西 南 交 通 大 学

本科毕业设计（论文）

Railway switch fault detection based on deep learning neural networks model

年　　级:2021 级
学　　号:2021110007
姓　　名:郝岳
专　　业:计算机科学与技术

指导老师（签字）：
时间：　　年　　月

School of Computer Science
FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Railway switch fault detection based on deep learning neural networks model

Yue Hao

Submitted in accordance with the requirements for the degree of BSc Computer Science

2024/25

XJCO3931 Project

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (09/05/2025)
Link to online code repository	URL	Sent to supervisor and assessor (09/05/2025)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) _____

Summary

This paper mainly studies how to find out if a railway switch machine has a problem by looking at the changes in its electric current during operation. The traditional way of checking these machines is not very good—it takes a lot of time, can be wrong, and often misses small problems. So we came up with an idea to use deep learning. The goal is to let the computer learn from past data and help detect faults more accurately and automatically.
In this project, we built and compared three models: MLP, BiGRU, and a CNN–LSTM–Attention model. All of them only use current signals as input. Before training, we used some methods to extract features from the signals, including values from both time and frequency domains. Our dataset had ten types of signals, including normal and nine different faults.
We focused on making sure the data was cleaned properly, the models were easy to understand, and the results were fair. We used common evaluation methods like accuracy, precision, recall, F1-score, and confusion matrix. Everything was coded in Python using PyTorch, and we ran the models on a regular computer with a CPU.
In the end, all three models worked well, with accuracy over 98
In short, this project gives a useful and lightweight way to help railway workers check for faults in switch machines. It can also be a good starting point for future research in AI and real-time fault monitoring.

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, **Dr. Keming Wang**, for his patient mentorship, thoughtful guidance, and generous support throughout the course of this project.

Under Dr. Wang's supervision, I gained invaluable insights into machine learning, time-series analysis, and fault detection theory. His technical expertise and encouraging feedback shaped my understanding of both the theoretical and practical aspects of data-driven modeling. The clarity with which he explained key concepts and the depth of discussions we had greatly enhanced my ability to think critically, design experiments rigorously, and interpret results meaningfully.

Throughout the development of this work, Dr. Wang consistently offered constructive suggestions that helped me refine my models, improve my data processing pipeline, and communicate my findings with precision. His mentorship was instrumental not only in the success of this project but also in my overall academic development.

I am deeply thankful for his guidance and support, which have left a lasting impact on my research mindset and professional growth.

Contents

1 Introduction and Background Research **1**

1.1 Introduction . . . 1

1.2 Literature Review . . . 1

1.2.1 Traditional Approaches . . . 1

1.2.2 Deep Learning Paradigm . . . 1

1.2.3 Multi-Domain Feature Fusion . . . 2

1.2.4 Human-in-the-Loop Strategy . . . 2

1.3 External Requirements and Risk Analysis . . . 3

2 Methodology **4**

2.1 Data Pre-processing and Feature Extraction . . . 4

2.1.1 Raw Current Acquisition . . . 4

2.1.2 Signal Cleaning and Windowing . . . 4

2.1.3 Time-Domain Statistic Vector . . . 4

2.1.4 Frequency-Domain Descriptor Vector . . . 5

2.1.5 Feature Scaling . . . 5

2.1.6 Optional Data Augmentation . . . 5

2.1.7 Rationale for Feature Choice . . . 5

2.2 Multilayer Perceptron (MLP) . . . 6

2.2.1 Motivation for Choosing MLP . . . 6

2.2.2 Architecture of MLP . . . 6

2.2.3 Input and Output Representation . . . 7

2.2.4 Mathematical Formulation . . . 7

2.2.5 Hidden Layer Computation . . . 8

2.2.6 Training Strategy . . . 8

2.2.7 Application in Fault Detection . . . 8

2.2.8 Summary . . . 8

2.3 Bidirectional Gated Recurrent Unit (BiGRU) . . . 8

2.3.1 Motivation for Choosing BiGRU . . . 8

2.3.2 Architecture of BiGRU . . . 10

2.3.3 Input and Output Representation . . . 10

2.3.4 Mathematical Formulation . . . 11

2.3.5 GRU Cell Structure . . . 12

2.3.6 Training Strategy . . . 12

2.3.7 Application in Fault Detection . . . 12

2.4 CNN–LSTM–Attention Hybrid Model . . . 13

2.4.1 Motivation for the Hybrid Architecture . . . 13

2.4.2 Model Architecture . . . 13

2.4.3 Mathematical Formulation . . . 14

2.4.4 Training Procedure 14
2.4.5 Application in Fault Diagnosis 15
2.4.6 Summary 16

3 Implementation and Validation **17**
3.1 Fault Type Labels and Visual Interpretation 17
3.2 Data Preprocessing and Conversion 17
3.3 Validation Procedures 22
3.4 Experimental Environment 23
3.4.1 Data Processing and Training Pipeline 23

4 Results, Evaluation and Discussion **25**
4.1 Training Performance Trends 25
4.2 Confusion Matrix Analysis 27
4.3 Per-Class Evaluation Metrics 29
4.4 Conclusion 31
4.5 Ideas for Future Work 31

5 Deployment Considerations and Practical Integration **33**
5.1 System Architecture for Field Deployment 33
5.2 Integration with Existing Maintenance Workflows 34
5.3 Compliance with Safety and Industry Standards 34
5.4 Monitoring, Updates, and System Maintenance 34
5.5 Resource-Aware Deployment and Optimization Strategies 35
5.6 Summary 35

References **36**

Appendices **38**

A Self-appraisal **38**
A.1 Critical self-evaluation 38
A.2 Personal reflection and lessons learned 38
A.3 Legal, Social, Ethical and Professional Issues 39

B External Material **41**

Chapter 1

Introduction and Background Research

1.1 Introduction

Railway switch machines (also called turnout or point machines) help guide trains from one track to another and are very important for the safety of train operations. A recent report said that in 2023, railways around the world ran more than 12 billion kilometers [11]. Field records also show that about 35% of signal failures come from switch machines [1]. Old-fashioned maintenance—based on fixed times and manual checking—can be slow and often misses early problems, especially in bad weather.

Even though switch machines are small, when something goes wrong, the results can be serious—like delays, derailments, or even big service shutdowns. As rail systems become more automated, just doing scheduled checks or using simple threshold rules isn't enough anymore. There is now a clear need for smarter, real-time systems that can find problems quickly and accurately.

This project tries to build a simple and practical fault detection method using only the motor current recorded while the machine is working. Compared to using cameras or vibration sensors, using current signals is easier, cheaper, and doesn't need big changes to the system. By letting the computer learn patterns in current signals, our goal is to find nine types of common problems with at least 95% recall and make decisions in under 5 seconds—targets discussed with the Shanghai Railway Maintenance Bureau. This method aims to solve both the technical problem of fault detection and the practical need to run on simple devices in the field.

1.2 Literature Review

1.2.1 Traditional Approaches

In the past, most studies used rule-based or threshold methods to find faults, such as checking the peak value or how long the signal lasts. These methods are simple but don't work well with noisy data or when the problem is not obvious [9]. Also, because different machines or tracks can have different behaviors, the rules often need to be changed again and again.

Some basic machine learning methods like SVM, k-NN, and decision trees made things a bit better, using features like RMS or wavelet energy. But these methods rely a lot on features made by hand and are sensitive to changes in data, like how it's scaled or sampled [13]. When the system gets bigger, it becomes harder to prepare features for every case. That's why more people are turning to deep learning, which can learn from raw data directly.

1.2.2 Deep Learning Paradigm

With deep learning becoming more popular, one-dimensional CNNs have been used a lot to catch local signal shapes, while RNNs—especially LSTM or GRU—are good for learning

time-based changes [14]. These models work with raw or slightly processed signals, so they save time on feature making.
Combining CNNs and LSTMs usually gives better results. CNNs help remove high-frequency noise and spot short-term changes, while LSTMs can keep track of long-term patterns. Adding attention mechanisms helps the model focus on important parts of the signal, which makes the results both more accurate and easier to explain [7, 16]. This is important in real use, especially when people want to understand how the system makes decisions.
Compared to traditional methods, deep models are more accurate and easier to scale. They can also handle noisy data better and work on different machines with less adjustment. So, deep learning has become a reliable way to do fault detection, not just in railways but in many other fields too.

1.2.3 Multi-Domain Feature Fusion

Another way is to use features from both the time domain and frequency domain. Time-domain features like RMS or kurtosis describe the shape of the signal, while frequency-domain features like spectral centroid or entropy show how the signal behaves in terms of frequency. Putting these together gives a fuller view and can help catch more kinds of faults.
Some methods, like short-time Fourier transform or wavelet packet transform, turn the signal into a spectrogram that CNNs can treat like an image [12]. This makes it easier to spot things like energy jumps or frequency changes that are hard to see otherwise. In our project, we keep using hand-made features to strike a balance between good results and fast performance, especially since the model needs to run on simple edge devices.

