enhancing TM's efficiency and effectiveness.

This optimization is not only a matter of adjusting the TM's internal mechanics but also involves aligning the learning process with the computational architecture, making it a pivotal aspect of deploying TMs effectively [48].

#### Drop Clause

In 2012, G. E. Hinton et al. [23] introduced a transformative concept known as dropout, which has since become a cornerstone in various machine learning algorithms. This technique involves temporarily removing portions of a machine learning model during training, thereby preventing overfitting and enhancing the model's generalization capability. N. Srivastava et al. [46] further elucidated this concept by demonstrating how neural networks incorporate dropout, significantly contributing to a model's robustness and overall accuracy improvement.

Building on this foundation, Jivitesh et al. [41] explored the application of dropout in the context of TM, proposing the Drop Clause (DC) technique. Unlike traditional dropout, which randomly omits neurons or connections, DC introduces stochasticity by selectively training subsets of clauses with a probability $(1 - p)$. This method not only amplifies the learning speed but also enhances the model's performance by making the clauses more sensitive and adaptive to the nuances of the data, thereby increasing their resilience to input disturbance.

The introduction of stochasticity in DC TM, as opposed to the stochastic gradient descent (SGD) approach of sampling mini-batches from the dataset, represents a novel strategy for enhancing learning dynamics. In DC TM, stochasticity emerges through the probabilistic selection of clauses for training, offering a more granular level of model adjustment. This unique approach to incorporating stochastic elements diverges from traditional dropout, positioning DC as a pivotal innovation that strengthens the model against overfitting while simultaneously improving its learning efficiency and data representation capabilities.

**Different Tsetlin Machines**

- **Multi-class Tsetlin Machine**
  Until now, the discussion on TM has mainly focused on their application in Boolean classification tasks, where the output is strictly limited to 1 or 0. However, numerous classification challenges require distinguishing among more than two classes. For instance, the MNIST [29] dataset encompasses ten distinct digits, each necessitating accurate classification. In such scenarios, a TM can be adapted to address multi-class classification by encoding the target variable, $y$, as a series of bits. Nevertheless, Granmo proposed an enhanced model specifically designed for this purpose, termed the Multi-class Tsetlin Machine (MTM) [18].

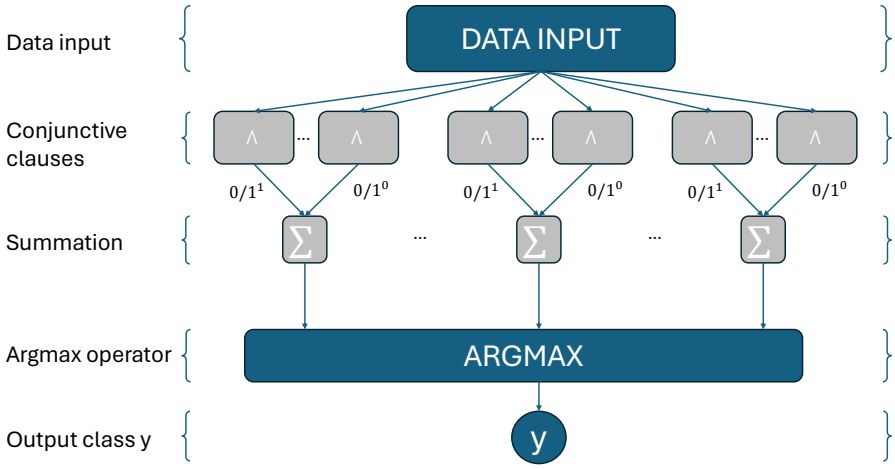



Figure 2.20: Architecture of the MTM.

As illustrated in Figure 2.20, the MTM's design deviates from the Classic TM by substituting the threshold functions associated with each output for a singular *argmax* operator. The *argmax* function operates by locating the index of the highest value within an array or sequence, thereby determining the most probable class by identifying the position of the maximum element. This modification allows the MTM to directly select the most appropriate class.

$$y = arg \max_{i=i,\ldots,m} \left( \sum_{j=1}^{\frac{n}{2}} C_j^{1,i}(X) - \sum_{j=1}^{\frac{n}{2}} C_j^{0,i}(X) \right). \tag{2.29}$$

Incorporating the *argmax* function into the MTM's operational framework fundamentally alters its decision-making process. As specified in Equation (2.29), the prediction $y$ previously defined by Equation (2.22) as $\hat{y}$, is recalculated using the *argmax* operator. Consequently, the index $i$ representing the class with the largest cumulative sum is selected as the output, signifying the MTM's class prediction.

- **Convolution Tsetlin Machine**
  Granmo also introduced the Convolutional Tsetlin Machine (CTM) [18] in his paper. Later, in 2019, Ole-Christoffer Granmo et al. [19] wrote the paper "The Convolutional Tsetlin Machine." To briefly demonstrate how the CTM works, if an input image has a size of $X \times Y \times Z$ where $Z$ represents the amount of binary layers, the CTM deploys a kernel over the input. The kernel size is set to $W \times W$. This kernel starts in the top left of the input, and moves with $d$ steps systematically to the bottom right, creating

batches $B$. $B = B_X \times B_Y$ where $B_X = \lceil \frac{X-W}{d} \rceil + 1$ and $B_Y = \lceil \frac{Y-W}{d} \rceil + 1$. Each clause inside the CTM has its own location-awareness [19] from a binary encoded location within the clause. This can prove to be useful when one clause might make its decision based on where its position is. Each clause outputs $B$ values per input, because the kernel moves to $B$ unique positions of the input [2].

- **Weighted Tsetlin Machine**
  The Weighted Tsetlin Machine (WTM) was introduced by Adrian Phoulady et al. [1] to improve on the TM and at the same time increase the sampling efficiency of the algorithm. Because of how the TM learns it can often get several versions of similar clauses that occur multiple times. The WTM was introduced to counter this effect by representing clauses as a single clause with a weight instead of having multiple clauses. The weight that is applied to each clause is also real valued, thus introducing fractional clauses. This can then create fewer and more compact clauses. Leading to a TM where less clauses are needed to get the same result. Less clauses then have a direct impact on the speed of the classification and the speed of the learning. The change from TM to WTM is not huge, and a few adjustments are needed. The Equation (2.22) gets modified to Equation (2.30).

$$s'(X) = \sum_{j=1}^{\frac{n}{2}} \omega_j^1 C_j^1(X) - \sum_{j=1}^{\frac{n}{2}} \omega_j^0 C_j^0(X). \tag{2.30}$$

The architecture of the WTM closely resembles that of the classical TM, with a key modification to integrate weights into the clause voting mechanism. As illustrated in Figure 2.21, the majority vote pipeline of the classical TM is altered to accommodate this feature. This simple yet effective adaptation allows the WTM to integrate seamlessly into existing TM frameworks, leveraging the basic TM structure while enhancing decision-making capabilities through weighted voting.

Learning with the WTM is done in a similar manner to the TM. First all the weights are set to 1, that way the WTM works just like the TM does at the first iteration. Then the weights need updating. This is done in the type one feedback and type two feedback. The weights are only updated when the output of a clause evaluates to 1 for the current input x. Each clause that has output 1 as its result, will get its weight updated by a learning rate of $\gamma \in [0, \infty)$. The new weight, $w_n$, is obtained by multiplying the old weight, $w_o$, by one plus the learning rate, as shown in Equation (2.31).

$$\omega_n = \omega_o \times (1 + \gamma). \tag{2.31}$$

One thing to note about the learning mechanism of the WTM, is that adjusting the TA's are done in the exact same way as in the TM.

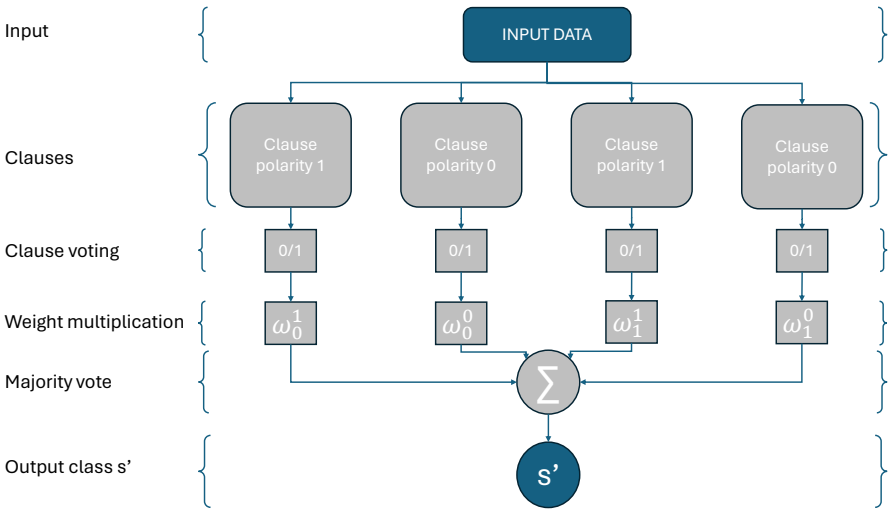



Figure 2.21: Pipeline of how input data is distributed to clauses. Clauses calculate their vote and the output is multiplied with its corresponding weight and then a majority vote predicts the output $s'$.

# Chapter 3

# Methods and System Design

## 3.1 Preprocessing

From the very start of this project we searched around online for bearing fault data and ended up with CWRU and NASA. Both of these were used for different types of bearing fault classifications. All the processing of the data has been done in Python, but this part sums up the common preprocessing done to all of the data for both the TM and the transformer.

The NASA files were text files of raw time-series data when downloaded originally, and the CWRU data was in the form of MATLAB files. The two datasets were separately processed, but the preprocessing steps were mostly the same. This is a simplified overview of the first preprocessing steps made to the data:

1. **Fast Fourier Transform** to make the raw time-series into spectrograms.
2. **Normalization**, adjusting the datapoints to range from 0 to 1.
3. **Decibel conversion** to change the signal into an amplitude value which is easier to see the full range of in a spectrogram.
4. **Split in two along the frequency axis** to remove the redundant half of the spectrograms.

These steps will be expanded upon in Section 3.1.1. For the TM models, additional preprocessing must be done. This will be detailed in Section 3.1.5.

### 3.1.1 Initial Preprocessing

The datasets were first converted into spectrogram using `numpy.fft.fft` A.2.1. There were two reasons for using spectrograms as the input data format. One reason is that they have shown to be an effective form for input in transformer models before, in models such as the AST. An assumption was made that this inherently makes spectrograms a good representation of a time-series, and this led to them being used in the TM as well. The other reason spectrograms were used is that they hold easily readable information about the change in frequency over time across the whole frequency spectrum. The manual processes involved in monitoring the signals that they employ at NOV, involve looking at certain frequencies as they change over time. Therefore, by looking at a spectrogram, it should be possible to see the indicators that experts look for. This is especially relevant for the problem of detecting bearing faults using binary classifiers.

The spectrograms were then normalized. This was done using a simplified version of the general min-max normalization formula shown in Equation (2.4), where we set the minimum

value $X_{min}$ to zero. This is justified by the nature of spectrogram data, where the values are inherently non-negative, making the minimal value effectively zero. The normalization formula used can be expressed as:

$$\text{spectrogram\_normalized} = \frac{\text{spectrogram}}{\max(\text{spectrogram})}. \tag{3.1}$$

This makes all spectrogram values scaled to a range between 0 and 1, relative to the maximum value observed in the dataset.

The next step was converting the spectrogram values to decibels using the formula shown in Equation (3.2). Lastly the files were split in half in the frequency axis so that the mirrored features were removed, due to it only adding complexity and computational waste. After preprocessing, the data was converted into MATLAB files. These steps are illustrated in Figure 3.1.

$$\text{spectrogram\_dB} = 20 \cdot \log_{10}(\text{spectrogram} + 10^{-6}). \tag{3.2}$$

Here, *spectrogram_dB* is the output of the conversion to decibels, 20 is chosen as the output should represent amplitude and not power, *spectrogram* is the input, and the addition of the very small value of $10^{-6}$ is to ensure the resulting output can't be zero.

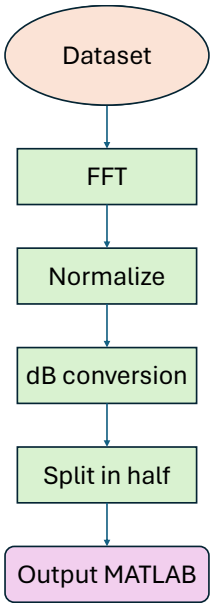



Figure 3.1: Steps involved in the initial preprocessing of datasets, converting raw input into a uniform and readable spectrogram without reduncancy.

$$\hat{x} = clip(x, 3\sigma - \overline{x}, 3\sigma + \overline{x}). \tag{3.3}$$

### 3.1.2 NASA Labeling

We were responsible for labeling the NASA dataset ourselves, as labels for the time of fault weren't already present in the dataset. Initially, we analyzed the spectrograms to identify rapid temporal changes in vibration signals. To enhance our understanding, we reviewed previous research papers on classification of the NASA spectrograms, but none had a concrete way to label them. We also received insight from NOV, who possess specialized expertise in analyzing and detecting bearing faults manually. In the end, we decided it would make the most sense to choose the label to be where the biggest temporal change happened across all frequencies, which is clearly visible to the eye in the example shown in Figure 3.2.

The NASA files vary in length and each has a unique segment where the fault begins, but all were labeled following the same principle. Figure 3.2 displays the shortest of the four NASA files.

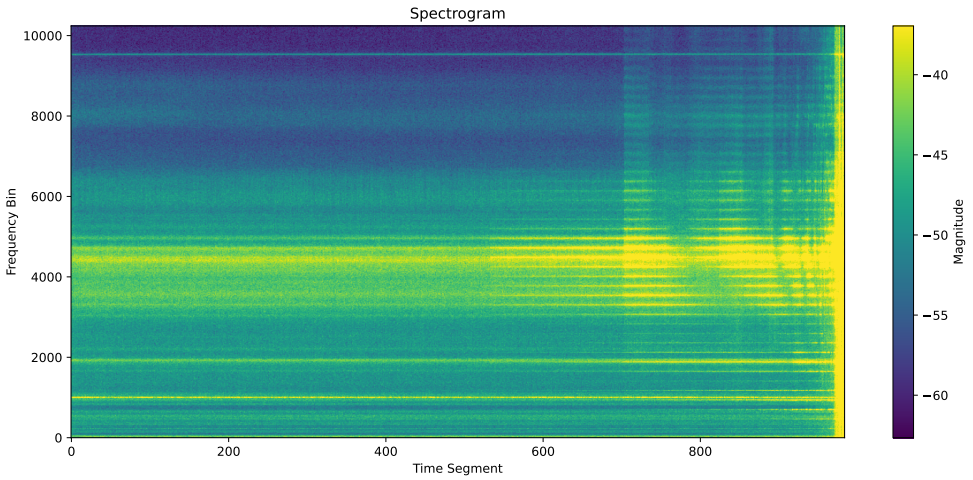



Figure 3.2: Spectrogram from NASA's Test 2, Channel 1, with a fault starting at 701. A vertical line indicates the first change in pattern, clearly visible in the image.

### 3.1.3 Data Division for Benchmark Model Validation

During the initial stages of model development, we attempted to utilize all available experiments from both the NASA and CWRU datasets. We implemented an 80/20 split on the data points, ensuring that both training and testing data were derived from the same experiments. This approach allowed the models to demonstrate promising performance levels. Subsequent hyperparameter tuning further optimized these results, enhancing the models' predictive capabilities and bringing them to be on par with existing models.

Despite the initial success, concerns regarding potential overfitting prompted the establishment of a benchmark set. For NASA, we designated one of the four experiments to serve exclusively as a benchmark set. Similarly, for the CWRU dataset, one experiment from each fault class was isolated to serve as the benchmark set. This isolation of one set of datapoints was intended to assess the models' generalization abilities beyond their training environment in a proper way. A strong argument for why it would be wrong not to isolate a set of data for benchmarking, is that each experiment is relatively constant across the time axis. When a model is tested on data points from an experiment it has already seen during training, and it shows good test results, this might indicate overfitting. This happens because time slices from one experiment are often very similar to each other. It could consequently display poor results on a new, previously unseen, experiment.

Upon re-evaluating the models with this new division of data into training, testing, and benchmark sets, a clear pattern emerged. While the models continued to achieve impressive results on the test sets, their performance on the benchmark sets was now only moderate. This discrepancy highlighted a potential overfitting issue, where models performed exceptionally well on the data they had encountered but struggled with completely new, unseen data. Because of the finding of overfitting we found out that testing on a completely unseen benchmark set was a good way to analyze the model's results.

### 3.1.4 Data Augmentation in Tsetlin Machines

Data augmentation is a pivotal technique in machine learning that can significantly enhance model performance, particularly by preventing overfitting. This technique becomes especially useful when a model, like the transformer, excels in handling sequential data. Inspired by this, we aimed to adapt similar strategies for the TM to harness the benefits of sequential data representation.

#### Implementing Sliding Window Technique

In our approach, we explored the potential of creating "sequential data" by inputting multiple time segments into the TM. This method intends to provide the model with insights into changes over time through the integration of different time sequences. However, a challenge arises in that increasing the input size of time segments reduces the total number of evaluation points available. For example, with 10 time slices and an input size of one segment, there are 10 total evaluation points. Increasing the input size to two segments reduces this number to five.

To address this reduction in evaluation points, we employed a data augmentation method known as the sliding window or rolling window technique [45]. This technique creates overlapping subsets of the original dataset by sliding a fixed-size window across the data, advancing a given step size for each time, thus generating some overlap in data points. In Figure 3.3, the application of a sliding window technique to a spectrogram is illustrated. This figure demonstrates how each data block holds multiple time segments, with overlap between consecutive blocks. For instance, examining block $n_1$, it becomes apparent that the data contained within this block also appears in both the previous block $n_0$ and the subsequent block $n_2$. This overlapping ensures that important temporal transitions between segments are preserved and can be analyzed effectively, facilitating a more comprehensive understanding of the time-dependent characteristics of the data.

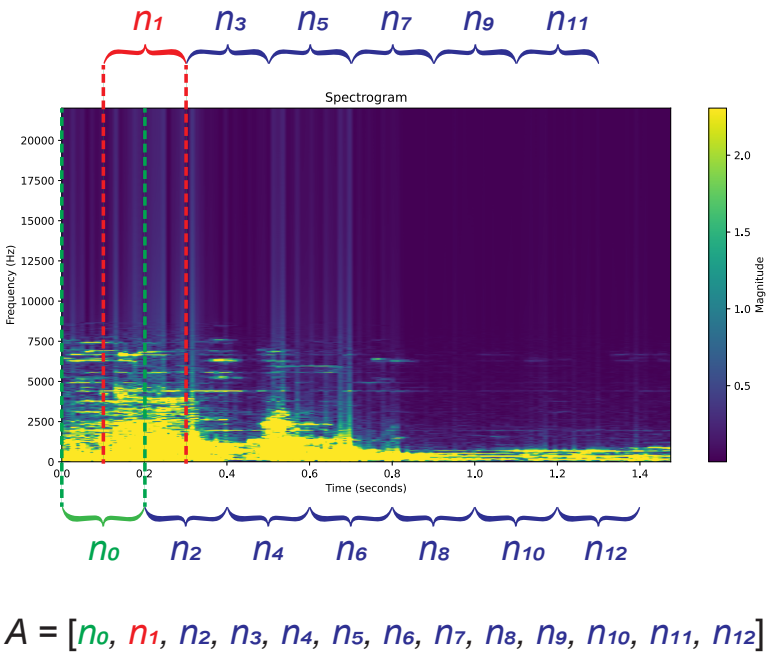



Figure 3.3: Spectrogram with sliding window applied with window size 0.2 and step size 0.1. The data blocks are appended to the set $A$.

### 3.1.5 Tsetlin Machine Data Input

Because the TM requires Boolean data as input, selecting an effective method for Booleanizing the data might be the most important step of the preprocessing. The majority of the steps required for the TM implementations lie in this Booleanization process. In Section 2.3.5, we explored various methods to achieve this. For this thesis, we have chosen not one, but three methods of Booleanizing the data:

- **Adaptive Gaussian Thresholding (AGT)**
- **Binary Embedding**
- **Thermometer Embedding**

These three methods of embedding the data into Boolean form are expected to yield variations in the forms of input data, capturing different aspects of the data that will then influence the TM. The use of these forms of embeddings will be further expanded upon below.

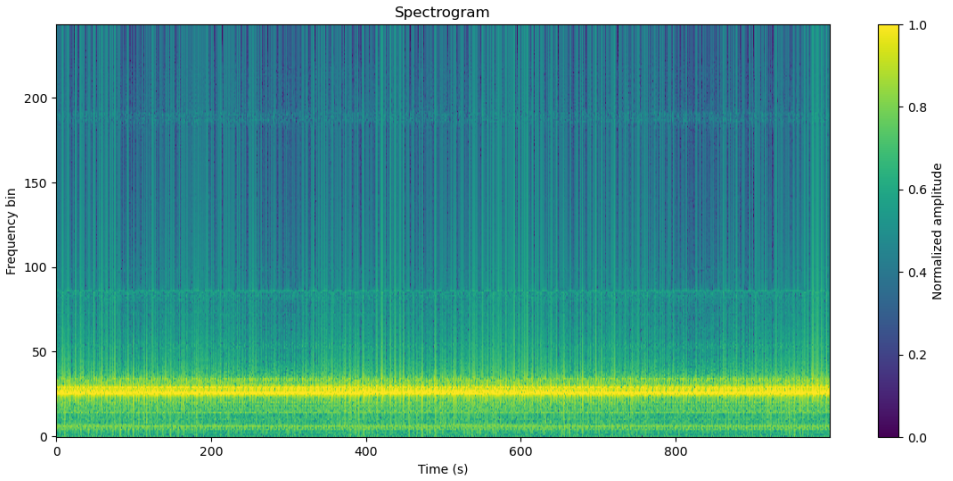



Figure 3.4: Normalized CWRU spectrogram where the x axis represents time and the y axis represents the frequency bin. The color represents the strength of the signal, where 0 is low amplitude and 1 is high amplitude.

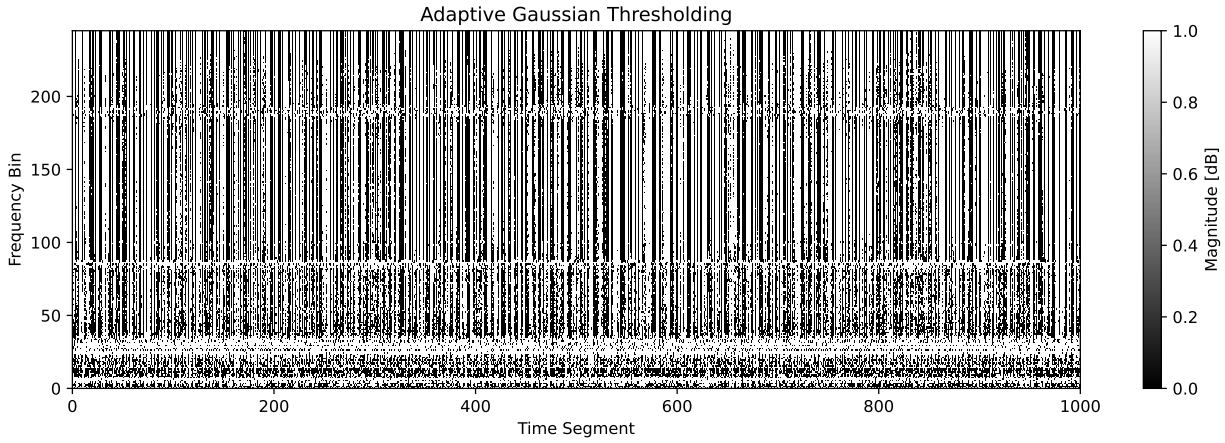



Figure 3.5: CWRU experiment with AGT applied. The block size is 51 and the constant is 2. There are only two colors in this image, black and white, represented by 0 and 1 respectfully.

**Adaptive Gaussian Thresholding Embedding**

The first method of embedding data for the TM involves using AGT. This process begins by converting the spectrogram into a grayscale image. This conversion was achieved by applying a `uint8` function, which maps the spectrogram data into an 8-bit single-channel image. In this format, pixel values range from 0 to 255, where 0 represents total black and 255 represents total white, effectively capturing the intensity spectrum of the spectrogram.

Once the spectrogram is transformed into a grayscale image, it is processed using OpenCV's implementation of AGT [35]. As discussed in Section 2.3.5, two primary parameters are essential for configuring AGT: the block size and the constant $C$.

By comparing Figure 3.4 and Figure 3.5, the impact of AGT on a typical CWRU experiment spectrogram is demonstrated. The application of AGT enhances the visibility of local features by emphasizing the differences between intensity levels in the spectrogram, making it especially suitable as a Boolean representation to be used in the TM. This method ensures that key spectral characteristics are retained and emphasized, making more of the spectrogram contain visible information.

**Binary Embedding**

Binary embedding, the second type of Boolean embedding used in our thesis. It involves dividing amplitude values equally among the specified number of bits. We tested with multiple amount of bits, one of them was 4 bits. For each of the 4 bits, a list is generated, with each list representing the corresponding index of the binary number. For example, the decimal value 5 is converted to the binary number 0101. The allocation of bits to the lists is as follows:

List 1: 1 (rightmost bit).

List 2: 0.

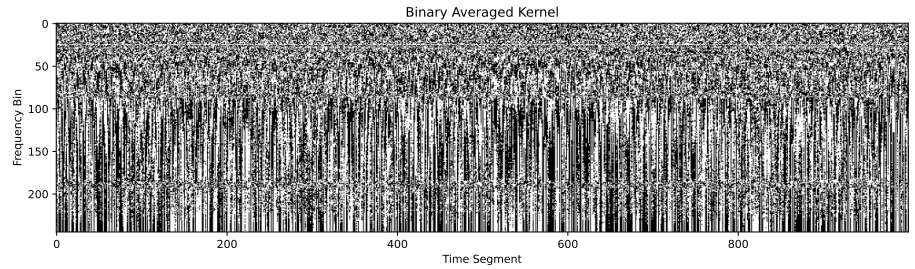
List 3: 1.

List 4: 0.

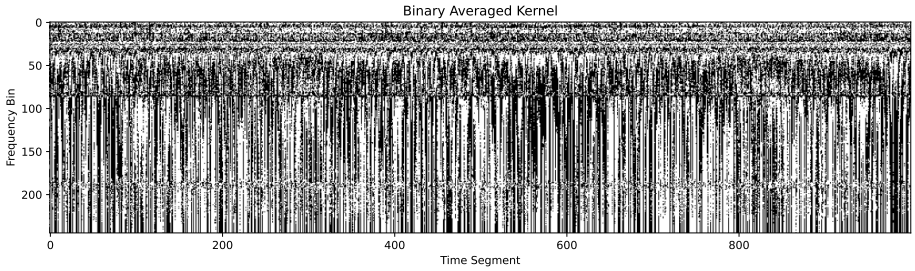
The process of applying binary embedding to a spectrogram begins at the bottom-left corner (y value 0, x value 0) and proceeds horizontally, incrementing the x value by one. Upon reaching the end of the x-axis, the y value is incremented by one, and the x value resets to 0. This procedure applies the binary embedding to each value, where each data point is converted into a binary number, and each index is assigned to its respective list.

Once the embedding process is complete, the number of lists equals the number of bits used, and each list represents a flattened array of the original spectrogram. These lists are then reshaped to match the original dimensions of the spectrogram, constructing the data from top left to bottom right. This creates a vertically mirrored image, but that does not affect the information contained within the data. This reconstruction can be visualized in Figure 3.6.

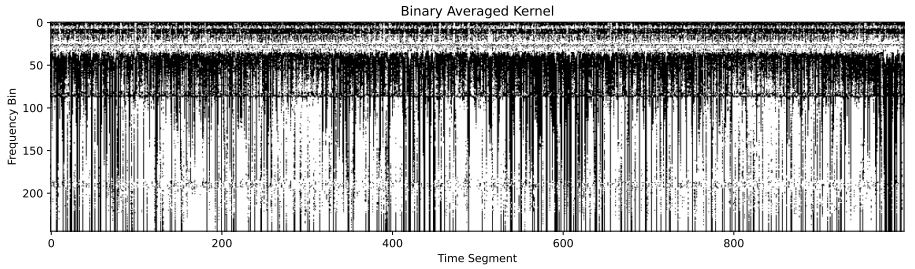
The individual binary maps are then concatenated to form a comprehensive binary representation of the spectrogram, as depicted in Figure 3.7. The dimensions of this binary representation are determined by multiplying the original height by the number of bits used, maintaining the original width of the spectrogram.



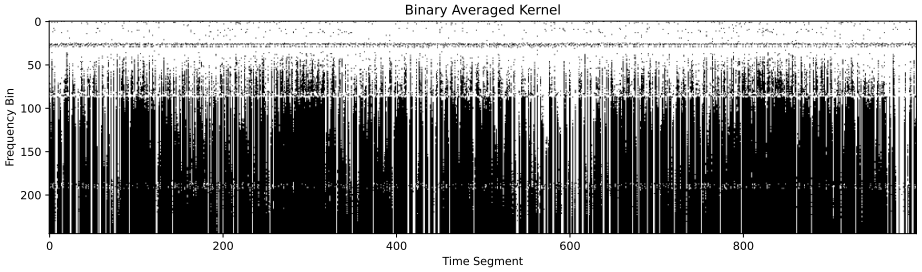

(a)




(b)




(c)




(d)

Figure 3.6: The different bit-maps that are created during the binary embedding process.

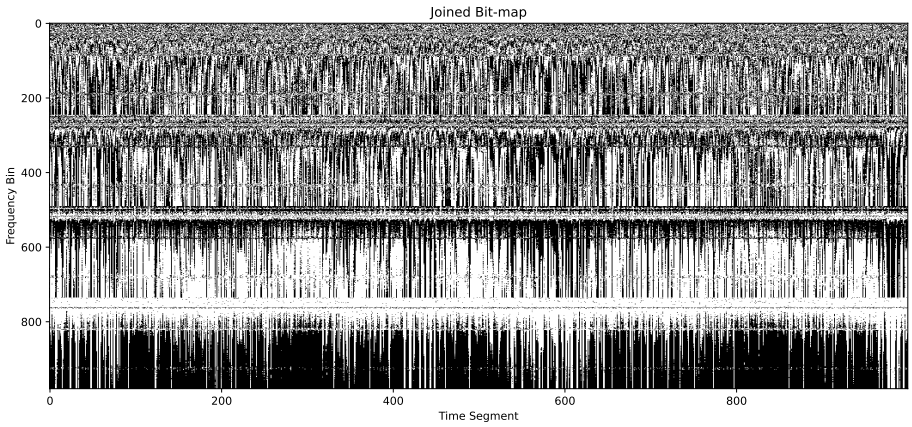



Figure 3.7: Joined bitmap of 4 bits. The x-axis displays time, the y-axis displays different frequency bins. Frequency bins from 0 to 245 are the first bit map, 246 to 491 are the second bit map, 492 to 737 are the third bit map, and 738 to 983 are the fourth bit map. The colors are black and white representing 0 and 1 respectfully.

### Thermometer Embedding

The third embedding type that this thesis examines for the TM, is the thermometer embedding. The initial step involves defining the thresholds-specifically, how much information each bit encodes. This is determined using Equation (3.4), which divides the range of data into segments corresponding to the number of bits.

$$\text{Threshold} = \frac{\max - \min}{\text{bits} + 1} \tag{3.4}$$

A list of thresholds is then created with a length equal to the number of bits. Each element of this list is assigned a value incremented by the index, starting from one, and each is multiplied by the threshold value derived from Equation (3.4).

The next step involves transforming each data value into a single-element array to facilitate element-wise comparisons. This comparison is performed against the thresholds: if a data value is greater than or equal to a threshold, it is assigned a value of 1; otherwise, it is assigned a value of 0. This process results in a binary array where each row corresponds to an original input value and each column to a threshold. An example of this will be shown below.

$$\text{Binary Value} = \begin{cases} 1 & \text{if value} \geq \text{threshold} \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

This Boolean encoding represents whether each input value meets or exceeds the respective thresholds, encapsulating the data into a Boolean vector with a length equal to the number of bits.

With the application of thermometer embedding, a three-dimensional tensor is formed, where each layer along the third dimension represents the Boolean encoding of the input values at different thresholds.

### Example Implementation

Consider a 2x2 matrix of random values between 0 and 1 as shown in Table 3.1. Setting the number of bits to 4, the threshold calculation adjusts according to Equation (3.6), which then establishes a list of thermometer thresholds to $[0.2, 0.4, 0.6, 0.8]$.

| | Column 1 | Column 2 |
|---|---|---|
| **Row 1** | 0.2685 | 0.6903 |
| **Row 2** | 0.0457 | 0.9984 |

Table 3.1: Example of a 2-dimensional table.

$$\text{Threshold} = \frac{1-0}{4+1} = \frac{1}{5} = 0.2 \tag{3.6}$$

Starting with the first element in the threshold list, 0.2, it is compared with the first matrix value, 0.2685. Since 0.2685 is greater than 0.2, a 1 is appended. This comparison continues with the next thresholds, resulting in a transformed value of $[1, 0, 0, 0]$ for the value in the first column and first row-0.2685. This method is applied to all values, with the results displayed in Table 3.2.

| | Column 1 | Column 2 |
|---|---|---|
| **Row 1** | 1, 0, 0, 0 | 1, 1, 1, 0 |
| **Row 2** | 0, 0, 0, 0 | 1, 1, 1, 1 |

Table 3.2: Example of a 3-dimensional thermometer table. Where the different colors of each element represent the third dimension.