1.2.4 Human-in-the-Loop Strategy

Although the system is designed for automatic fault detection, keeping humans in the loop is crucial for reliability, especially in safety-critical applications like railway operations. By combining AI-based decisions with expert oversight, the system becomes more trustworthy and flexible.
In practical deployment, human operators can be involved in two key ways:

- **Result Verification:** For high-risk fault predictions or uncertain classifications, the system can flag outputs for manual confirmation. Maintenance staff can review waveform snapshots and decide whether to take immediate action.
- **Feedback Collection:** After each confirmed fault or false alarm, feedback from technicians can be used to improve future model performance. This information can be added to the training dataset during scheduled model updates.

This collaborative setup ensures that the system does not operate in a black-box manner and provides accountability. Over time, integrating human expertise also improves the system's interpretability and reduces the likelihood of unnoticed misclassifications.

1.3 External Requirements and Risk Analysis

This project is based on a trial agreement with the Shanghai Railway Maintenance Bureau. As part of the deal, the model must reach at least 95% recall. To test this, we collected 2,116 current signals between August 1 and December 1, 2023. We also used Gaussian-based data augmentation to grow the dataset to 2,220 samples.

We looked at several possible risks. (i) **Misclassification:** The system might raise false alarms or miss real problems, especially when it's not clear. (ii) **Data privacy:** Current signals could leak information about train activity or schedules. (iii) **Model drift:** Changes in machines or repairs might slowly change the signals, which could reduce accuracy.

To reduce these problems, we made sure the model stays under 10 MB so it can run on edge devices, and we use encrypted data transfer that follows the 2022 Railway Safety Act. We also retrain the model every month using the latest data from the field [4]. For any uncertain results, we use a confidence threshold to mark them so staff can check. In the future, we plan to try federated learning, so we can update models without sending raw data between different places.

Chapter 2

Methodology

2.1 Data Pre-processing and Feature Extraction

This section explains how the original current signals from switch machines are turned into two types of input: one that keeps time-based features and one that shows frequency-based information. Each sample passed to the model includes a waveform with a fixed length of $N = 200$ and a 16-dimensional feature vector $\mathbf{x} \in R^{16}$.
These formats are chosen to reflect both the general trend and the detailed patterns in the current signal. This way, the model can focus on key parts that relate to faults, while skipping over unhelpful noise or repeated parts. The steps below show how we cleaned, converted, and structured the data for the whole project.

2.1.1 Raw Current Acquisition

Each time the switch runs, the current is collected at $100, \mathrm{Hz}$ over about two seconds, covering four phases: *unlock*, *conversion*, *locking*, and *buffering*. To make things uniform, every signal is cut or padded with zeros to get exactly $N = 200$ samples.
This fixed length helps the neural models, which need inputs of the same size, while still keeping all the key actions in the switching process. Although we don't mark out each phase by hand, the time structure of the signal already includes the full sequence, so the model can learn phase changes from the signal itself.

2.1.2 Signal Cleaning and Windowing

To fix missing values or errors, we use forward interpolation, and very large or very small values are clipped at the 99^{th} percentile. Formally, for a raw signal $\tilde{\mathbf{x}} = \tilde{x}_1, \ldots, \tilde{x}_N$, the cleaned version is:

$$x_i = \min!\big(\max(\tilde{x}_i, p_{0.5}), , p_{99.5}\big), \tag{2.1}$$

where $p_{0.5}$ and $p_{99.5}$ are the 0.5th and 99.5th percentiles from the whole dataset. This kind of clipping is often used in time-series data to stop extreme values from affecting the results [10]. It's important because in real railway work, current readings can be affected by things like electromagnetic noise or unstable voltage. Without this step, big spikes might throw off the model's understanding of what's normal.

2.1.3 Time-Domain Statistic Vector

From each cleaned signal $\mathbf{x} = x_1, \ldots, x_N$, we take ten time-domain features, listed in Table **??**. One of them, short-time energy, is based on a 50-sample sliding window.
These features are picked to show things like average current, sharp spikes, and signal balance. For example, if the kurtosis is high, it might mean something hit or jerked; the rectified mean

tells us how much total current was used. Using a sliding window helps us catch faults that only happen in short parts of the signal.

2.1.4 Frequency-Domain Descriptor Vector

We apply the Fast Fourier Transform (FFT) to move the signal into the frequency domain, giving us a spectrum $X[k]$. From this, we get six features: main frequency f_1, five extra harmonics $f_{2...6}$, spectral centroid C, spectral energy E, and energy difference between signal frames ΔE. These are shown in Table **??** with the time-domain ones.
The spectral centroid tells us where most of the energy is in terms of frequency, which helps detect mechanical wear or electrical issues [15]. Together, these frequency features help spot problems like misalignment (harmonics), unstable current (energy changes), or slow wear over time (high ΔE).

2.1.5 Feature Scaling

To prepare the features for model training, we scale all values into the range $[-1, 1]$ using this formula:

$$x_i^{(\text{scaled})} = 2\frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)} - 1, \tag{2.2}$$

This method, called min-max normalization, helps make sure no one feature dominates the others, especially when we use ReLU [5].
It's very useful when we mix features with different scales or units. Without scaling, features with bigger values could take over and reduce the model's ability to learn from the smaller ones.

2.1.6 Optional Data Augmentation

To help with imbalanced classes, we generate more samples by adding random Gaussian noise $\mathcal{N}(0, \sigma^2_{\text{aug}})$ and scaling the signal by a factor $f_{\text{scale}} \in [0.9, 1.1]$. Since this is done after clipping (Equation (2.1)), it doesn't ruin the original signal structure.
This kind of augmentation gives us more training samples for rare faults and adds useful randomness to help the model generalize. In our experiments, it especially helped improve recall for the rare and serious fault types.

2.1.7 Rationale for Feature Choice

Time-domain features focus on how strong or jumpy the signal is, while frequency-based ones catch repeated or changing signal patterns. Using both gives us a clearer picture. For instance, more low-frequency energy might mean resistance, high crest factor could mean impacts, and shifting spectral centroid may point to aging parts.
This method is in line with current research in fault detection, and gives the model rich and useful inputs [3].
Also, by using short feature vectors instead of whole signals, we keep the model small and fast—perfect for use on simple edge devices. It also makes the model's output easier to explain, which helps engineers and technicians understand or fix real problems.

2.2 Multilayer Perceptron (MLP)

2.2.1 Motivation for Choosing MLP

In this project, we use the Multilayer Perceptron (MLP) to classify current signals from switch machines. The MLP is a simple and efficient model that works well for this kind of task. It helps us find the relationship between the input features and the fault types, even if the relationship is not linear.
Since our inputs are fixed-length feature vectors without any time order, using an MLP makes sense. It trains quickly, is easy to understand, and doesn't require much computing power [2]. Compared to models like RNNs, MLPs are easier to train because they don't depend on time steps or sequences. This makes them suitable for small devices or embedded systems used in real-world fault detection.

2.2.2 Architecture of MLP

An MLP is made up of an input layer, some hidden layers with activation functions, and an output layer. Figure 2.1 shows the general structure.

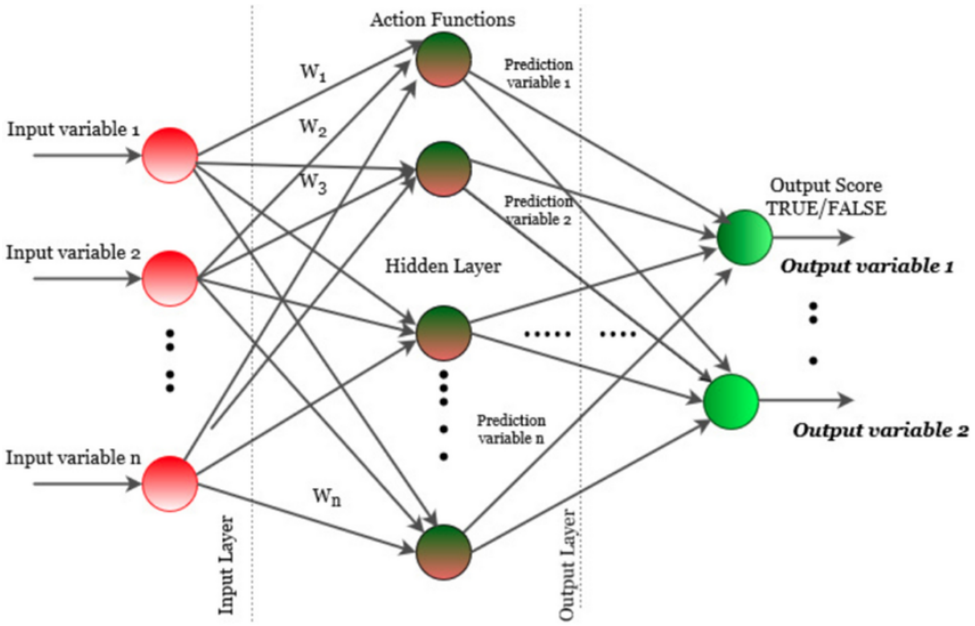

Figure 2.1: Basic architecture of a Multilayer Perceptron, showing input, hidden, and output layers.

Our model is built as follows:

- The input layer has 16 neurons, one for each time–frequency feature.
- There are two hidden layers with 64 and 32 neurons. Both use the ReLU function.
- The output layer has 10 neurons, one for each fault class, followed by a softmax function.

This design keeps the model small but effective. The number of neurons in the hidden layers was chosen through testing. ReLU is used because it adds non-linearity and helps the model train better.

2.2.3 Input and Output Representation

Figure 2.2 shows how data moves through the model. A 16-dimensional input goes through the network, and the output is a set of probabilities for all fault types.

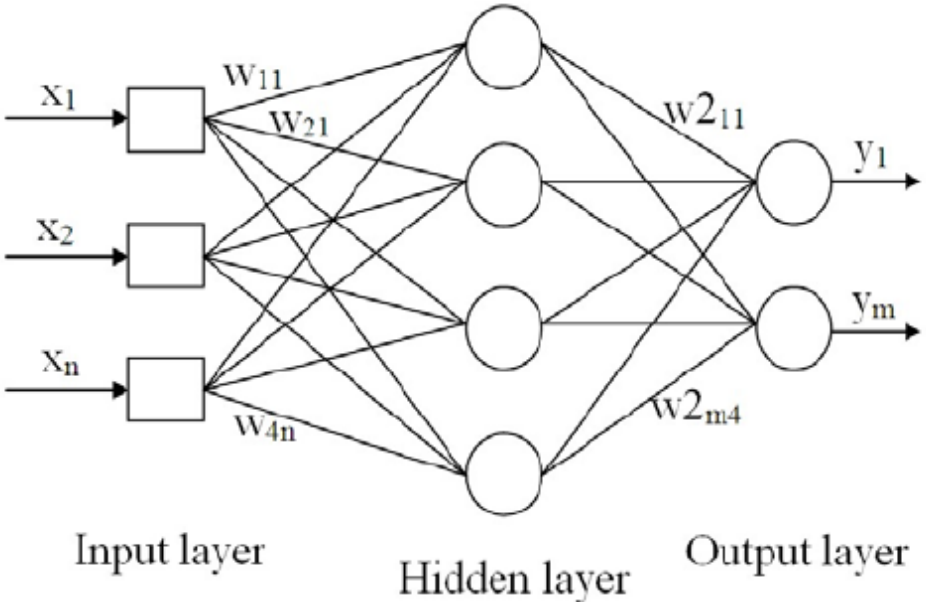

Figure 2.2: Schematic view of feature input and output probability distribution in MLP.

The softmax output shows how confident the model is for each class. This is useful if we want to set a threshold for uncertain results and ask for manual checks.

2.2.4 Mathematical Formulation

The MLP calculates its outputs step by step using these formulas:

$$\mathbf{h}^{(1)} = \text{ReLU}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}), \tag{2.3}$$

$$\mathbf{h}^{(2)} = \text{ReLU}(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}), \tag{2.4}$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}), \tag{2.5}$$

Here, $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ are the weights and biases for each layer.
The training goal is to reduce the cross-entropy loss:

$$\mathcal{L} = -\sum_{j=1}^{C} y_j \log(\hat{y}_j), \tag{2.6}$$

where $C = 10$ is the number of classes, y_j is the true label (one-hot), and $\hat{y}_j$ is the predicted value.
This loss function works well for multi-class classification and pushes the model to make accurate predictions.

2.2.5 Hidden Layer Computation

Each hidden layer uses the ReLU function, which helps with gradient flow and speeds up training. Figure 2.3 shows how each neuron works in a hidden layer.

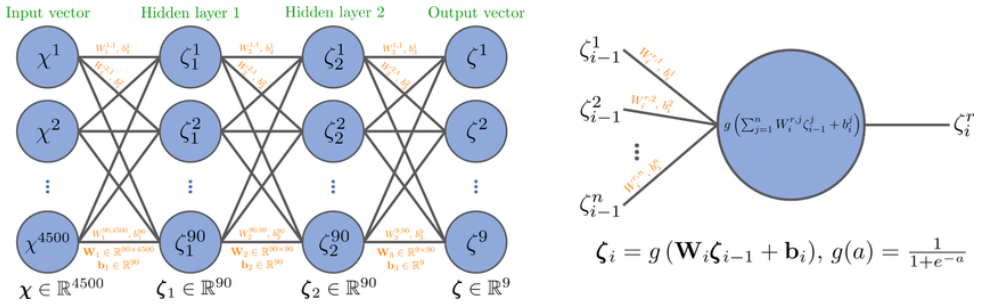

Figure 2.3: Zoomed-in view of MLP hidden layer computation using activation units.

ReLU is very simple—it keeps positive values and turns negatives into zero. This helps the model train faster and makes the output more stable.

2.2.6 Training Strategy

We use the Adam optimizer [6] with a learning rate of 0.001. Training is done in batches of 32 for up to 100 epochs. We use early stopping based on the validation loss.
Adam adjusts learning rates automatically, which helps when the data is uneven. Early stopping prevents overfitting, which is helpful when data is small or unbalanced.

2.2.7 Application in Fault Detection

Figure 2.4 shows how the MLP is used in real-world fault detection.
At run-time, the model takes in a 16-feature vector and gives probabilities for the ten fault types. Because the model is small, it can run on simple devices like microcontrollers. It's also easier to debug and explain, which is important in safety-critical systems.

2.2.8 Summary

Even though MLPs can't handle time sequences, they work well for structured data like this. They are easy to build, fast to run, and don't need much memory.
Their simple design also makes them good for use in safety systems. While other models might give slightly better results, MLPs are a strong and reliable choice for classifying switch faults.

2.3 Bidirectional Gated Recurrent Unit (BiGRU)

2.3.1 Motivation for Choosing BiGRU

To handle current signals that change over time, the Bidirectional Gated Recurrent Unit (BiGRU) offers a reliable way to learn from both past and future data points in a sequence. Unlike regular models that only read signals forward, BiGRU looks at the whole picture—both directions—making it easier to spot patterns that depend on what happens before and after a point in time [**?**].

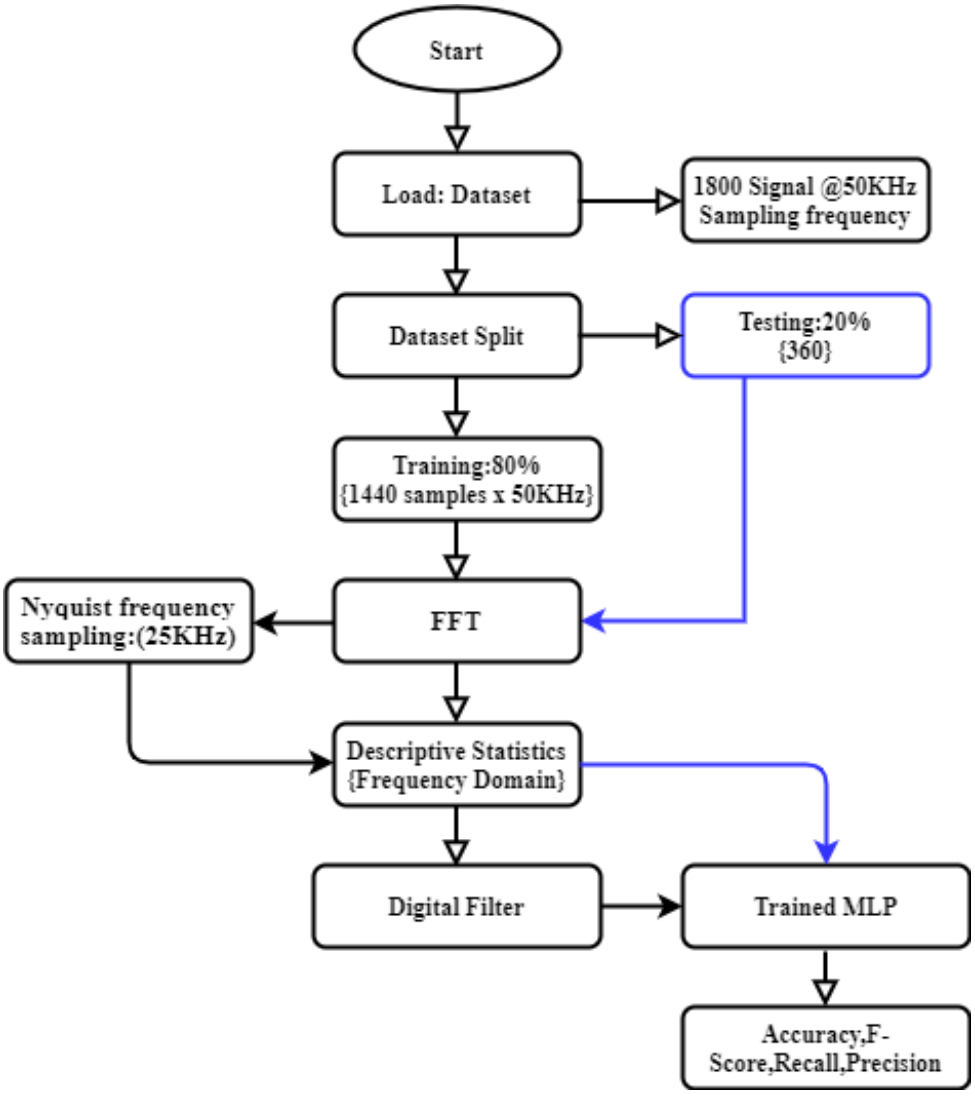

Figure 2.4: MLP used for fault classification in an electromechanical diagnosis scenario.