Depending on the TM model used, the data can either remain in a three-dimensional form, or be transformed into a two-dimensional form. CTMs can handle three-dimensional data by taking sequences of two-dimensional data. MTMs, however, can only take sequences of one-dimentional inputs. Three-dimensional data therefore has to be converted into a two-dimensional form by stacking the data in the third dimension on top of each other in the height dimension. When using MTM thermometer embedding, this results in a a two-dimensional array where the height is the product of the original spectrogram's height and the number of bits used. The width remains the same as the original spectrogram's width. This transformation is depicted in Table 3.3.

| | Column 1 | Column 2 |
|---|---|---|
| **Row 1** | 1 | 1 |
| **Row 2** | 0 | 1 |
| **Row 3** | 0 | 1 |
| **Row 4** | 0 | 0 |
| **Row 5** | 0 | 1 |
| **Row 6** | 0 | 1 |
| **Row 7** | 0 | 1 |
| **Row 8** | 0 | 1 |

Table 3.3: Example of a 2-dimensional thermometer table of the transformed 3-dimensional thermometer table.

When applying the thermometer embedding on the CWRU spectrogram in Figure 3.4, it is possible to see the details in the thermometer embedding clearly, as shown in Figure 3.8.

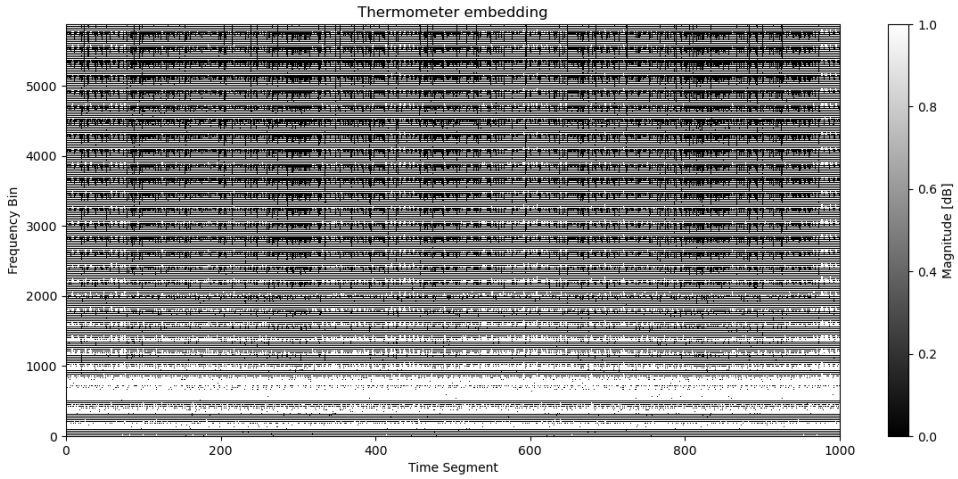



Figure 3.8: Image of a CWRU spectrogram where thermometer embedding is applied with 24 bits. The x axis displays the time segments, the y axis is frequency bins times 24. The different colors are black and white represented by 0 and 1 respectfully.

## 3.2 Transformer

### 3.2.1 Transformer Multiclass Classification

#### Data Preparation

The CWRU dataset consists of many MATLAB files, but they can be sorted into the 3 types of faults. This dataset is the one used for the multiclass classification model. The files were converted into a spectrogram in the frequency domain from a vibration signal in the time-domain as described in Section 3.1.1.

This is the pipeline for the further processing steps done specifically for the transformer multiclass classification model, also illustrated in Figure 3.9:

1. **Padding:** Firstly we use zero-padding on the files to get equal lengths for all spectrograms in the frequency axis. The time axis already had a length of 1000 for all files. Since the transformer needs the same input shape, it is necessary to pad the y-axis of the spectrograms. The spectrogram lengths initially varied from 243 to 245. The most beneficial number was 250 to make it more easily divisible for potential downsampling.

2. **Slicing:** The time axis of each spectrogram was sliced into slices with a width of 3. The reason for choosing 3 as slice width was because the transformer could learn the pattern and recognize it better than other widths tested. The trade-off of a wider slice is that it produces fewer slices to be used as inputs in the model, so a relatively small width of 3 gives the model more inputs to train and test on. Each CWRU file initially had 1000 data points in the time-axis. Using 80 % of all data points for training and 20 % for testing, we get a total of 19200 initial data points and in the end 6400 slices to train on from all files. Transformers are known to need quite a lot of data to perform well and this is the reason for choosing a low slice width. Other widths were also tested in order to find the optimal width of 3.

3. **Standardization and Normalization:** Each time-slice underwent standardization (subtracting the mean and dividing by 3 times the standard deviation) and was subsequently normalized to range from 0 to 1 using min-max scaling. This step is critical for neural network performance, ensuring that all input features contribute equally

to model training. Note that although the spectrogram consists of already min-max scaled data, it was normalized again after standardization as the min and max values can change as a result.

4. **Label Encoding:** To prepare for training the neural network, each fault type label was first converted into numeric format using `sklearn.preprocessing.LabelEncoder`, from the `sklearn` library referenced in Appendix A.2.3. This turns the categorical labels into a sequence of integer values, making them interpretable for the model. One-hot encoding the labels is important when approaching multiclass classification, so the integer encoded labels were then converted to one-hot encoded vectors. This format aligns with the output of the softmax layer in the transformer multiclass classification model.

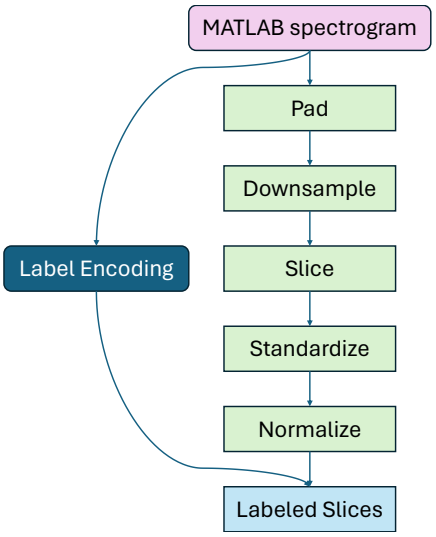



Figure 3.9: The additional steps taken in processing the spectrograms into the final input slices for the transformer.

### Multiclass Transformer Pipeline

Part of the core of our methodology is a transformer-based architecture designed to handle the sequential nature of the spectrogram slices effectively. The `TensorFlow.Keras A.2.3` library has been used to create the transformer model. The model captures dependencies across the sequence of data points within each slice, crucial for recognizing patterns indicative of different bearing faults.

- **Input:** The input for the transformer model takes in a vector with a sequence length of 3 and an embedding dimension length of 250.
- **Projection Layer:** The input is connected to a dense layer that connects the input of the model with the rest of the neural networks. The input is therefore scaled up to feed-forward dimension as a fixed input size. The dense layer is connected with the neural network training done in the transformer model. It helps with getting more information out of the input slices that are going into the transformer. The activation function used in the dense layer is "GELU." Projecting the slices to a higher dimension increases the model performance significantly.
- **Positional Encoding:** After the input embedding went through the dense layer, positional encoding is applied to each slice. The positional encoding is very important to let the transformer understand the order of each input embedding the model takes in. A formula is used to generate a unique positional encoding for each position in the

sequence of each input embedding. The formula used is the same as the formula in Equations (2.14) and (2.15) found in the Background chapter.

- **Transformer Block:** This is where the multi-head attention, feed-forward network and layer normalization takes place. We have used 3 transformer blocks, also illustrated in Figure 3.10. This is what each transformer block contains:
  - **Multi-Head Attention:** The model includes 4 attention heads which allows the model to learn and remember different features at different positions at the same time. The multi-head attention layer of the transformer block uses a dropout rate of 25
  - **Feed-Forward Network:** Since each attention output is processed through a feed-forward network, this enhances the model to see more features and generalize. The feed-forward dimension is set to 128. The activation function used for the feed-forward network is "GELU." A dropout rate of 25 % is used in this layer as well.
  - **Normalization:** This layer is applied twice throughout the transformer block, first after the multi-head attention and then after the feed-forward network. Normalizing promotes stability in the models training process.

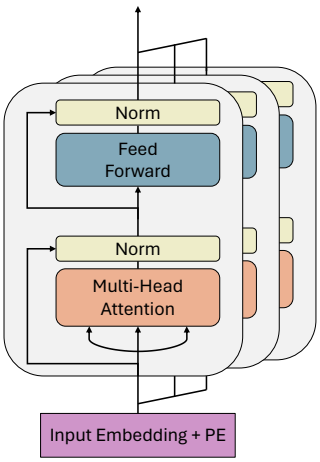



Figure 3.10: Configuration of the 3 transformer blocks.

- **Global Average Pooling:** The role of the global average pooling is to reduce the dimensionality of the output from the transformer blocks. This is an important step because it helps the model focus on the most important features. For example, if the output from the transformer block was "(batch size, sequence length, embedding dimension)", applying a pooling function would reduce it to "(batch size, embedding dimension)." This reduction simplifies the data structure. After reducing the output of the transformer block, it is connected to the MLP dense layers, which requires this new fixed size input.
- **MLP** This is where the neural networks are trained and the different layers that have been set up throughout the architecture are tuned. Having enough neurons for training is a key factor for model performance when setting up the MLP. In our transformer configuration, we utilized 64 neurons in the first layer and 32 in the second layer. The activation function "GELU" is applied with each dense layer of the MLP, which brings non-linearity, allowing the model to learn complex patterns. A dropout rate of 25 % was used in the dense layers within the MLP as well, in order to promote generalization.

- **Softmax and Output Classification** After processing through the transformer blocks and the subsequent layers, the transformed data finally reaches the output layer. This layer maps the high-dimensional features learned by the network into the final output classes. The "softmax" activation function generates the probabilities to determine the final class labels. These probabilities are used to classify the test and benchmark data.

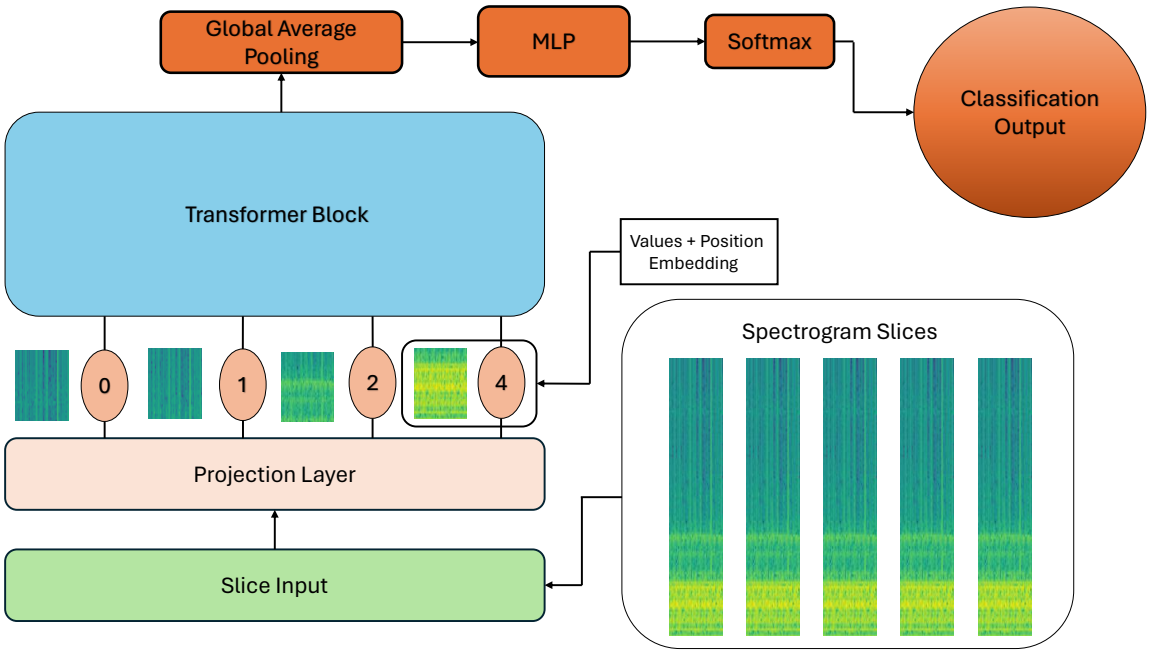



Figure 3.11: Multiclass classification transformer pipeline.

### Training and Validation

When implementing the model we used the `TensorFlowKeras` library, also referenced in Appendix A.2.3, to build and train the neural network. The optimizer used is Adam. The Adam optimizer adapts the learning rate during training, which can lead to faster convergence. The learning rate is set to 0.0001 and a categorical cross entropy loss function is used. Training was conducted over 70 epochs with a batch size of 32. We also used a keras library function called `ReduceLROnPlateau`, also referenced in Appendix A.2.3, to monitor the validation loss. This adjusts the learning rate when it detects that the models performance isn't improving anymore. This indicates a plateau in convergence. To evaluate the models performance and avoid overfitting, the data was split into training, test and benchmark. The training and testing set consists of 24 files, 80 % of which was used to train and the remaining 20 % was used to test on. The remaining files were used for benchmarking. Further, the files contain 1000 time series data points each, this means we get 19200 time slices for training and 4800 for testing. The bench marking gets 3 files that are separate from the training files. These 3 files comprise of one of each type of bearing fault. This ensures we validate the model on completely unseen data. The splitting of files is visualized in Figure 3.12.

To validate the performance of the multiclass model, these methods are employed:

- **Benchmark Accuracy:** Used as the overall performance of the model by testing how it performs on unseen data.
- **Confusion Matrix:** To see exactly which classification errors are present.

- **Benchmark Accuracy Convergence Graphs:** Making sure the model trains properly and seeing how the number of epochs influences the accuracy.
- **Mean, Standard Deviation, and Peak:** Taken from the accuracies of the last 20 epochs of a training run to see variations in accuracy.

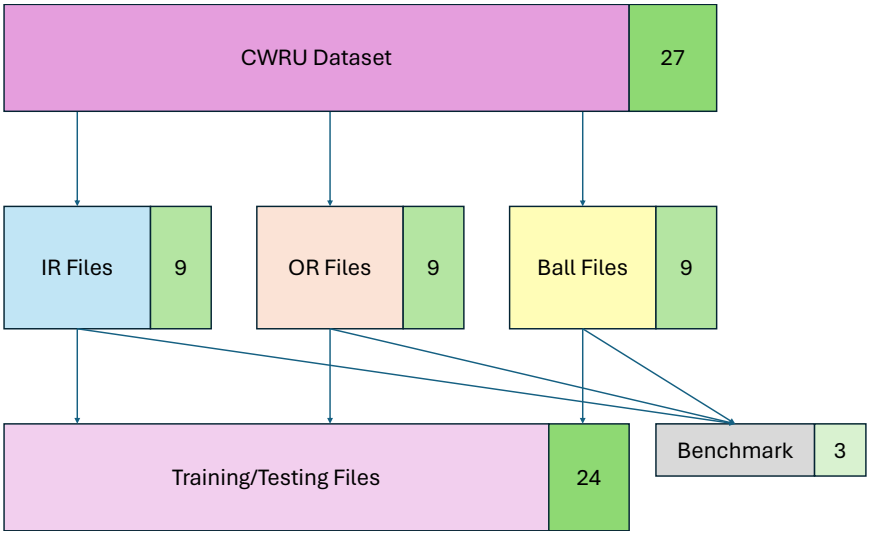



Figure 3.12: How the CWRU files are divided into a training/testing set and a benchmark set.

### 3.2.2 Transformer Binary Classification

After explaining the multiclass classification transformer which identifies types of bearing faults, this section focuses on the binary classification model. The binary model aims to detect the presence of a fault in bearings, which is important for preventative maintenance in industrial applications.

#### Data Preparation

All preprocessing steps except padding and labeling remain consistent with the multiclass model to maintain methodological consistency. There is no padding required for the NASA dataset since it has consistent frequency axis length. These are the additional processing steps that are unique to the binary classification model:

- **Downsampling** The length of the frequency axis is the same across all 4 experiments in the NASA dataset. The length of the frequency axis is originally 10240, derived from the sampling frequency of 20480. For the transformer to take in 10240 features it would take too much computational power, so it is downsampled with a factor of 20 down to 512 features, which still retains enough information to be trained on and give good model performance.
- **Labeling** The same way of slicing the data to a width of 3 as is done for the multiclass model, is also done for the binary classification model. The slices are then labeled according to the part of the file they are in. The NASA dataset is labeled into two categories: "Healthy" and "Faulty." The "Healthy" is labeled as 0 and "Faulty" is labeled as 1.

#### Binary Transformer Pipeline

The setup of this transformer architecture is very similar to the multiclass classification model. Here we explain the ways the binary transformer pipeline differs from the multiclass pipeline.:

- **Input:** The input for the binary classification transformer uses the same slice width of 3, but the embedding dimension is set to 512, as this performed the same as 1024 as embedding dimension but with less computational power required.