This ability is especially useful when diagnosing faults, since some issues can only be recognized by looking at earlier and later signal behaviors. For example, if the locking process is slower than usual at the end of a cycle, BiGRU can compare it to the unlocking part at the beginning, something a one-way model might miss.

2.3.2 Architecture of BiGRU

The BiGRU consists of two GRU layers—one reads the sequence from start to end, and the other reads it in reverse. Their outputs are combined at each step to make predictions. This setup is shown in Figure 2.5.

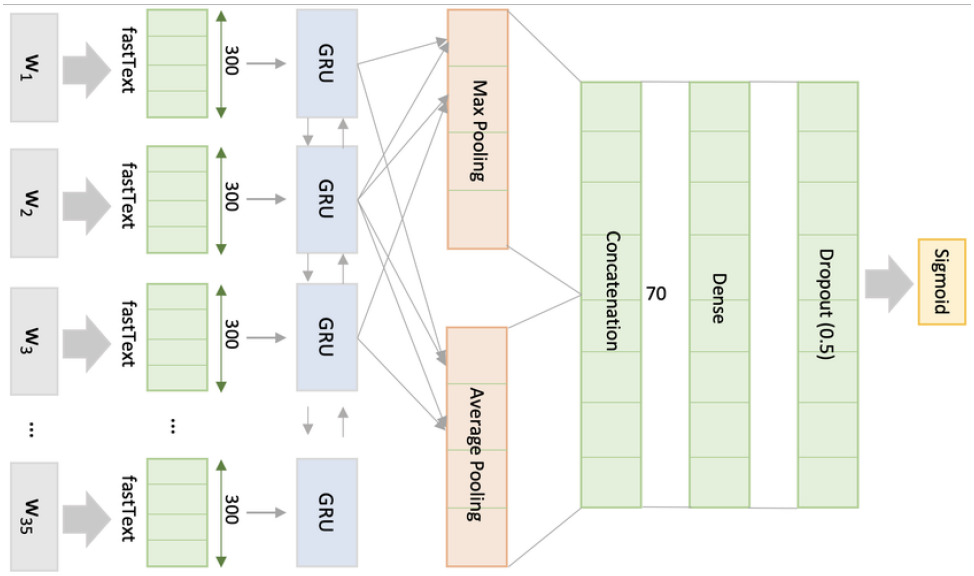

Figure 2.5: Basic architecture of a Bidirectional GRU, showing forward and backward processing layers.

In our setup:

- The input is a time series with 200 steps, each containing one feature.
- A BiGRU layer with 64 units handles the sequence in both directions.
- The output is passed through a dense layer with 10 neurons (one for each fault), using softmax to get probabilities.

This design provides a good trade-off between capturing useful features and keeping the model lightweight enough for practical use.

2.3.3 Input and Output Representation

As shown in Figure 2.6, the input sequence flows through the bidirectional GRU, and the model returns a probability for each fault class.
This keeps the original time-order information intact and allows the model to detect unusual shapes in the signal, such as sudden drops or flat segments—signs of things like hardware jams or wiring problems.

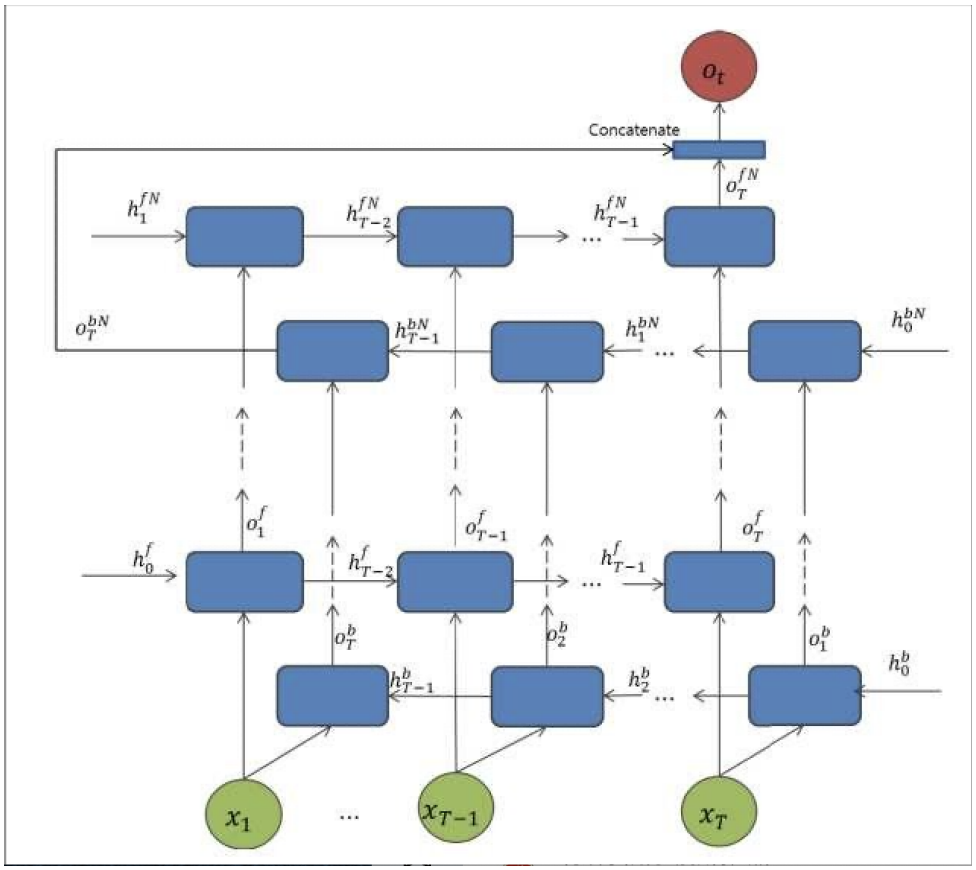

Figure 2.6: Schematic view of time-series input and output probability distribution in BiGRU.

2.3.4 Mathematical Formulation

At time step t, the forward and backward GRUs are calculated as:

$$\overrightarrow{h}_t = \mathrm{GRU}(x_t, \overrightarrow{h}_{t-1}) \tag{2.7}$$

$$\overleftarrow{h}_t = \mathrm{GRU}(x_t, \overleftarrow{h}_{t+1}) \tag{2.8}$$

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t] \tag{2.9}$$

Here, x_t is the signal value at time t, and h_t is made by joining the forward and backward hidden states.

To train the model, we use categorical cross-entropy as the loss function:

$$\mathcal{L} = -\sum_{j=1}^{C} y_j \log(\hat{y}_j), \tag{2.10}$$

where $C = 10$ is the number of fault types, y_j is the actual class (in one-hot form), and $\hat{y}_j$ is the predicted probability.

This setup helps the model learn from the whole sequence and makes sure it gets the classification right across different kinds of signals.

2.3.5 GRU Cell Structure

A GRU cell uses two key gates: the update gate and the reset gate, as shown in Figure 2.7.

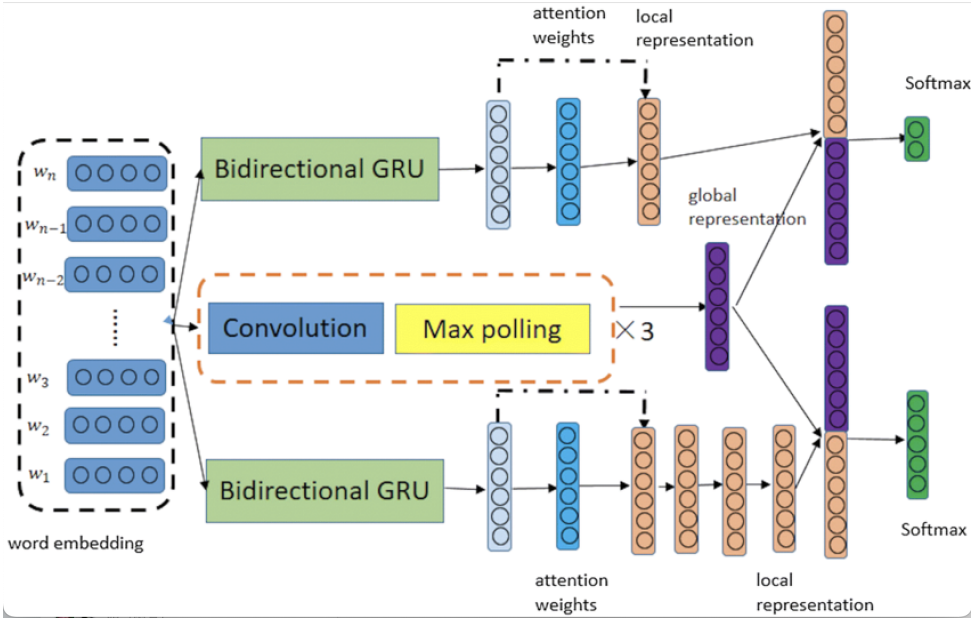

Figure 2.7: Internal structure of a GRU cell, showing update and reset gates.