- **Projection Layer:** This part remains almost the same, except the activation function used is "ReLU."

- **Transformer Block:** The number of transformer blocks used is 3. These are the differences in configuration compared to the multiclass model:

  - **Multi-Head Attention:** The model includes 4 attention heads with a dropout rate of 30 %.
  - **Feed-Forward Network:** The feed-forward dimension is set to 256, also with a dropout rate of 30 %
  - **Normalization:** This layer is applied both after the multi-head attention and after the feed-Forward network.

- **Global Average Pooling:** The global average pooling is done the same way as in the multiclass classifier pipeline.

- **MLP:** The dropout rate is set to 30 % in the dense layers.

- **Sigmoid and Output Classification:** The binary classification version modifies the final output layer to feature a single neuron with a "Sigmoid" activation function instead of "Softmax", in order to have a binary output.

#### Training and Validation

When training this model, compared to the multiclass model, the only drastic change is the use of the binary cross entropy instead of categorical cross entropy as the loss function. This is what is used to get the loss of the binary output. The Adam optimizer is used again to fine tune the learning rate throughout the training process. The batch size used is 64 and 40 epochs are used for training. The same split percentages are used for training and testing.

Similar validation methods as in the multiclass classifier are used, but in addition to benchmark accuracy, f1-score is also used. This is because the number of datapoints for each class in the NASA dataset is not balanced, unlike with the CWRU dataset. These are the methods employed to evaluate the performance of the NASA binary classification transformer:

- **Benchmark Accuracy:** Used as the overall performance of the model by testing how it performs on unseen data.

- **Confusion Matrix:** To see exactly which classification errors are present.

- **F1-Score Convergence Graphs:** Making sure the model trains properly and seeing how the number of epochs influences the accuracy.

- **Mean, Standard Deviation, and Peak:** Taken from the f1-score of the last 20 epochs of a training run to see variations in f1-score.

- **Classification Error Position:** Plotted on top of the benchmark spectrogram to see exactly where in the time-axis the model makes classification errors.

## 3.3 Tsetlin Machine

Due to our initial unfamiliarity with the TM, we decided to implement multiple versions of it. For each classification task, whether multiclass or binary, we utilized both the MTM and the CTM.

The MTM models were extensively tested across a range of Booleanization methods to evaluate their effectiveness in different scenarios:

- **Adaptive Gaussian Thresholding Embedding:** AGT was initially tested for its potential to improve generalization by reducing feature redundancy and noise.
- **Binary Embedding:** This method was employed to enhance information retention by allowing a higher bit representation of features.
- **Thermometer Embedding:** Utilized for its ability to maintain ordinal relationships within the data, which is particularly beneficial in multiclass settings.

For the CTM, however, we exclusively applied thermometer embedding. This decision was based primarily on the time and scope of this thesis.

This diverse approach allowed us to systematically explore and compare the efficacy of different Booleanization techniques within the framework of TM technology, thereby identifying the most suitable methods for each type of classification task. The insights gained from these experiments can help refine our understanding and potential for implementations of TMs in various real-world applications.

### 3.3.1 Tsetlin Machine Multiclass Classification

#### Data Preparation of CWRU

As previously noted, the CWRU dataset comprises of numerous MATLAB files categorized into three types of faults, which are what our multiclass classification model aims to classify. These files, originally consisting of 2D vibration signals, have been transformed into spectrograms in the initial preprocessing steps.

This is the pipeline, which is also illustrated in Figure 3.13, for the preprocessing done specifically for the TM multiclass classification models:

1. **Zooming:** As with the transformer model, we want to make all the spectrograms the same shape. Originally, the time axis is 1000 in length in all CWRU spectrograms. This consistency is not present in the frequency axis, where we see fluctuations of 243 to 245 in the frequency bins. For the TM we utilized zoom functions from SciPY to make all the spectrograms have a height of 245 in the frequency axis. See Appendix A.2.2 for an overview libraries used.
2. **Standardization:** We now want to remove the outliers from the spectrograms, the way we do that is by using Equation (3.3). This makes all values that are higher than three standard deviations above the mean, $\mu + 3\sigma$, instead equal to three standard deviations above the mean. At the same time, all values that are lower than three standard deviations below the mean, $\mu - 3\sigma$, now equals to three standardization below the mean.
3. **Normalization:** The spectrograms are normalized using min-max again, to adjust the data to range from 0 to 1 again after zooming.

4. **Booleanize:** For the TM, all the zoomed and standardized spectrograms were then used as input into the embedding models. The options used for embedding models are AGT, binary, and thermometer. For AGT, the parameters used were a window size of 51 and a constant of 2. The binary embedding has only the bits as a parameter and the number of bits used was 3. The thermometer embedding also has bits as parameters, and 8 bits were used.

5. **Slicing and Data Augmentation:** After successfully applying various embedding models to the dataset, the input segments for the models are created. For the CTM, attempts were made to optimize performance by applying sliding window on the data. Unfortunately, these efforts did not yield the expected outcomes. This was mainly because of the size of the model and input creating problems in the memory of the computer used. In contrast, the MTM effectively handled bigger time segments and sliding window. Because of this the CTM utilized a window size of 2 and a step size of 1. For the MTM models, they used a window size of 5 and step size of 1.

6. **Label Encoding:** After the slicing and data augmentation of the model is done, each data slice is given its own class label. For the multiclass model each experiment contained in one file has one class. That means that every data slice gets the same class for the whole experiment. The labels are placed in a separated list of one dimension. The length of that list is equal to the number of data slices. Ball faults gets labeled as class 0, inner bearing faults gets labeled as class 1 and outer bearing faults gets labeled as class 2.

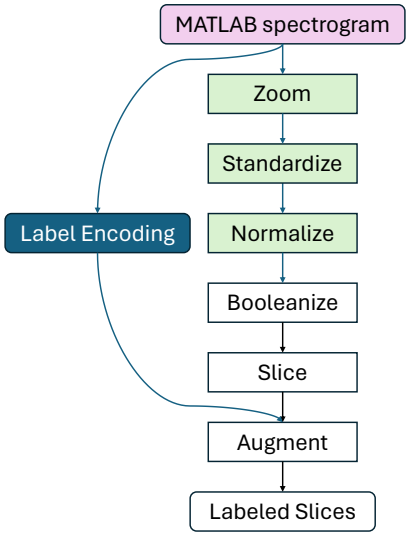



Figure 3.13: The additional steps taken in processing the spectrograms into the final labeled input slices for the multiclass classifying TM.

### Training and Validation

After the data is fully processed, sliced and labeled, it is split into the training set and benchmark set. This is done the same way as in the multiclass transformer model, where one file of each bearing fault type is set aside to be used for benchmarking as visualized in Figure 3.12. Of the remaining files, 80 % of the slices are used for training and 20 % are used for testing. The training is done by inputting the slices into the `fit` function of the `tm.MultiClassTsetlinMachine` class taken from the `PyTsetlinMachineCUDA` library referenced in Appendix A.2.2. The best performing embedding model was the thermometer embedding using MTM. The $C$, $T$ and $s$ values used for the best performing configuration were 9800, 69 and 19.93, respectively.

To validate the performance of the multiclass TM, the same methods as in the multiclass transformer are employed.

### 3.3.2 Tsetlin Machine Binary Classification

#### Data Preparation of NASA

The NASA experiments have a different setup than the CWRU experiments. Each experiment goes from the start of the bearing lifespan to the destruction of the bearing. Because of this, the pipeline for how the TM handles the NASA data is slightly different than for the data from CWRU. The data from NASA is used for binary classification.

This is the pipeline for the preprocessing done specifically for the TM binary classification models, also illustrated in Figure 3.14:

1. **Downsampling:** Before the spectrogram gets standardized, it is downsampled. Each of the NASA experiments in the frequency bin axis is downsampled with a factor of 20. The reason for this was the speed and the size of the model. Having too much data could create problems with the program and make it crash because of memory issues. We used the zoom functions from SciPY [40] to do this. It is important to note that each of the experiments in NASA has a different length in the time dimension. The length of the time dimension is not modified at all when zooming.

2. **Standardization:** The data is standardized to lie between $\mu - 3\sigma$ and $\mu + 3\sigma$, same as in the multiclass models' pipeline.

3. **Normalization:** Normalizing using min-max again.

4. **Booleanization:** All the experiments then undergo Booleanize embedding: AGT, binary embedding and thermometer embedding, same as in the multiclass classifier. For AGT, the parameters used were a window size of 51 and a constant of 2. The binary embedding has only the bits as a parameter and the number of bits used was 3. The thermometer embedding also has bits as parameters, and 8 bits were used.

5. **Slicing and Data Augmentation:** The creation of data augmentation with high resolution did not work well on binary classification with the CTM either. That is why CTM got a window size of 2 and step size of 1, whilst the MTM models got window size of 5 and step size of 1.

6. **Label Encoding:** The separation of the different classes creates two different lists that needs to be labeled. The first part, stretching from start to the break point are classified as class 0. Whilst the last part stretched from the break point to end of file and is labeled as class 1.

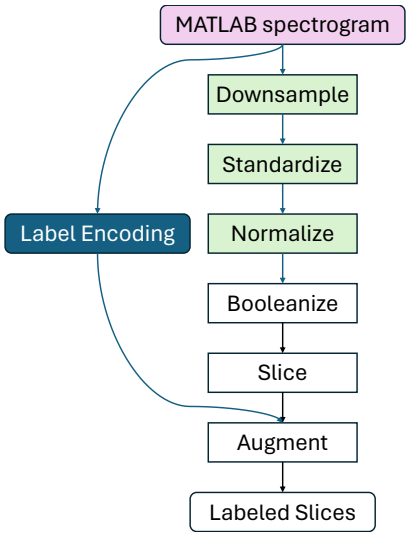



Figure 3.14: The additional steps taken in processing the spectrograms into the final labeled input slices for the binary TM.

### Training and Validation

The NASA files are divided the same way as they are for the transformer model, where one file is isolated to be used for benchmarking, while the remaining 3 files are used for training. The slices in the three files are then divided into 80 % for training and 20 % for testing. The `tm.MultiClassTsetlinMachine` class from the `PyTsetlinMachineCUDA` library was used for the NASA binary classification TM as well. The best performing embedding model was the thermometer embedding using MTM. The $C$, $T$, and $s$ values used for the best configuration were 20480, 97, and 21.80, respectively.

The same methods are employed to evaluate the performance of the NASA binary classification TM as are used for the NASA binary classification transformer.

# Chapter 4

# Results

This chapter shows the results from our research regarding the transformer and the TM. The chapter is laid out as follows, structuring it similarly to the trend through the whole thesis:



Each of the sub-sections here also follow a common structure, consisting of these elements:

- Best results from a single training run and average results from 5 training runs using the best configuration.
- Confusion matrix from a single training run using the best configuration.
- Training convergence graphs for a single training run using the best configuration.
- Variance, mean and peak performances across last 20 epochs from a single training run using the best configuration.
- For the NASA binary classification models, a graph showing which parts of the NASA benchmark spectrogram are classified erroneously.
- Other results that are not directly related to the best performing results.

## 4.1 Transformers

### 4.1.1 CWRU Multiclass Classifier Transformer

| | Test | Benchmark |
|---|---|---|
| Single Best Run | 98.40 % | 100 % |
| 5 Runs Average | 99.49 % | 99.27 % |

Table 4.1: Single best and average of 5 runs of the CWRU multiclass classification transformer. GELU was used with 6 attention heads, slice width 3.

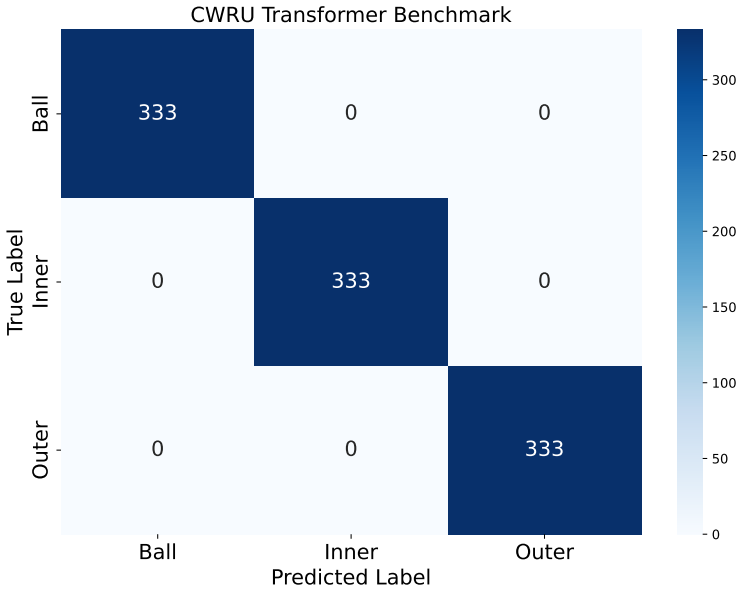
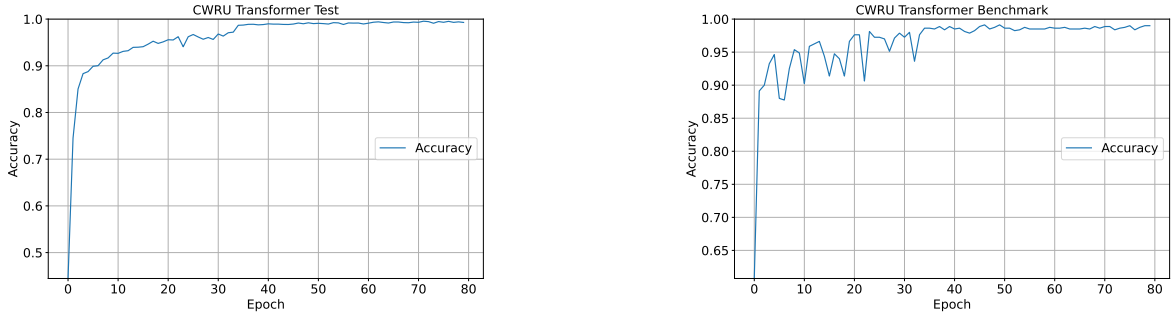


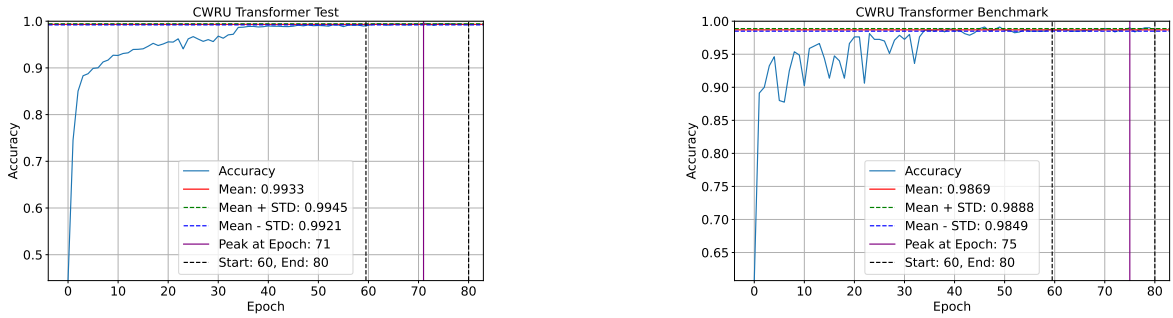

Figure 4.1: Confusion matrix for a single CWRU multiclass classification transformer run using the best configuration.




(a) CWRU test set.

(b) CWRU benchmark set.

Figure 4.2: Accuracy convergence graph for a single CWRU multiclass classification transformer run using the best configuration.




(a) CWRU test set.

(b) CWRU benchmark set.

Figure 4.3: Mean, standard deviation, and peak of last 20 epochs visualized on the same graph as Figure 4.2.

| | Mean | Standard Deviation | Peak |
|---|---|---|---|
| Test | 99.33 % | 00.12 % | 99.55 % |
| Benchmark | 98.69 % | 00.20 % | 99.00 % |

Table 4.2: Mean, standard deviation and peak of the last 20 epochs for a single CWRU multiclass classification transformer run using the best configuration.