The update gate decides how much of the past information to keep, and the reset gate controls what to forget. This helps the model focus on the important parts of the signal, especially over longer sequences, and reduces the risk of forgetting key events.
Compared to LSTMs, GRUs are simpler and faster, which makes them more suitable for real-time fault detection on devices with limited processing power.

2.3.6 Training Strategy

We train the BiGRU model using the Adam optimizer [6], starting with a learning rate of 0.001. Early stopping is used to stop training if the validation performance stops improving. The batch size is set to 32, and training can run for up to 100 epochs.
Since time-series models can be sensitive to their initial settings and data order, Adam's adaptive updates and early stopping help stabilize training and avoid overfitting.

2.3.7 Application in Fault Detection

Figure 2.8 shows how BiGRU is applied in real scenarios for diagnosing faults in railway switch machines.
This model works particularly well when two faults look similar in the frequency domain but behave differently over time. For example, if the current suddenly drops near the end of the cycle, BiGRU can detect this by comparing it to what happened earlier. Since it looks at the whole signal sequence, it provides a deeper understanding of how faults develop, which is critical for catching early-stage issues.

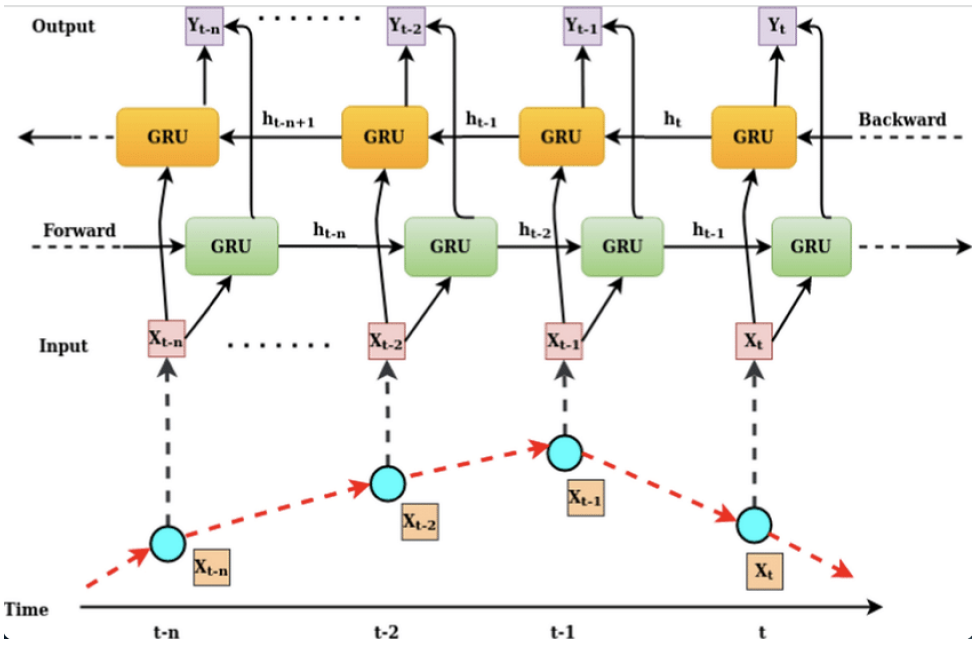

Figure 2.8: BiGRU used for fault classification in an electromechanical diagnosis scenario.

2.4 CNN–LSTM–Attention Hybrid Model

2.4.1 Motivation for the Hybrid Architecture

In time-series tasks like fault detection, it's important to catch both short spikes and long-term trends in the data. This is why we decided to combine three types of layers—CNN, LSTM, and attention—into one model. Each part plays a different role: CNN helps find small patterns quickly, LSTM tracks how the signal changes over time, and attention tells the model which parts of the signal to focus on [17].
This design helps with common problems in current signals, such as:

1. noisy fluctuations,
2. changes in timing between signal phases,
3. and small differences that point to different fault types.

2.4.2 Model Architecture

The hybrid model is made up of these key components:

- **CNN Layers**: These extract local patterns and help reduce noise.
- **LSTM Layers**: These follow how patterns unfold over time.
- **Attention Mechanism**: This lets the model highlight the most useful parts of the signal.
- **Fully Connected Layer**: This final layer maps the attention results to the fault categories.

Figure 2.9 shows what the full model looks like.

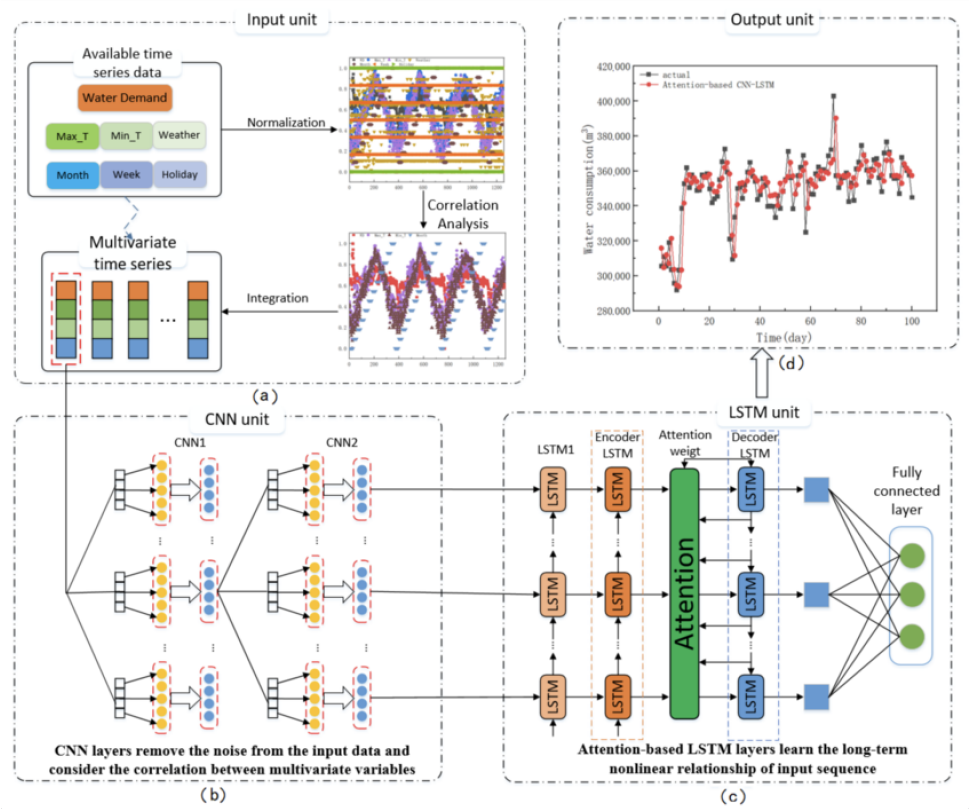

Figure 2.9: Overall architecture of the CNN–LSTM–Attention model.

2.4.3 Mathematical Formulation

Let's say the input signal is $X = [x_1, x_2, ..., x_T]$, where each x_t is a vector of signal features.

CNN Layer:

$$c_t = \text{ReLU}(W_c * x_{t:t+k-1} + b_c) \tag{2.11}$$

LSTM Layer:

$$h_t = \text{LSTM}(c_t, h_{t-1}) \tag{2.12}$$

Attention Mechanism:

$$\alpha_t = \frac{\exp(W_a h_t)}{\sum_{i=1}^{T} \exp(W_a h_i)} \tag{2.13}$$

$$c = \sum_{t=1}^{T} \alpha_t h_t \tag{2.14}$$

Output Layer:

$$\hat{y} = \text{softmax}(W_o c + b_o) \tag{2.15}$$

2.4.4 Training Procedure

We use categorical cross-entropy to train the model:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{2.16}$$

Here, C is the total number of fault types. We train with the Adam optimizer [6], using a learning rate of 0.001. To avoid overfitting, we apply dropout (rate = 0.3) after the LSTM layer and use early stopping based on how the model performs on the validation set.

2.4.5 Application in Fault Diagnosis

We feed the model with 1D current signals, each 200 points long. It can recognize patterns linked to faults like contact wear, stalling, or incomplete motion. Figure 2.10 shows a real-world use case.

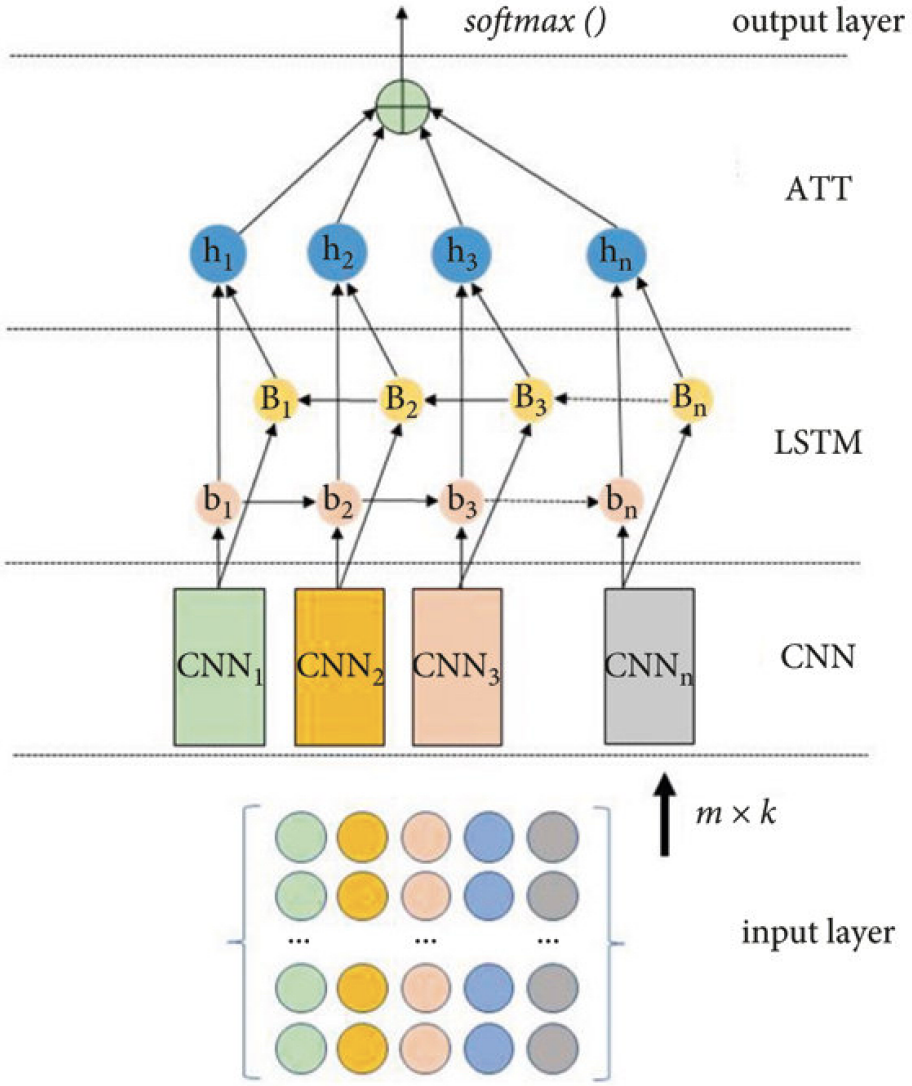

Figure 2.10: Application of the CNN–LSTM–Attention model in a fault diagnosis task.

Compared to using CNN or LSTM alone, this model does a better job at handling noisy data and imbalanced fault categories. Plus, the attention scores show which parts of the signal mattered most, which is helpful when explaining the results or debugging.

2.4.6 Summary

By mixing CNNs, LSTMs, and attention, this model can catch both quick changes and longer patterns in signals. It's flexible enough to work with other types of time-series data too. Thanks to the attention layer, it's easier to see why the model made a certain prediction, which is important when used in safety-critical systems like railway monitoring.

Chapter 3

Implementation and Validation

3.1 Fault Type Labels and Visual Interpretation

The dataset used in this project includes ten class labels, ranging from 0 to 9. Label 0 indicates normal operation, while Labels 1 through 9 represent different fault types in switch machine current signals. Each class has a distinct waveform shape, some showing minor issues and others revealing clear signs of malfunction.
Figures 3.1 to 3.10 provide sample waveforms and a short description of what each signal might suggest in terms of fault behavior.

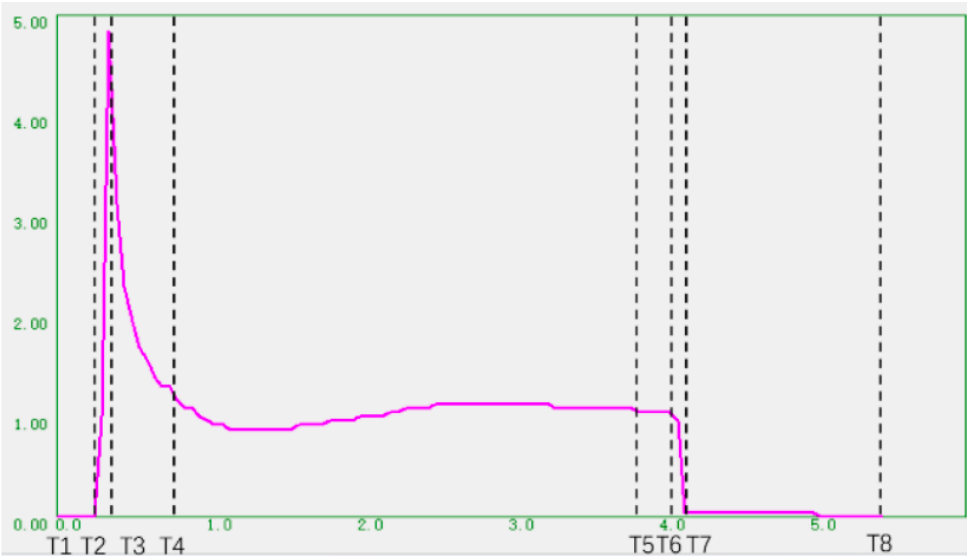

Figure 3.1: Label 0 (Normal): Smooth current waveform with clear segmentation between startup, unlocking, conversion, and locking phases.

3.2 Data Preprocessing and Conversion

The preprocessing pipeline is built in Python and designed to extract both time-domain and frequency-domain features from the raw current waveforms. While the theory behind these features has been explained in Chapter 2, this section outlines the steps actually used in implementation.
Each waveform, with 200 data points, is converted into a 16-dimensional feature vector. Time-domain features are calculated using NumPy, and frequency components are extracted using the Fast Fourier Transform from `scipy.fft.fft`. The strongest frequency peaks are selected based on amplitude.
We normalize all features using `MinMaxScaler` from `scikit-learn`, which is fit only on the training set to avoid data leakage. These processed vectors are directly used as input for MLP training. For BiGRU and CNN–LSTM–Attention models, the vectors are combined with the

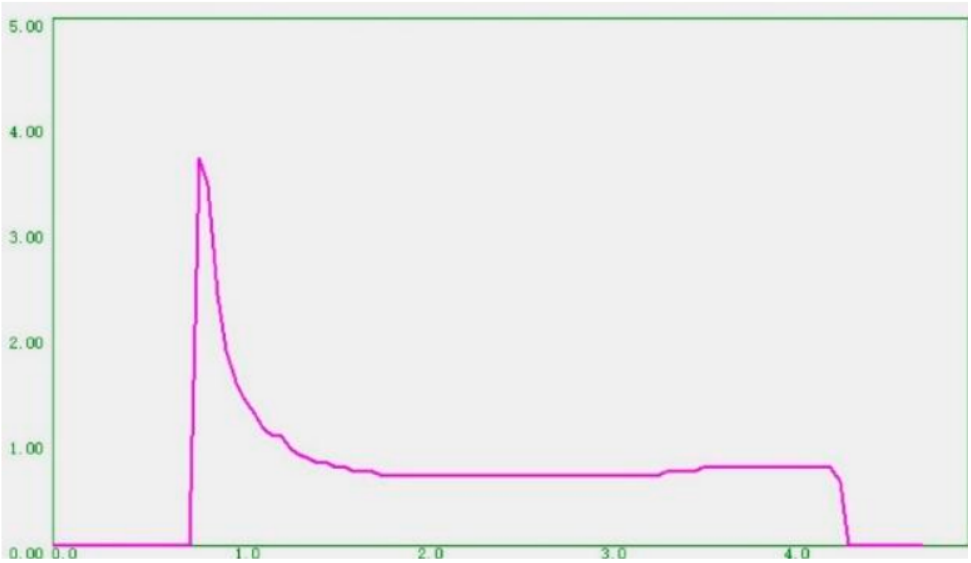

Figure 3.2: Label 1: Flat start indicating a delay during startup, possibly due to relay contact wear or oxidation.