**Other CWRU Transformer Results**

3 types of activation functions are tested with two different slice widths and two different numbers of attention heads. The results below show the average of 5 runs tested with different combinations. They have all been run with 70 epochs, 32 batch size, 0.25 dropout, 3 transformer blocks, 128 feed-forward dimension and MLP layer with 64 neurons in the first hidden layer and 32 in the second.

| Activation Function | Slice Width | Attention Heads | Embedding Dimension | Test Accuracy | Benchmark Accuracy |
|---|---|---|---|---|---|
| ReLU | 3 | 6 | 250 | 99.09 % | **99.01 %** |
| ReLU | 5 | 6 | 250 | 99.13 % | 98.93 % |
| ReLU | 3 | 4 | 250 | 98.80 % | 98.88 % |
| ReLU | 5 | 4 | 250 | **99.40 %** | 98.86 % |

Table 4.3: CWRU transformer results using ReLU activation function and varying slice width and number of attention heads.

| Activation Function | Slice Width | Attention Heads | Embedding Dimension | Test Accuracy | Benchmark Accuracy |
|---|---|---|---|---|---|
| GELU | 3 | 6 | 250 | **99.62 %** | 99.26 % |
| GELU | 5 | 6 | 250 | 99.38 % | 98.97 % |
| GELU | 3 | 4 | 250 | 99.28 % | 99.14 % |
| GELU | 5 | 4 | 250 | 99.49 % | **99.27 %** |

Table 4.4: CWRU transformer results using GELU activation function and varying slice width and number of attention heads.

| Activation Function | Slice Width | Attention Heads | Embedding Dimension | Test Accuracy | Benchmark Accuracy |
|---|---|---|---|---|---|
| Swish | 3 | 6 | 250 | **99.48 %** | **99.43 %** |
| Swish | 5 | 6 | 250 | 99.02 % | 99.36 % |
| Swish | 3 | 4 | 250 | 98.92 % | 99.29 % |
| Swish | 5 | 4 | 250 | 99.05 % | 99.20 % |

Table 4.5: CWRU transformer results using Swish activation function and varying slice width and number of attention heads.

Different downsampling factors were tried for the CWRU model. The difference in convergence in test accuracy is shown in Figure 4.4. These 5 models with different downsampling factors were trained for 140 epochs using a slice width of 3, batch size of 32, dropout rate of 0.25, 3 transformer blocks, 4 attention heads, 128 in feed-forward dimension, GELU activation function and MLP layer with 64 neurons in the first hidden dimension and 32 in the second.

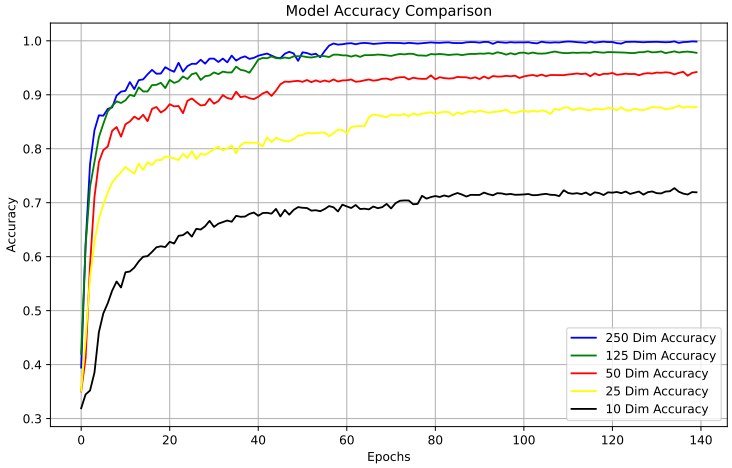



Figure 4.4: Comparison of accuracies in the CWRU transformer model training using downsampling factors of 1, 2, 5, 10 and 25.

### 4.1.2 NASA Binary Classifier Transformer

| | Test | Benchmark |
|---|---|---|
| Single Best Run | 98.93 % | 99.86 % |
| 5 Runs Average | 99.61 % | 98.72 % |

Table 4.6: Single best and average accuracies of 5 runs of the NASA binary classification transformer. ReLU was used with 4 attention heads and slice width 3.

| | Test | Benchmark |
|---|---|---|
| Single Best Run | - | 99.72 % |
| 5 Runs Average | 97.09 % | 97.45 % |

Table 4.7: Single best and average f1-scores of 5 runs of the NASA binary classification transformer. ReLU was used with 4 attention heads and slice width 3.

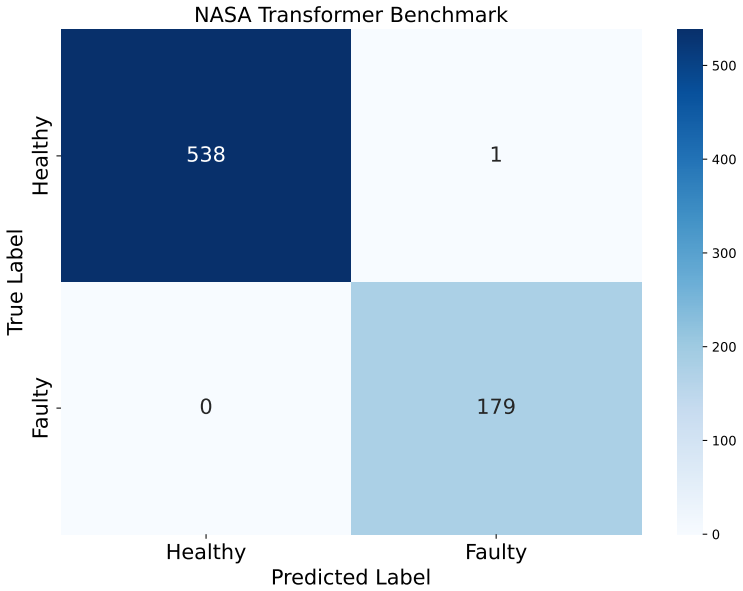
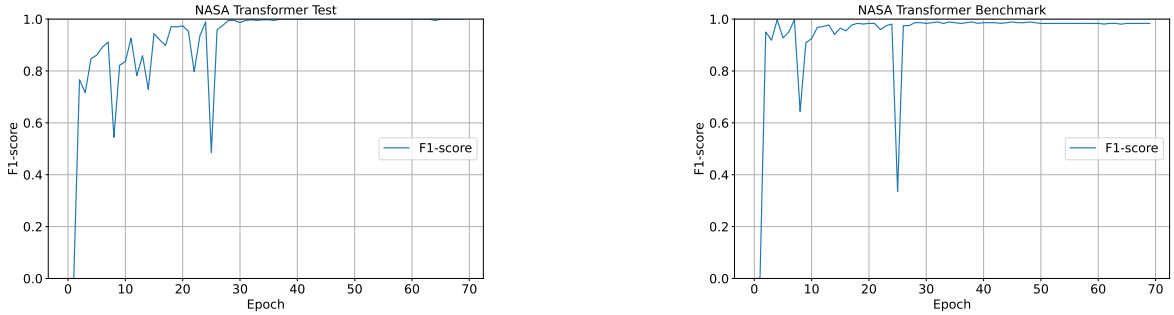


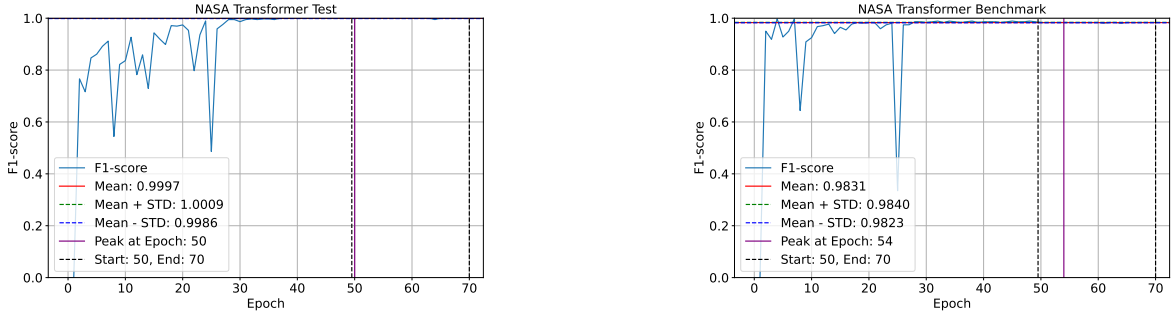

Figure 4.5: Confusion matrix for a single NASA binary classification transformer run using the best configuration.




(a) NASA test set.

(b) NASA benchmark set.

Figure 4.6: F1-score convergence graph for a single NASA binary classification transformer run using the best configuration




(a) NASA test set.

(b) NASA benchmark set.

Figure 4.7: Mean, standard deviation, and peak of last 20 epochs visualized on the same graph as Figure 4.6.

| | Mean | Standard Deviation | Peak |
|---|---|---|---|
| Test | 99.97 % | 00.12 % | 100.00 % |
| Benchmark | 98.31 % | 00.09 % | 98.35 % |

Table 4.8: Mean, standard deviation and peak of the last 20 epochs for a single NASA binary classification transformer run using the best configuration.

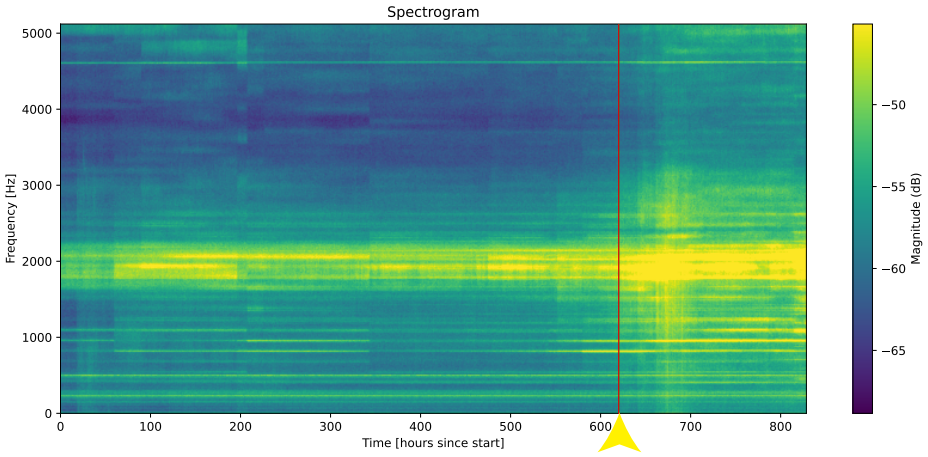



Figure 4.8: Transformer classification error in the fault detection in NASA experiment 1 channel 6, marked by a red line on top of the spectrogram data. The point between healthy and faulty bearing state is shown as a yellow line, further indicated by a yellow arrow.

### Other NASA Transformer Results

3 types of activation functions are tested with two different slice widths and two different numbers of attention heads. The results below show the average of 5 runs tested with different combinations. They have all been run with 40 epochs, 64 batch size, 0.3 dropout, 3 transformer blocks, 256 feed-forward dimension and MLP layer with 128 neurons in the first hidden layer and 64 in the second.

| Activation Function | Slice Width | Attention Heads | Embedding Dimension | Test Accuracy | Benchmark Accuracy |
|---|---|---|---|---|---|
| ReLU | 3 | 6 | 512 | **99.98 %** | **99.47 %** |
| ReLU | 5 | 6 | 512 | 99.84 % | 98.93 % |
| ReLU | 3 | 4 | 512 | 99.61 % | 98.72 % |
| ReLU | 5 | 4 | 512 | 99.91 % | 99.12 % |

Table 4.9: NASA transformer results using ReLU activation function.

| Activation Function | Slice Width | Attention Heads | Embedding Dimension | Test Accuracy | Benchmark Accuracy |
|---|---|---|---|---|---|
| GELU | 3 | 6 | 512 | 99.94 % | **99.33 %** |
| GELU | 5 | 6 | 512 | **100 %** | 99.12 % |
| GELU | 3 | 4 | 512 | 99.99 % | 99.28 % |
| GELU | 5 | 4 | 512 | 99.85 % | 98.56 % |

Table 4.10: NASA transformer results using GELU activation function.

| Activation Function | Slice Width | Attention Heads | Embedding Dimension | Test Accuracy | Benchmark Accuracy |
|---|---|---|---|---|---|
| Swish | 3 | 6 | 512 | 99.87 % | 99.14 % |
| Swish | 5 | 6 | 512 | 99.96 % | 99.12 % |
| Swish | 3 | 4 | 512 | **99.99 %** | **99.14 %** |
| Swish | 5 | 4 | 512 | 99.96 % | 98.98 % |

Table 4.11: NASA transformer results using Swish activation function.

The different downsampling factors were also tested for the NASA model. The difference in convergence in test accuracy is shown in Figure 4.9 and f1-score is shown in Figure 4.10. These 8 models with different downsampling factors were trained for 70 epochs using a slice width of 3, batch size of 64, dropout rate of 0.3, 3 transformer blocks, 4 attention heads, 256 in feed-forward dimension, ReLU activation function and MLP layer with 128 neurons in the first hidden dimension and 64 in the second.

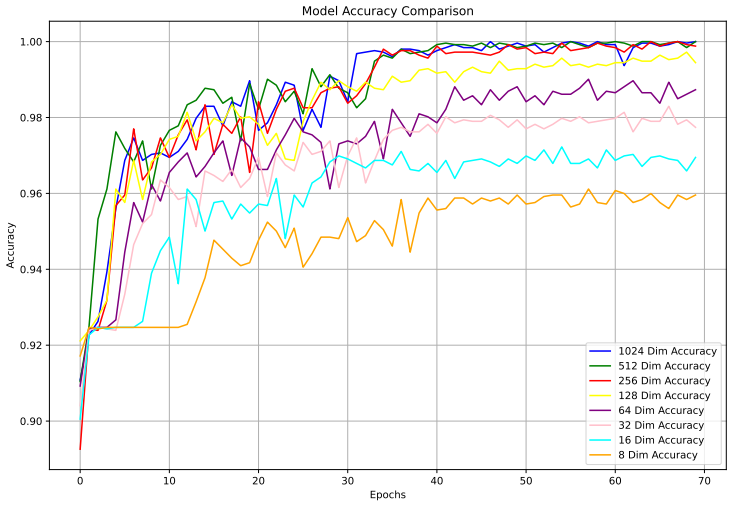



Figure 4.9: Downsample comparison of accuracy in the NASA transformer model

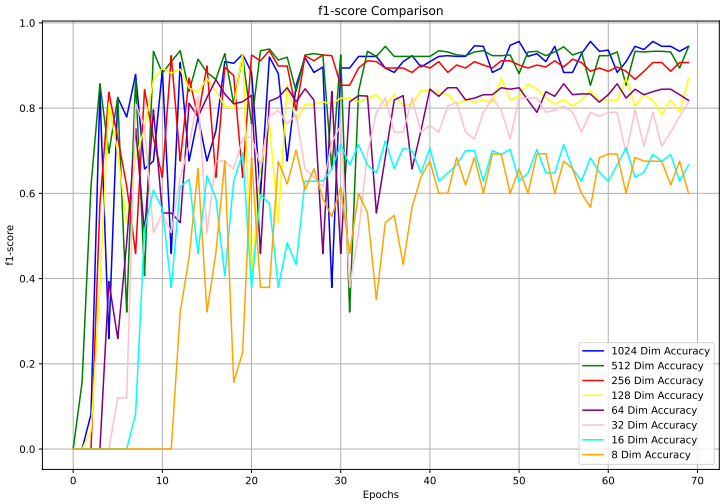



Figure 4.10: Downsample comparison of f1-score in the NASA transformer model

## 4.2 Tsetlin Machines

### 4.2.1 CWRU Multiclass Classifier Tsetlin Machine

| MTM Thermometer | Test | Benchmark |
|---|---|---|
| Single Best Run | 99.74 % | 99.83 % |
| 5 Runs Average | 98.34 % | 98.74 % |

Table 4.12: Single best and average of 5 runs of the CWRU multiclass classification TM. The best performing model was MTM thermometer. The parameters used was: $C = 9800$, $T = 69$, $s = 19.93$, bits $= 8$, window size $= 5$, step size $= 1$.

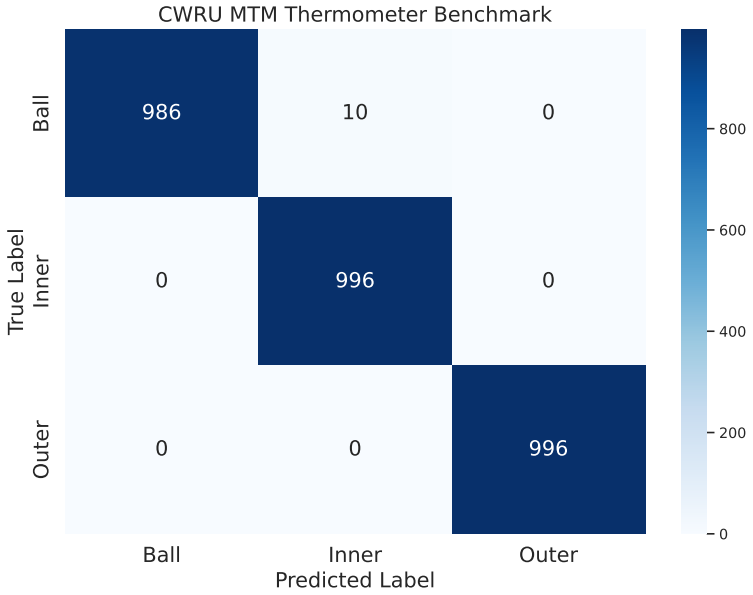
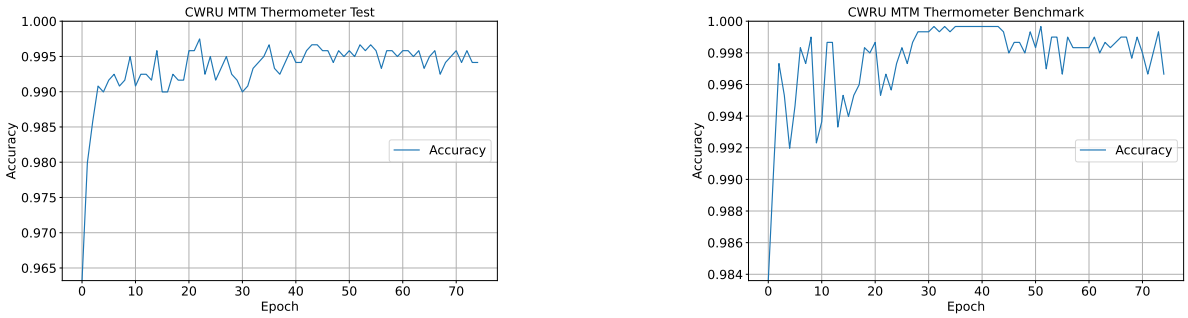


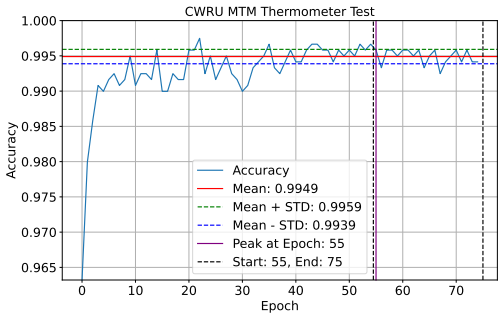

Figure 4.11: Confusion matrix for a single CWRU multiclass classification TM run using the best configuration.



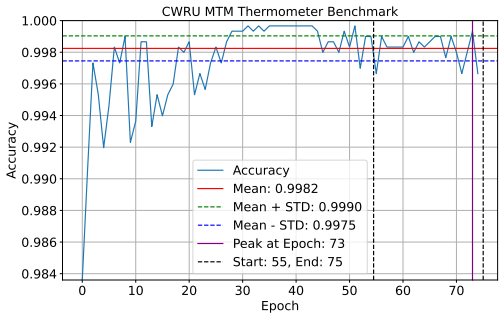

(a) CWRU test set.

(b) CWRU benchmark set.

Figure 4.12: Accuracy convergence graph for a single CWRU multiclass classification TM run using the best configuration.




(a) CWRU test set.




(b) CWRU benchmark set.

Figure 4.13: Mean, standard deviation and peak of last 20 epochs visualized on the same graph as Figure 4.12.

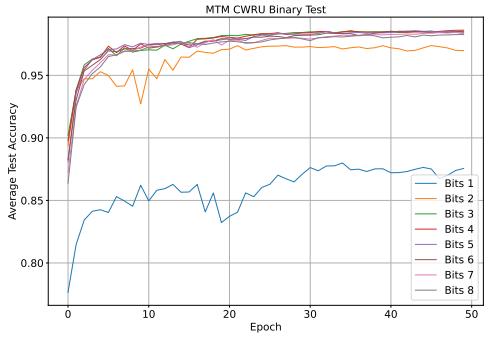
| | Mean | Standard Deviation | Peak |
|---|---|---|---|
| Test | 99.49 % | 00.10 % | 99.58 % |
| Benchmark | 99.82 % | 00.08 % | 99.93 % |

Table 4.13: Mean, standard deviation and peak of last 20 epochs for a single CWRU multiclass classification TM run using the best configuration.

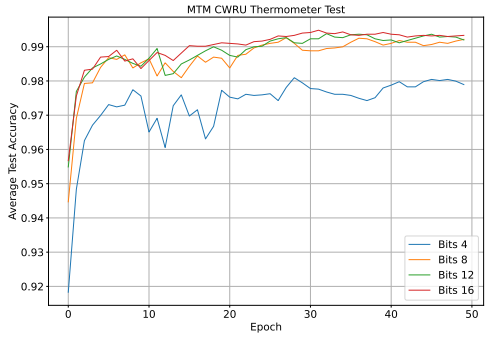
### Other CWRU Tsetlin Machine Results

| Model | $C$ | $T$ | $s$ | CWRU test | CWRU benchmark |
|---|---|---|---|---|---|
| MTM Binary | 3675 | 44 | 17.44 | 99.21 % | 99.73 % |
| MTM Thermometer | 9800 | 69 | 19.93 | **99.74 %** | **99.83 %** |
| CTM Thermometer | 490 | 17 | 12.34 | 97.06 % | 98.30 % |
| MTM AGT | 4900 | 51 | 18.18 | 98.49 % | 99.02 % |

Table 4.14: Best result from CWRU multiclass classification TM models.

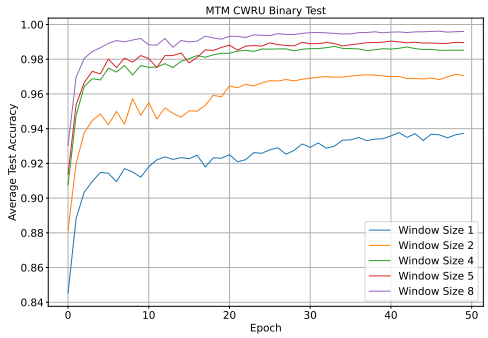



(a) CWRU binary test set.

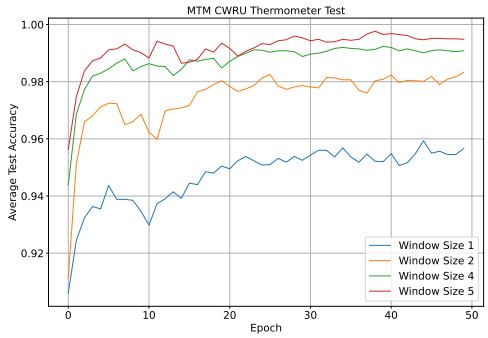



(b) CWRU thermometer test set.

Figure 4.14: The impact of bits on binary a) and thermometer b) embedding models. The accuracies is captured from 5 run average. Parameters used: $C = 3000$, $T = 41$, $s = 16.93$, window size $= 5$, step size $= 1$.




(a) CWRU binary test set




(b) CWRU thermometer test set

Figure 4.15: The impact of window size on binary a) and thermometer b) embedding models. The accuracies is captured from 5 run average. Parameters: $C = 3000$, $T = 41$, $s = 16.93$, bits $= 3$, step size $= 1$.

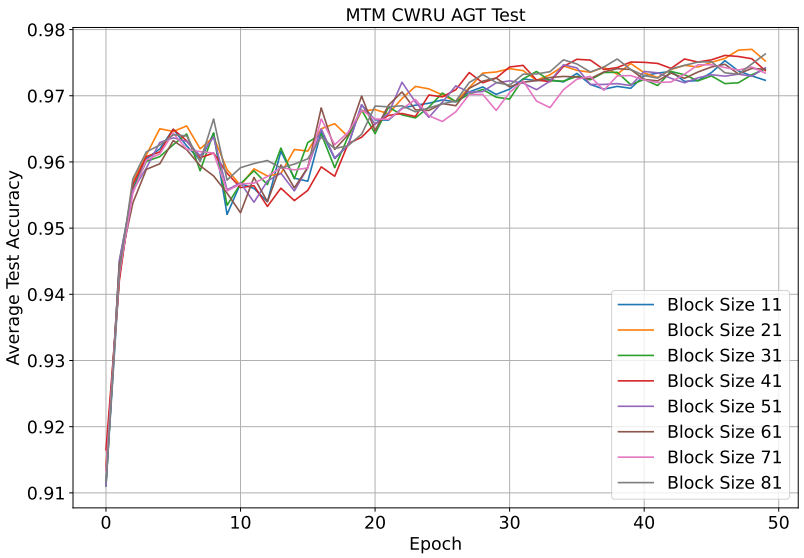



Figure 4.16: The impact of block size on the AGT embedding model. The accuracy is captured from 5 run average. Parameters used: $C = 3000$, $T = 41$, $s = 16.93$, window size $= 5$, step size $= 1$

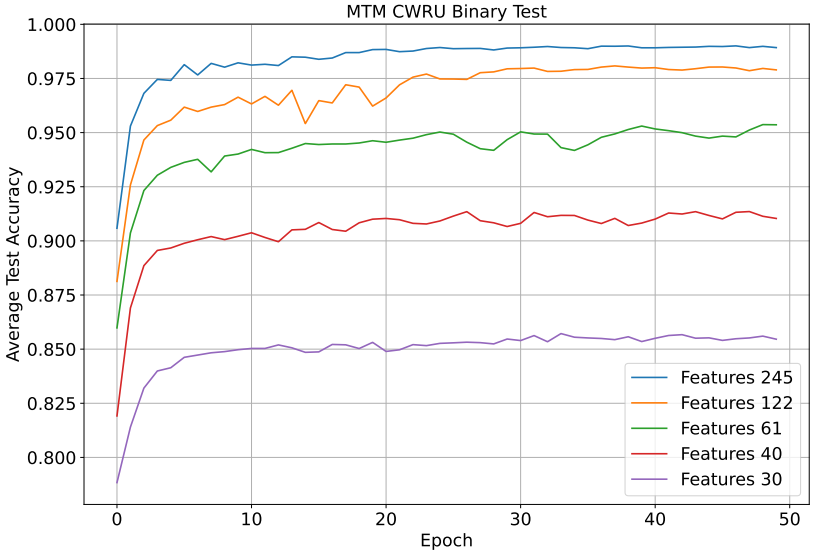



Figure 4.17: The impact of features on the binary embedding model. The accuracy is captured from 5 run average. Parameters used: $C = 3000$, $T = 41$, $s = 16.93$, bits $= 3$, window size $= 5$, step size $= 1$.

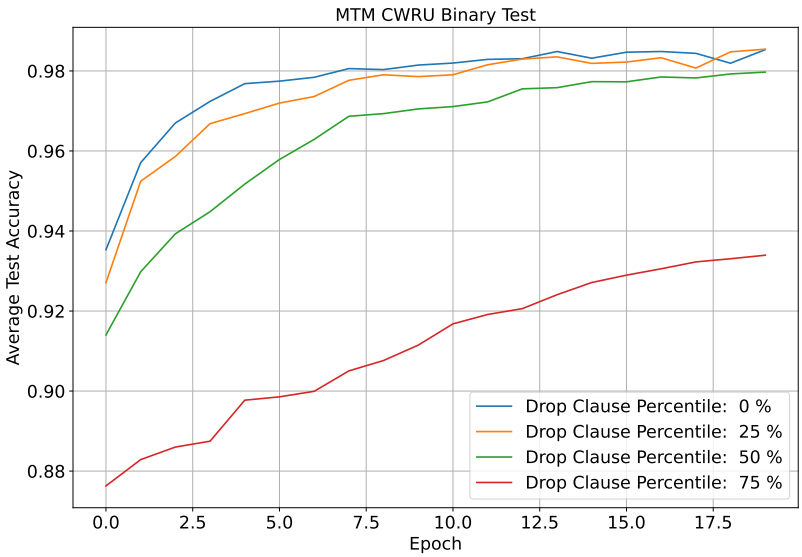



Figure 4.18: The impact of DC on the binary embedding model. The accuracy is captured from 5 run average. Parameters used: $C = 3000$, $T = 41$, $s = 16.93$, bits $= 3$, window size $= 5$, step size $= 1$.

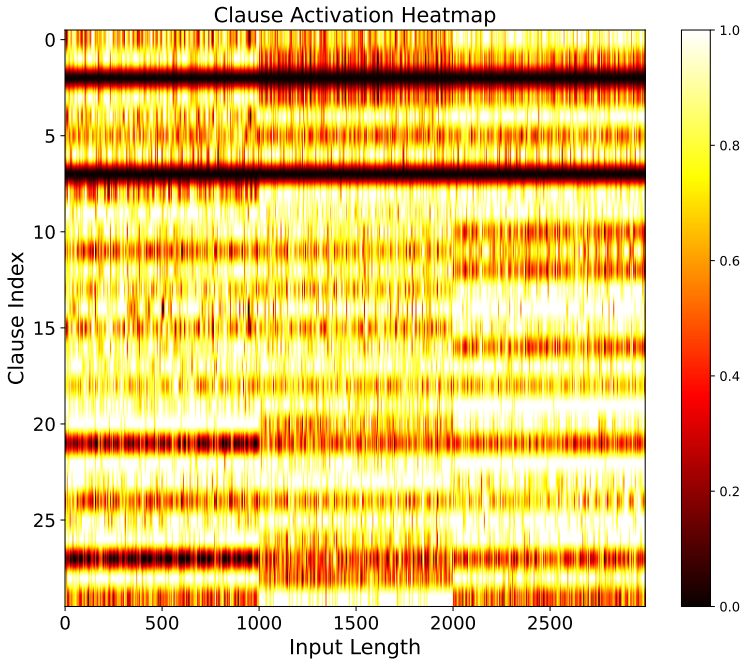



Figure 4.19: Heatmap of clauses activated on AGT where input was CWRU benchmark set. Parameters used was: $C = 10$, $T = 3$, $s = 2.48$, block size = 51, step size = 2, window size = 4, step size = 1.

### 4.2.2 NASA Binary Classifier Tsetlin Machine

| | Test | Benchmark |
|---|---|---|
| Single Best Run | 99.68 % | 99.67 % |
| 5 Runs Average | 99.76 % | 99.52 % |

Table 4.15: Single best and average accuracies of 5 runs of the NASA binary classification TM. The best performing model was MTM thermometer. The parameters used were: $C = 20480$, $T = 97$, $s = 21.80$, bits = 8, window size = 5, step size = 1.

| | Test | Benchmark |
|---|---|---|
| Single Best Run | 99.00 % | 99.57 % |
| 5 Runs Average | 99.24 % | 99.36 % |

Table 4.16: Single best and average f1-score of 5 runs of the NASA binary classification TM. The parameters used were: $C = 20480$, $T = 97$, $s = 21.80$, bits = 8, window size = 5, step size = 1.

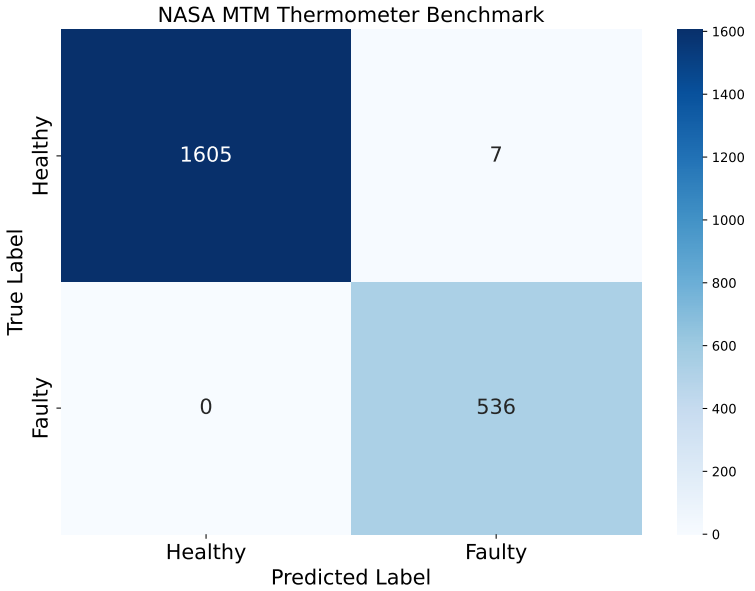
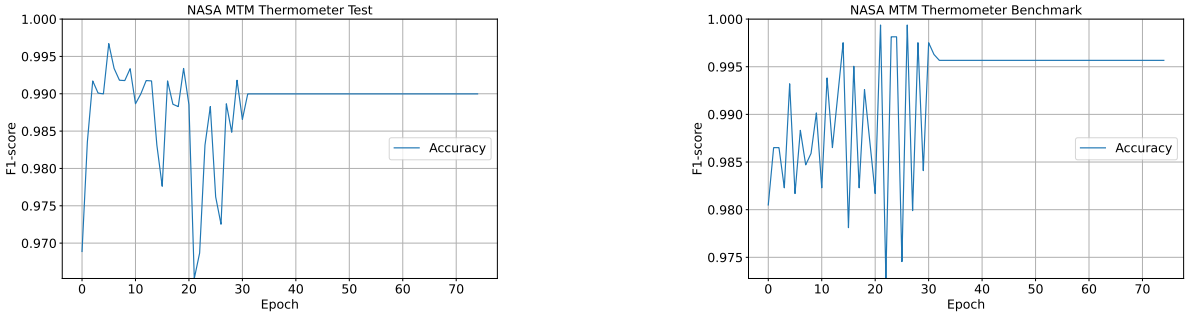


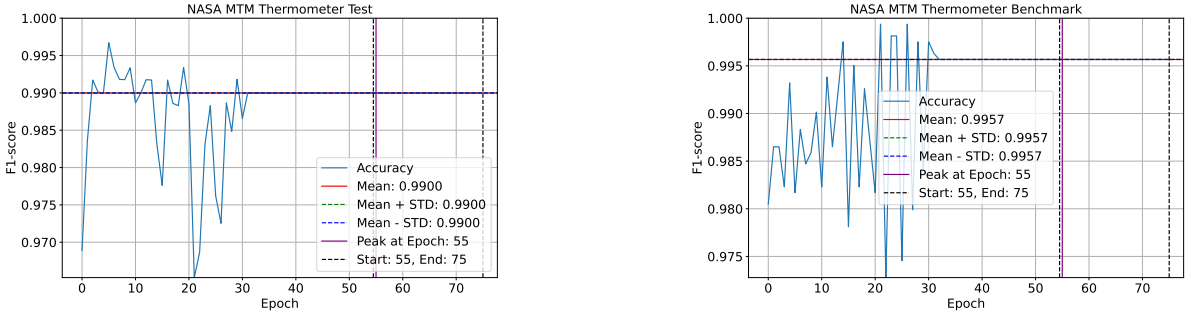

Figure 4.20: Confusion matrix for a single NASA binary classification TM run using the best configuration.