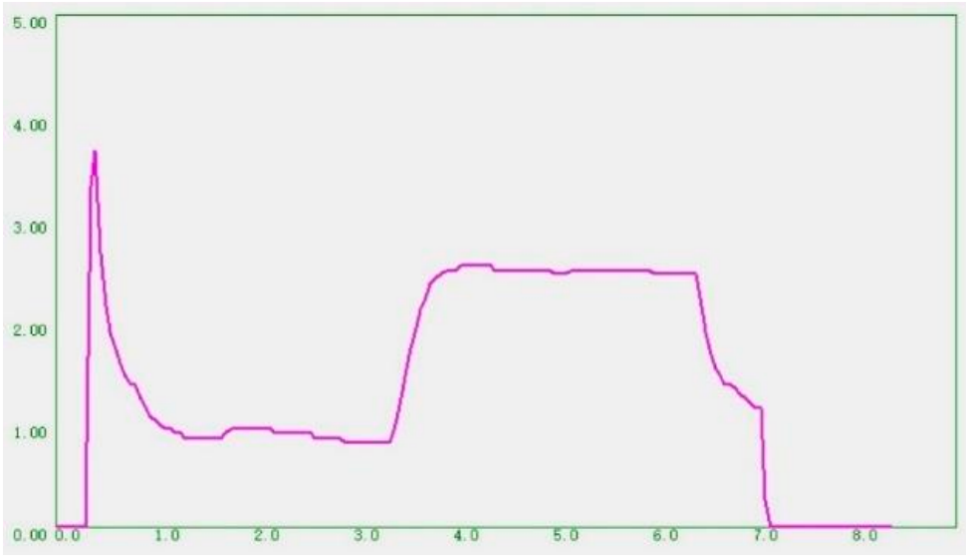

Figure 3.3: Label 2: Prolonged high current during the locking stage, likely caused by mechanical jamming.

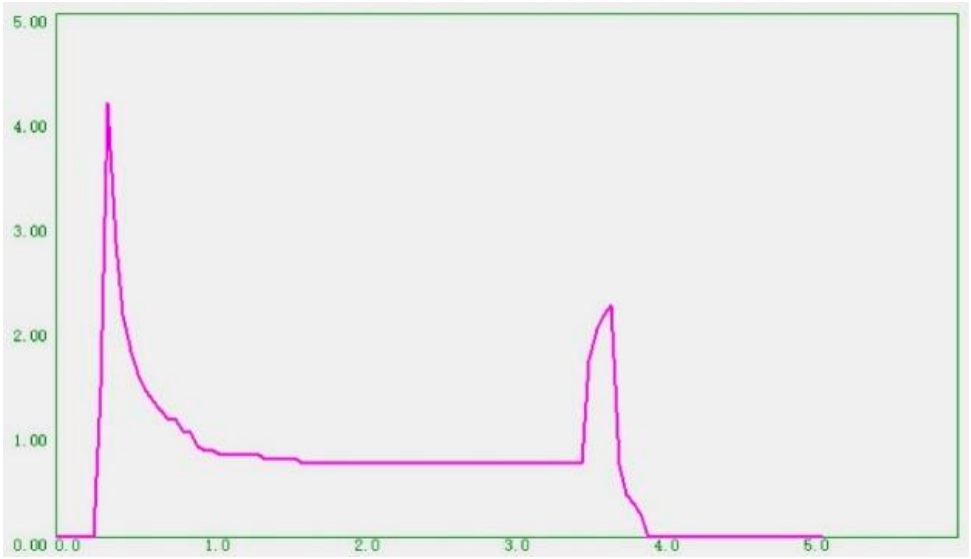

Figure 3.4: Label 3: Sharp current spike during locking, often triggered by sudden resistance or misalignment.

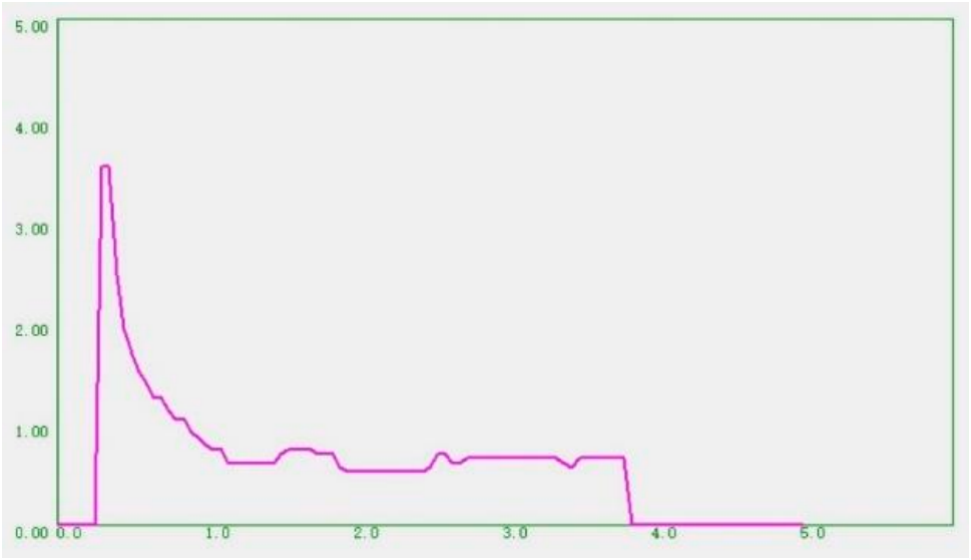

Figure 3.5: Label 4: Irregular oscillations in the conversion phase, possibly related to loose parts or debris.

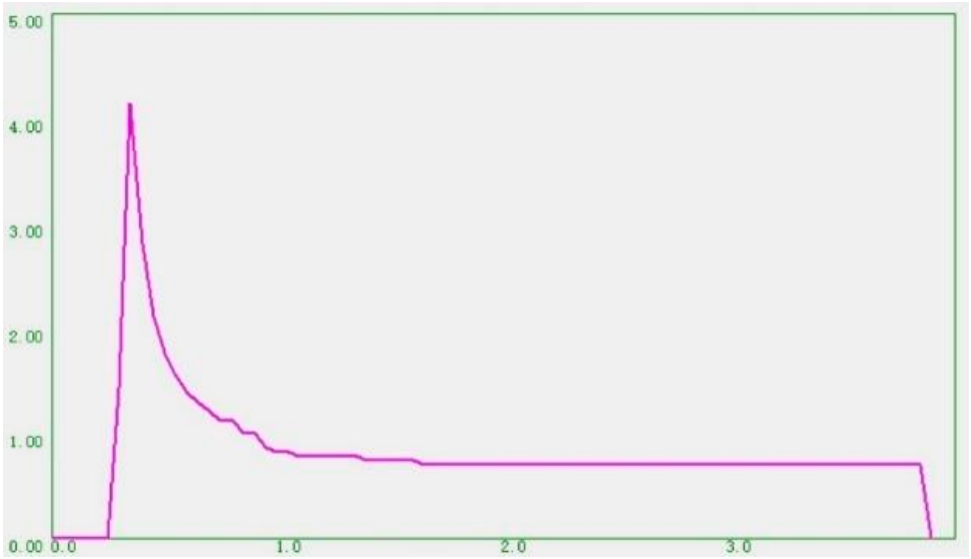

Figure 3.6: Label 5: A sudden drop after a brief plateau during conversion, suggesting an abrupt blockage.

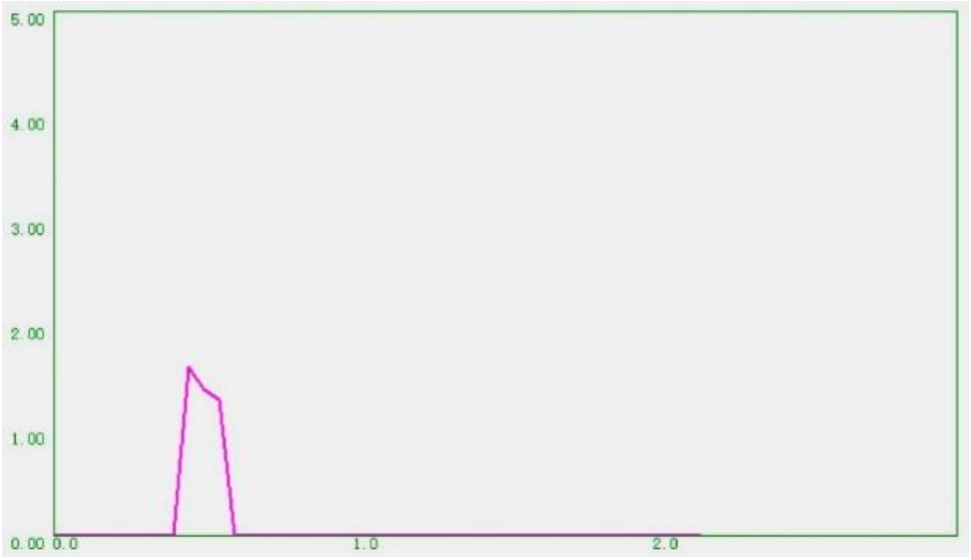

Figure 3.7: Label 6: Truncated waveform, often seen in cases of motor stall, fuse blowout, or emergency cutoff.