(a) NASA test set.

(b) NASA benchmark set.

Figure 4.21: F1-score convergence graph for a single NASA binary classification TM run using the best configuration.




(a) NASA test set.

(b) NASA benchmark set.

Figure 4.22: Mean, standard deviation and peak of last 20 epochs visualized on the same graph as Figure 4.21.

| | Mean | Standard Deviation | Peak |
|---|---|---|---|
| Test | 99.00 % | 00.00 % | 99.00 % |
| Benchmark | 99.57 % | 00.00 % | 99.57 % |

Table 4.17: Mean, standard deviation and peak of last 20 epochs for a single CWRU multiclass classification TM run using the best configuration.

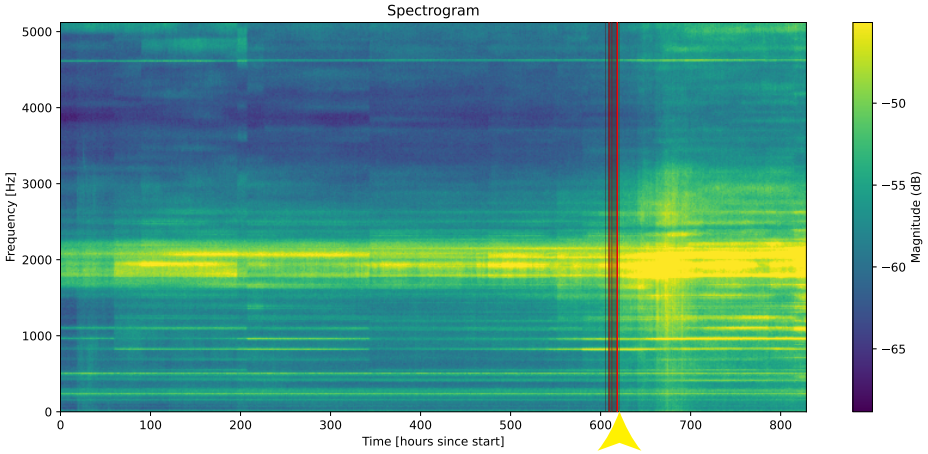



Figure 4.23: TM with binary embedding classification errors in the fault detection in NASA experiment 1 channel 6, marked by red lines on top of the spectrogram data. The point between healthy and faulty bearing state is shown as a yellow line, further indicated by a yellow arrow.

### Other NASA Tsetlin Machine Results

| Model | $C$ | $T$ | $s$ | NASA test | NASA benchmark |
|---|---|---|---|---|---|
| MTM Binary | 7680 | 62 | 19.32 | 92.53 % | 98.65 % |
| MTM Thermometer | 20480 | 97 | 21.80 | **99.00 %** | **99.57 %** |
| CTM Thermometer | 2560 | 38 | 16.53 | 96.44 % | 99.26 % |
| MTM AGT | 2560 | 38 | 16.53 | 85.04 % | 43.23 % |

Table 4.18: Best f1-scores acquired for all NASA binary classification TMs.

# Chapter 5

# Discussions

This chapter is a reflective discussion on the methodologies employed in this thesis, the challenges that were encountered, and the lessons learned throughout the course of this project. Each step of the project, from the initial problem identification to the final testing and validation stages, had different challenges and offer insights into potential future directions this work may take. This chapter will be a critique of our methodological choices and their effectiveness, as well as a discussion of the limitations and unplanned challenges that played a role in shaping the final results. Aspects that could be improved if this study is repeated will be discussed, aiming to provide more insight for potential extensions of the research.

## 5.1 Alternative Configurations

This section will discuss more about the different methods, configurations and concepts that were tested during the iterative development process, but ultimately not included in the final model configurations presented in the system design. Details about the different configurations, the reasoning behind their initial consideration, and a reflection on why these alternatives were less effective than the final methods, will be discussed here. This is intended to show the trial and error nature of the research and to inform about potential pitfalls in future studies.

### 5.1.1 Alternative Transformer Configurations

#### SMOTE

The Synthetic Minority Over-sampling Technique (SMOTE) technique was also tested on the NASA (binary classification) model. SMOTE takes the unbalanced data and creates synthetic data to balance out the minority data. This is done by adding small amounts of noise to the data points. However, in our experiments it didn't seem to make any difference on the NASA transformer model performance. The use of SMOTE was deemed unnecessary for this specific model configuration.

#### Transposed Input

We investigated alternative input configurations for the transformers by modifying the input values, one of which wherein the embedding dimension was set to correspond to the sliced width of the time series, and the sequence length was aligned with the frequency axis of the spectrogram. This experimentation was aimed at evaluating performance variations with these configurations as the slices of data could also be looked at as a sequence of varying frequency intensities with the time-axis as the embedding dimension. However, we observed that this approach significantly extended the training convergence time and demanded substantial computational resources. While the performance was satisfactory, it did not reach the optimal levels achieved by the more conventional configuration.

#### Activation Functions, Slice Widths and Attention Heads

The transformer models have been tried with different activation functions, slice widths, and numbers of attention heads. The differences in results for the CWRU multiclass classification models can be found in Table 4.3, Table 4.4 and Table 4.5. For the NASA binary classification models, the results can be seen in Table 4.9, Table 4.10 and Table 4.11.

Multiple activation functions were tried to see how ReLU compares to newer methods like Gaussian Error Linear Unit (GELU), and Swish. For the CWRU multiclass transformer, Swish along with 6 attention heads and slice width 3 had the best average benchmark results with 99.43 % as the average final performance of 5 training runs, but the best benchmark result was achieved with GELU. It performed with a benchmark accuracy of 100 % on the best single run using 4 attention heads and slice width 3. ReLU with 6 attention heads and slice width 3 had the best 5 run average benchmark accuracy for the NASA binary classifier transformer with 99.47 % benchmark accuracy. The single best run was better with 4 attention heads, however, which yielded 99.86 % benchmark accuracy.

### 5.1.2 Alternative Tsetlin Machine Configurations

Maximizing the TM results based on its configurations and preprocessing steps was done through an iterative process. Because of this, we tested many different solutions for all aspects of the system design.

#### Choosing Downsampling and Booleanization Methods

Initially, to address the long training time required for generating Boolean embeddings, we tried significantly reducing the feature space in our datasets. The NASA dataset was reduced from 10240 all the way to 20 frequency bins, which was later adjusted to 512 instead after big improvements in optimizing the Booleanization algorithms and seeing how much a larger feature space improved model performance. For the CWRU dataset, we didn't do any downsampling to preserve data granularity.

Multiple different methods for Booleanization were tested in this project. Binary embedding initially outperformed others, but expert consultations and further testing led us to look into the use of additional methods. Thermometer embedding was a method introduced to us through expert insight after consulting with Ole-Christoffer Granmo, the creator of the TM. We then focused mainly on binary embedding and thermometer embedding, but after refining hyper-parameters, optimizing preprocessing, and making data augmentation work, AGT was revisited with positive results. We then ended up using all three methods for the thesis.

#### Comparison of Booleanization methods

The comparison of results for the different Booleanization methods for CWRU is shown in 4.14, and for NASA is shown in 4.18. Results from the CTM with thermometer embedding are also shown. MTM with thermometer embedding performed best on both the CWRU multiclass classification TM and the NASA binary classification TM.

#### Data Augmentation

The transformer models excel in handling sequential data. A similar concept was attempted to be used in the TM by using multiple time slices as one input data point. This was done early on in the development process, and although minor, the results were improved. After

refining the Booleanization algorithms as previously mentioned, data augmentation played a bigger role in improving the accuracy and generalization of the TM models. This is what started closing the gap between the accuracy of the TM and the transformer models.

#### Zero Padding and Zooming

Prior to removing the 0 hp tests from our subset of the CWRU dataset, we encountered issues with aligning data points across different tests. We experimented with zero padding the spectrograms to get around this, but it did not yield any meaningful improvements. The zoom function was giving slightly better results, so it was used instead. At a later stage in the development, it was revealed that the 0 hp tests were significantly different in shape compared to all the other experiments. There was no specified reason for this in the documentation, so we removed the 0 hp tests from the subset that we used. This significantly improved the performance across all multiclass models. Despite the improvement, the zoom function was kept in the TM pipeline.

#### Hyper-Parameter Optimization

Initially, our TM testing phases often set the hyper-parameter $T$ to 80 % of $C$, and $s$ consistently stayed around a lower range of 3 to 15. The $s$ parameter had a relatively minor impact on the model performance compared to $C$ and $T$, illustrated in the results in Figure 2.19, where increasing $s$ visibly yields diminishing returns.

Further, experimentation with the settings suggested by O. Tarasyuk et al. [48], improved both the training time greatly, and improved the accuracy on benchmark data. This was done through finding optimal $T$ and $s$ values by following the suggestions in the mentioned study.

The selection of the hyper-parameter $C$ was strategically determined to scale with the size of the input data. Specifically, as the input size increases, the value of $C$ is adjusted upward to correspond proportionally to the expanded data input.

#### Drop Clause

While studying transformer architectures during the research phase of this project, we encountered the concept of dropout. This improves model generalization and combats overfitting. Applying this concept to the TM was immediately interesting as generalization is an important factor for this model. Jivitesh et al. [41] introduced DC.

Testing this concept of DC yielded positive results as shown in Figure 4.18 but the benefits began to diminish once the preprocessing techniques were optimized and hyper-parameters were tuned. The Python library supporting DC also greatly increased training time, so it was excluded from the final model configuration.

#### The Impact of Bits

The binary embedding method for Booleanizing the input for TMs, was initially done using 6 bits to 8 bits. However, throughout the development progress and the pipeline refinement, the amount of features we could include in the input data increased. This meant that the number of bits used in the binary embedding could be adjusted to fit better with the size of the feature space. The impact of changing the number of bits can be seen in Figure 4.14a and in Figure 4.14b. Seeing how small the impact was on performance from increasing the number of bits, we chose to utilize 3 bits on the binary models. The impact of different numbers of bits ranging from 1 to 8 is visualized in Figure 4.14.

#### The Impact of Window Size and Block Size

To find the optimal window size of 5, different sizes were compared to see the impact it had on the convergence of benchmark accuracy throughout the training epochs. The visualization of these convergences for the CWRU multiclass TM can be seen in Figure 4.15. Similarly, different block sizes, numbers of features, and DC rates were compared to find their optimal values. These are visualized for the CWRU multiclass TM in Figure 4.16, Figure 4.17, and Figure 4.18, respectively.

#### Interpretability Proof of Concept

A clause activation heatmap was constructed, shown in Figure 4.19. This serves as a proof of concept of the potential for interpretability in the TM models. It shows which clauses are activated in each spectrogram slice input. This can help an expert analyze how the TM to different inputs and why it makes the decisions it makes. Extracting more specific useful information would require more extensive domain knowledge.

## 5.2 Reflection and Future Work

This section of the chapter is intended as an evaluation of the broader implications of the research done in this thesis, the limitations and challenges encountered, the good and bad decisions made, and the potential future paths that can be taken around the topic of this thesis.

### 5.2.1 Reflection on Methodological Approaches

The methodology of this research was constructed to align with the challenges and potential theoretical contributions introduced by the problem statement. The project began with a comprehensive literature review to establish a solid foundation of the existing knowledge on different AI models used for similar tasks. This first phase of the project was essential to see exactly where there are gaps in current research on the topic.

The next phase would be to progress through the cycles of innovation, development and testing, then finally the validation. Each part of this was an extensive experimentation process where many different configurations of both data preprocessing and model structures were tested for both the transformer and the TM models. The iterative nature of this approach was tedious and required the availability of the computational power needed to be able to run tests frequently, but this allowed for continual refinement based on real numbers, ultimately leading to effective solutions that prove the potential of both the transformer and the TM for this type of application.

### 5.2.2 Limitations

This subsection outlines the limitations that were encountered throughout this project, which have influenced both the scope of the thesis and the final outcomes and findings.

#### Availability of Real-World Data

A significant contribution to this thesis that was expected was the incorporation of real-world data. This data was intended to validate the models developed in practical operational settings, demonstrating the applicability in real-world scenarios. There were attempts to get this data from NOV, but it was not received within the project timeline. This limitation restricted our ability to evaluate the model's performance in a comprehensive way, confining

us to the NASA and CWRU datasets. Another aspect to this is the fact that most real-world machines aren't run until failure, so datasets like the NASA bearing dataset are scarce.

#### Inconsistencies in Data

The CWRU dataset, while widely used for bearing fault diagnosis research, came with its own set of obstacle. It includes experiments with many varying conditions and setups, which introduced inconsistencies that complicated the training and validation of the models. To get around this, a subset of the dataset was selected for its relative consistency in experimental conditions. This decision limited the diversity and total amount of data available for training. This limitation shows the balance between data consistency and diversity.

#### Benchmarking

Previous studies that were used as a reference point throughout the development of the models in this project, often lack clear documentation on which data was used for testing, and whether the data was completely unseen or if datapoints from one experiment was used both for training and for testing. This made direct comparison of model performance difficult. In our approach, to ensure a valid model, one experiment from each fault class in the CWRU dataset, and one of the four experiments from the NASA dataset was isolated to be used exclusively for testing. This practice was not clear whether it was applied in previous research.

#### Documentation for Tsetlin Machine Library

The Python library for TMs was integral to this project, but it has limited documentation. This introduced challenges in understanding and implementing certain functionalities. However, this limitation had a reduced impact due to the access to direct communication with the creator of the TM, who is a professor at our university. This might not be the case for others conducting research on the TM, which underlines the importance of a comprehensive documentation available to the public.

#### Implementing TM Drop Clause

While developing the TM models, employing DC was considered. This is a dropout-like mechanism specifically for TMs, made to improve generalization by preventing overfitting. However, there were limitations to this in the library support. The library that supports DC does not currently support the use of CUDA, which would significantly accelerate the computations by using NVIDIA's GPU architecture. Using DC would have resulted in substantially increased training times, which was decided to be impractical for the scope of this project given the large datasets and complex configurations.

### 5.2.3 Potential Improvements

A more structured approach to the exploration of different transformer and TM configurations would have made the workload required to do this project lower, and potentially would have given us more time to structure the thesis itself after finding the best results. Finding a clear path on which modifications to try out earlier in the project might require a specific strategy.

#### Hands-On Experience

This project could have benefited from a deeper theoretical understanding and some hands-on experience with the two machine learning architectures that were used. This could also

have given us pre-emptive knowledge on which aspects of the transformers and TMs we should be focusing more on for this specific application of fault detection.

#### Structured Testing of Configurations

The testing of different configurations was often more reactive than proactive. This could have followed a more pre-planned, structured, approach, including predefined phases with specific objectives. On the other hand, the more reactive approach resulted in very promising results without having to test every possible combination of settings.

#### Data Augmentation for Transformer Models

Data augmentation was tested on in the early stages of developing the transformer models. It was specifically aimed at the binary classifier transformer model because of the limited amount of data. When tested at that time it didn't seem to have any significant impact, so it wasn't explored further. The impact might have been different after the model pipeline was significantly improved, but it was never tested.

### 5.2.4 Future Work

#### Further Improvements in Performance

Hyperparameters can be further tuned for both the transformer models and TM models through grid search or other algorithms for hyperparameter tuning [7] to find the full potential of the models' performance on these tasks. This will also enable the direct comparison between the TMs and transformers to see how they perform against each other on a level playing field. This wasn't done for this project as the results were promising enough to show the applicability of both models.

#### Deeper Exploration of Tsetlin Machine Applications

With proving the TM's capabilities for classifying temporal frequency data, many applications other than bearing fault classification and detection can be explored. This could be any signal analysis task where changes in the frequency spectrum across time are indicators to be detected. Examples include but are not limited to:

- **Vibration Analysis in Other Machinery:** This could be any other machinery components such as gears, motors and turbines, which can potentially indicate faults through changes in vibrations.
- **Audio Signal Processing:** Anything from speech recognition to environmental sound classification.
- **Electromyography (EMG) Signal Interpretation:** TMs could be used to analyze EMG signals, which measure electral activity produced by skeletal muscles.
- **Seismic Activity Monitoring:** The analysis of seismic data could be done using the TM architecture to improve earthquake prediction models.
- **Financial Market Analysis:** Financial time series analysis where the goal is to predict market trends or detect anomalies is another possible application for the TM, where interpretability could also be a groundbreaking benefactor.

**Interpretable Tsetlin Machine**

The direct application of the interpretability features of the TM can be further explored, analyzing heatmaps like the clause activation heatmap shown in Figure 4.19. This requires more research specifically aimed at this interpretability to further understand the information within the heatmap. Bridging the gap between the well performing TM model and the practical application of the heatmap is a research path with high potential for innovation.

**More General Detection Model**

The generalizability of the binary classification models seems like it can be significantly improved by incorporating a more diverse dataset. By training a model with a wide variety of data, it can more effectively identify faults within entirely new vibrational data it has never encountered before. Ideally, the model would be so robust and versatile that it could accurately detect faults across all bearing types, regardless of speed, size, or load. With more data available, a valuable path to branch off from this research would be to implement a general model for real-world applications by using a much larger dataset to train on.

# Chapter 6

# Conclusions

This thesis aims to contribute to the research within the application of advanced machine learning models for fault detection and classification in bearings in industrial machinery. Through testing and benchmark validation, the transformer and the TM architectures have been thoroughly evaluated, demonstrating high performance both in detecting and in classifying bearing faults.

The transformer model excels in capturing long-range dependencies within time-series data, which contributed to its accuracy in fault classification. This model is notably effective in processing sequential data, and its application resulted in high accuracy rates of 99.27 % average benchhmark accuracy over 5 runs for the CWRU multiclass classification task, and 99.13 % for the NASA binary classification task. The best training run results for a single training run were 100 % benchmark accuracy for the multiclass classification task, and 99.86 % for the binary classification task.

The TM, known for its potential for interpretability and its efficiency, also showed positive results. Its unique feature of handling inputs in a Boolean format and using propositional logic, opens the door for interpretable machine learning models as the need for them grows. This makes it applicable in the industry where understanding an AI's decisions is as crucial as its accuracy.

The contributions of this research is the extension of knowledge on the use and applications of transformers and TMs in fault detection, areas previously unexplored for these architectures. Secondly, the findings support the potential these models have to replace traditional techniques that are costly and potentially inaccurate. Additionally, TMs are generally underexplored for applications in temporal data analysis, a field they are now shown to be effective in.

In short, this research not only shows the effectiveness of using transformers and TMs in fault detection, but also opens up avenues for future exploration, especially in the field of predictive maintenance. The successful application of these models could lead to significant advancements in the industry, reducing downtime and unnecessary maintenance costs caused by bearing failures.

# Bibliography


[1] S. R. Gorji A. Phoulady O. G. Granmo and H. A. Phoulady. *The Weighted Tsetlin Machine: Compressed Representations with Weighted Clauses*. https://arxiv.org/abs/1911.12607. Accessed 05-April-2024. 2019.

[2] D. Abeyrathna, O. C. Granmo, and M. Goodwin. "Convolutional Regression Tsetlin Machine: An Interpretable Approach to Convolutional Regression." In: *Proceedings of the 2021 6th International Conference on Machine Learning Technologies* (2021). Accessed 24-April-2024. URL: https://api.semanticscholar.org/CorpusID:237424617.

[3] Adafruit. *Time Domain and Frequency Domain Representation Example*. Licensed under CC BY 3.0. URL: https://learn.adafruit.com/assets/11430. Accessed 15-April-2024.

[4] Open AI. *GPT-4*. Accessed 26-April-2024. Mar. 2023. URL: https://openai.com/research/gpt-4.

[5] T. Akan, S. Alp, and M. A. N. Bhuiyan. "ECGformer: Leveraging Transformer for ECG Heartbeat Arrhythmia Classification." In: *arXiv preprint arXiv:2401.05434* (2024). Accessed 19-April-2024. URL: https://ar5iv.org/abs/2401.05434.

[6] Anthropic. *Introducing the Next Generation of Claude*. Accessed 26-April-2024. Mar. 2024. URL: https://www.anthropic.com/news/claude-3-family.

[7] James Bergstra et al. "Algorithms for Hyper-Parameter Optimization." In: *Advances in Neural Information Processing Systems*. Accessed 10-May-2024. 2011. URL: https://papers.nips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html.

[8] Case School of Engineering. *Bearing Data Center - Seeded Fault Test Data*. Accessed 05-April-2024. URL: %5Curl%7Bhttps://engineering.case.edu/bearingdatacenter%7D.

[9] S. Chang. "New Algorithms for Processing Images in the Transform-Compressed Domain." In: *Other Conferences*. Accessed 06-May-2024. 1995. URL: https://api.semanticscholar.org/CorpusID:62568526.

[10] Z. D.Samimifard. *Preprocessing for CWRU Bearing Data Analysis*. Accessed 08-April-2024. 2023. URL: https://github.com/zahra88ir/MasterThesis/tree/main/preprocessing/CWRU.

[11] Robert Jr. Demaria et al. "Shuffling-Based Data Augmentation for Argument Mining." In: *AI³@AI*IA*. 2022. URL: https://api.semanticscholar.org/CorpusID:257584917.

[12] A. Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." In: *arXiv preprint arXiv:2010.11929* (2021). Accessed 19-April-2024. URL: https://arxiv.org/abs/2010.11929.

[13] N. Enshaei and F. Naderkhani. "Application of Deep Learning for Fault Diagnostic in Induction Machine's Bearings." In: *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*. Accessed 19-April-2024. 2019, pp. 1–7. DOI: 10.1109/ICPHM.2019.8819421. URL: https://ieeexplore.ieee.org/document/8819421.

[14] Hugging Face. *BERT 101 - State Of The Art NLP Model Explained*. Accessed 19-April-2024. 2023. URL: https://huggingface.co/blog/bert-101.

[15] Hugging Face. *Yes, Transformers are Effective for Time Series Forecasting (+ Autoformer)*. Accessed 29-April-2024. 2023. URL: https://huggingface.co/blog/autoformer.

[16] *Fourier Transforms (scipy.fft) — SciPy v1.13.0 Manual*. Accessed 22-April-2024. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.fft.html.

[17] Y. Gong, Y. Chung, and J. Glass. "AST: Audio Spectrogram Transformer." In: *arXiv preprint arXiv:2104.01778* (2021). Accessed 18-April-2024. URL: https://arxiv.org/abs/2104.01778.

[18] O. C. Granmo. *The Tsetlin Machine – A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic*. https://arxiv.org/abs/1804.01508. Accessed 05-April-2024. 2018.

[19] O. C. Granmo et al. *The Convolutional Tsetlin Machine*. Accessed 08-April-2024. 2019. URL: https://arxiv.org/abs/1905.09688.

[20] Great Learning. "Introduction to Pattern Recognition in Machine Learning." In: (2023). Accessed 18-April-2024. URL: https://www.mygreatlearning.com/blog/pattern-recognition-machine-learning/.

[21] R. P. Grimaldi. *Discrete and Combinatorial Mathematics An Applied Introduction 5th ed.* 5 th. Gerg Tobin, 2004, pp. 47–122.

[22] Liang Guo et al. "Deep Convolutional Transfer Learning Network: A New Method for Intelligent Fault Diagnosis of Machines With Unlabeled Data." In: *IEEE Transactions on Industrial Electronics* 66.9 (2019). Accessed 19-April-2024. DOI: 10.1109/TIE.2018.2877090. URL: https://ieeexplore.ieee.org/document/8511076.

[23] G. E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. https://arxiv.org/abs/1207.0580. Accessed 05-April-2024. 2012.

[24] The White House. *FACT SHEET: President Biden Issues Executive Order on Safe, Secure, and Trustworthy Artificial Intelligence*. Accessed 26-April-2024. Oct. 2023. URL: https://www.whitehouse.gov/briefing-room/statements-releases/2023/10/30/fact-sheet-president-biden-issues-executive-order-on-safe-secure-and-trustworthy-artificial-intelligence/.

[25] F. Hussain, U. Qamar, and S. Zeb. "A Novel Approach for Searching Linguistic Synonyms through Parts of Speech Tagging." In: *IEEE/WIC/ACM International Conference on Web Intelligence*. Accessed 06-May-2024. IEEE. Islamabad, Pakistan, 2016, pp. 465–468. URL: https://ieeexplore.ieee.org/abstract/document/7817093.

[26] IBM. *What Is Supervised Learning?* Accessed 18-April-2024. 2023. URL: https://www.ibm.com/topics/supervised-learning.

[27] S. Jaiswal. "What is Normalization in Machine Learning? A Comprehensive Guide to Data Rescaling." In: *Radar ai edition* (2024). Accessed 26-April-2024. URL: https://www.datacamp.com/tutorial/normalization-in-machine-learning.

[28] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. Accessed 18-April-2024. 2017. URL: https://arxiv.org/abs/1412.6980.

[29] Y. LeCun. *NMIST*. 1994. URL: https://www.kaggle.com/datasets/hojjatk/mnist-dataset. Accessed 08-April-2024.

[30] Tao Lu et al. "A Generic Intelligent Bearing Fault Diagnosis System Using Convolutional Neural Networks With Transfer Learning." In: *IEEE Access* 8 (2020). Accessed 19-April-2024, p. 164807. DOI: 10.1109/ACCESS.2020.3022840. URL: https://ieeexplore.ieee.org/abstract/document/9187800/.

[31] meta. *Introducing Meta Llama 3: The Most Capable Openly Available LLM to Date*. Accessed 26-April-2024. Apr. 2024. URL: https://ai.meta.com/blog/meta-llama-3/.

[32] microsoft. *Promptbase*. Accessed 26-April-2024. May 2024. URL: https://github.com/microsoft/promptbase.

[33] D.D. Miller. "The medical AI insurgency: what Physicians Must Know About Data to Practice with Intelligent Machines." In: *npj Digital Medicine* (2019). Accessed 20-April-2024. URL: https://doi.org/10.1038/s41746-019-0138-5.

[34] Kumpati S. Narendra and Mandayam A. L. Thathachar. “Learning Automata - A Survey.” In: *IEEE Trans. Syst. Man Cybern.* 4 (1974). Accessed 18-April-2024, pp. 323–334. URL: https://api.semanticscholar.org/CorpusID:17464562.

[35] OpenCV. *Image Thresholding*. 2024. URL: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html. Accessed 11-April-2024.

[36] European Parliament. *Artificial intelligence: Threats and Opportunities*. Accessed 26-April-2024. 2020. URL: https://www.europarl.europa.eu/topics/en/article/20200918STO87404/artificial-intelligence-threats-and-opportunities.

[37] European Parliament. *EU AI Act: first Regulation on Artificial Intelligence*. Accessed 26-April-2024. June 2023. URL: https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence.

[38] B. Peng et al. “A Survey on Fault Diagnosis of Rolling Bearings.” In: *Algorithms* 15.10 (2022). Accessed 18-April-2024, p. 347. DOI: 10.3390/a15100347. URL: https://www.mdpi.com/1999-4893/15/10/347.

[39] Z. D. Samimifard. “Using 2D CNN Models with Mid-Level Fusion-based approach for Multi-Classification of Bearing Faults.” Accessed 08-April-2024. Master’s thesis. Kongsberg, Norway: University of South-Eastern Norway, 2023. URL: https://openarchive.usn.no/usn-xmlui/handle/11250/3107216.

[40] SciPy Community. *Scipy.Ndimage.Zoom*. Accessed 06-May-2024. SciPy. 2024. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.zoom.html.

[41] J. Sharma et al. *Drop Clause: Enhancing Performance, Interpretability and Robustness of the Tsetlin Machine*. https://arxiv.org/abs/2105.14506. Accessed 05-April-2024. 2021.

[42] Alex Sherstinsky. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network.” In: *Physica D: Nonlinear Phenomena* 404 (2023). Accessed 18-April-2024. URL: https://arxiv.org/abs/1808.03314.

[43] Siemens AG. *The True Cost of Downtime 2022*. Accessed 18-April-2024. 2023. URL: https://assets.new.siemens.com/siemens/assets/api/uuid:3d606495-dbe0-43e4-80b1-d04e27ada920/dics-b10153-00-7600truecostofdowntime2022-144.pdf.

[44] L Simon. *Reasoning with Propositional Logic: From SAT Solvers to Knowledge Compilation*. Springer, Cham, May 2020, pp. 115–152.

[45] A. Singh. *Mastering Sliding Window Techniques*. Accessed 06-May-2024. 2023. URL: https://medium.com/@rishu__2701/mastering-sliding-window-techniques-48f819194fd7.

[46] N. Srivastava et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. https://jmlr.org/papers/v15/srivastava14a.html. Accessed 05-April-2024. 2014.

[47] Z. Sun et al. “Generating Diverse Translation by Manipulating Multi-Head Attention.” In: *arXiv preprint arXiv:1911.09333* (2020). Accessed 06-May-2024. URL: https://arxiv.org/abs/1911.09333.

[48] O. Tarasyuk et al. “Systematic Search for Optimal Hyper-parameters of the Tsetlin Machine on MNIST Dataset.” In: (2023). Accessed 18-April-2024. URL: https://doi.ieeecomputersociety.org/10.1109/ISTM58889.2023.10454969.

[49] V. Tyagi. *NASA/IMS Bearing Dataset*. 2020. URL: https://www.kaggle.com/datasets/vinayak123tyagi/bearing-dataset. Accessed 04-April-2024.

[50] National Oilwell Varco. *Personal Communication*. Meeting held at National Oilwell Varco office, Dvergsnesbakken 3, 4639 Kristiansand. Nov. 2023.

[51] Ashish Vaswani. *Attention Is All You Need*. https://arxiv.org/abs/1706.03762. Accessed 26-April-2024. 2017.

# Appendix A

# Development Environment and Libraries

## A.1 Development Environment

The development of the models discussed in this thesis was done through a Jupyter environment hosted by the Univeristy of Agder's servers. This platform provided us with more powerful computational resources than we otherwise had available, including NVIDIA A100 and NVIDIA V100 GPUs, which were used for the training of the machine learning models. Choosing this environment for development allowed for high-performance computation, supporting our iterative approach to development and letting us test many different configurations in a shorter timespan.

## A.2 Libraries Used

### A.2.1 Preprocessing Libraries

The preprocessing was done using these libraries:

- `numpy`: Used for mathematical operations including the FFT implementation in the function `numpy.fft.fft`.
- `scipy`: For saving and loading MATLAB files used for storing spectrogram data.
- `matplotlib`: Used for plotting spectrograms.
- `os`, `datetime`, `tqdm`: For file operations, formatting timestamps, and visualizing loop progress, respectively.

### A.2.2 Tsetlin Machine Libraries

For the development and testing of the Tsetlin Machine models, the following Python libraries were notably used:

- `pyTsetlinMachine.tm`: This library contains the implementation of `MultiClassTsetlinMachine`, which was used for the TM model experiments.
- `PyTsetlinMachineCUDA.tm`: Utilized for employing the GPUs capabilities to make the training process of the Tsetlin Machine models a lot faster.
- `numpy`, `scipy`, `matplotlib`: These libraries were used for mathematical operations, data handling and visualizations, respectively.
- `sklearn.metrics`: Functions such as `accuracy_score` and `f1_score` from this library were used to evaluate model performance.

- `tqdm`, `os`, `pandas`: Used for visualising the progress of loops, handling file operations and data manipulation, respectively.
- `seaborn`: For advanced data visualization, in addition to `matplotlib`.
- `random`, `json`, `pickle`: These libraries were used for random operations, JSON data handling and object serialization for saving objects to files, respectively.

### A.2.3 Transformer Libraries

The development of the Transformer models was done using a different set of Python libraries, mostly from the TensorFlow ecosystem:

- `tensorflow`, `keras`: These libraries contain the core framework and tools for building, training, and validating deep learning models, including the layers and model structures used in the Transformer.
- `tensorflow.keras.optimizers, callbacks`: The Adam optimizer from this module was used for model training and the ReduceLROnPlateau for adjusting learning rate.
- `sklearn.preprocessing`: The `LabelEncoder` and `to_categorical` functions were used for preparing labels suitable for model input.
- `sklearn.metrics`: Used for model evaluation through metrics like `precision_score`, `recall_score`, and others.
- `matplotlib`, `seaborn`: For plotting and visualizing data and results.
- `numpy`, `scipy`, `os`: Used for mathematical operations, signal processing and system operations, respectively.