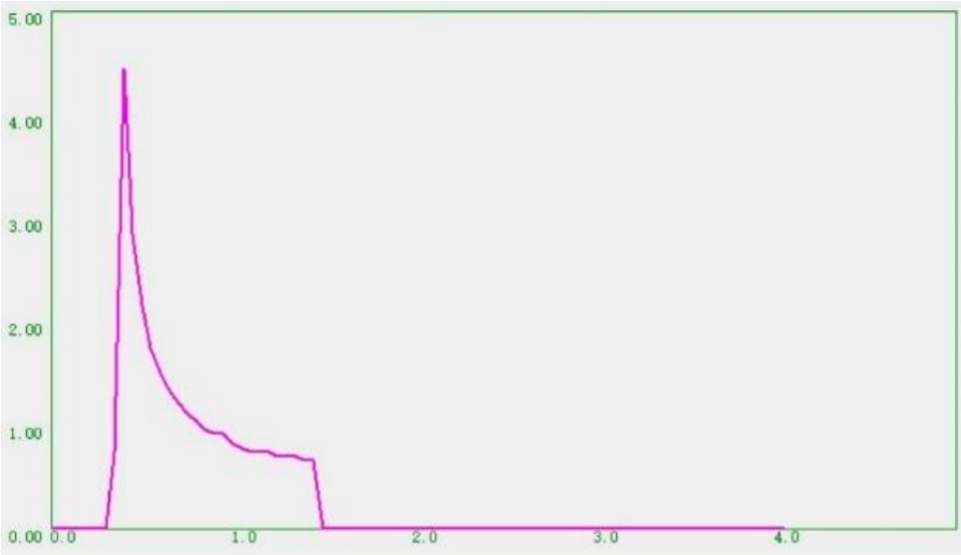

Figure 3.8: Label 7: Sudden interruption, typically due to wiring failure or unstable relay contact.

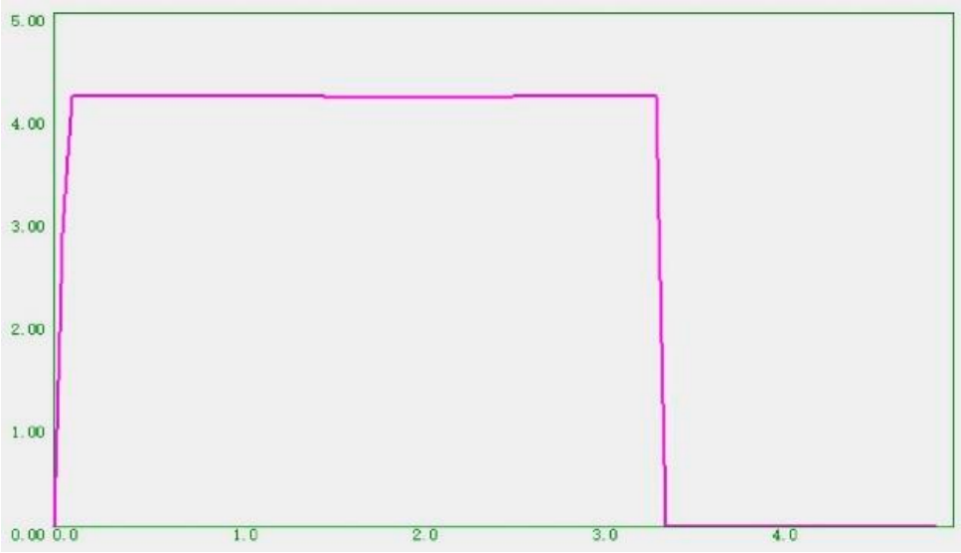

Figure 3.9: Label 8: Persistently high peak current, often caused by a jammed rotor or a short circuit.

Figure 3.10: Label 9: Oversized surge at startup, possibly due to delayed switching or welded relay contacts.

original time sequences to support both temporal and statistical learning.

3.3 Validation Procedures

A consistent validation workflow is adopted across all models to ensure fair comparison and reproducibility.

Dataset Splitting

We divide the dataset into 80% training and 20% validation using a fixed random seed via `train_test_split` from `scikit-learn`. Cross-validation is not used, and no separate test set is included.

Evaluation Metrics

Model performance is measured on the validation set using the following metrics:

- **Accuracy**: The percentage of correctly classified samples overall.
- **Precision**: Macro-averaged precision across all ten classes.
- **Recall**: Macro-averaged recall, showing the model's ability to capture each class.
- **F1-Score**: The harmonic mean of macro precision and recall.
- **Confusion Matrix**: A 10×10 matrix displaying class-level prediction results.

Visualization Outputs

For model comparison and interpretation, the following plots are generated:

- **Confusion Matrices**: To reveal common misclassifications between classes.

- **Class-Wise Bar Charts**: Precision, recall, and F1-score plotted per class.
- **Training Curves**: Accuracy and loss over epochs for training and validation sets.

All plots are created with `matplotlib` and saved automatically. Their analysis is provided in Chapter 4.

Comparison Strategy

The MLP, BiGRU, and CNN–LSTM–Attention models are all trained and validated on the same split using identical settings. We do not use external benchmarks; instead, we focus on direct comparison under the same experimental conditions. The next chapter presents the results and discusses how each model performs under practical constraints.

3.4 Experimental Environment

All experiments were conducted on a local machine with the configuration shown in Table 3.1.

Table 3.1: Experimental hardware and software setup.

Component	**Specification**
Operating System	macOS (MacBook Air M1, 2020)
Processor	Apple M1 chip
Programming Language	Python 3.10
Frameworks	PyTorch 2.x, NumPy, SciPy
Libraries	Scikit-learn, Pandas, Matplotlib
Compute Mode	CPU only (no GPU used)

3.4.1 Data Processing and Training Pipeline

Figure 3.11 illustrates the full pipeline, from waveform input to model evaluation. Each step—signal processing, feature extraction, normalization, training, and validation—is implemented as a separate Python module.

This modular setup allows easy swapping of model architectures without altering preprocessing steps. For example, the same processed inputs can be used across all models. Intermediate outputs such as cleaned signals and scaled features can be saved for debugging, offline training, or interpretation via tools like SHAP or Grad-CAM.

The pipeline is designed for batch execution and can be adapted for real-time fault detection with minimal changes. By keeping all stages independent and scriptable, the workflow supports easy updates, reproducibility, and scalability.

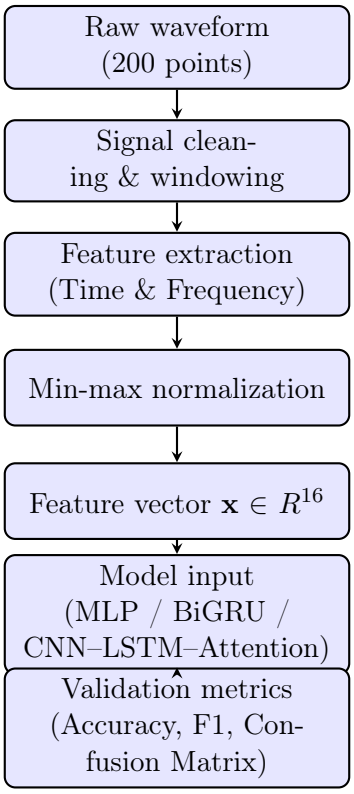

Figure 3.11: End-to-end pipeline for data processing and model evaluation.

Chapter 4

Results, Evaluation and Discussion

4.1 Training Performance Trends

To understand how each model learns during training, we tracked accuracy, precision, recall, and F1-score over all training epochs. We looked at both training and validation sets to see if the models were learning properly or starting to overfit. The learning curves for each model are shown in Figures 4.1, 4.2, and 4.3.

Multilayer Perceptron (MLP)

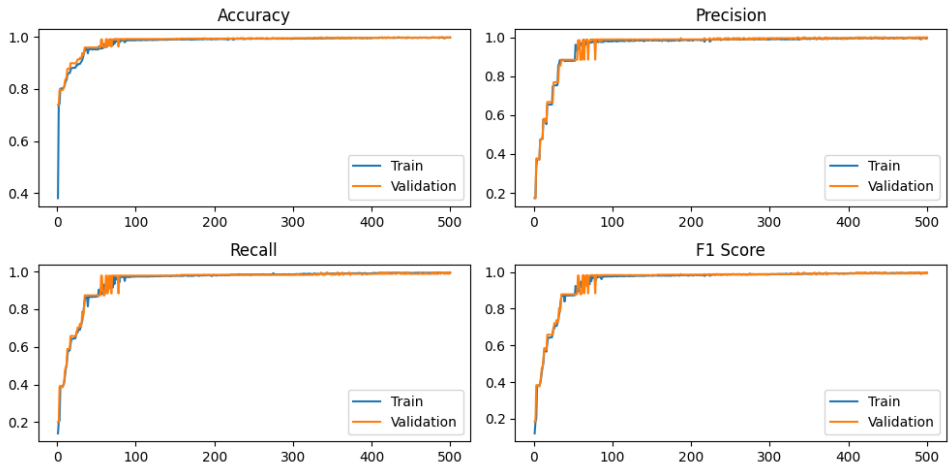

Figure 4.1: Training and validation performance curves of the MLP model over 500 epochs.

The MLP model reached good accuracy quickly, mostly within the first 50 rounds of training, and then stayed stable. The training and validation lines stayed close, which means it was not overfitting. Even though MLP is a simple model, it works well here because the input features were already well-prepared.

Bidirectional GRU (BiGRU)

BiGRU also ended up with high accuracy. It was a bit more unstable at the beginning of training, with ups and downs in the curves, but later it smoothed out and stayed above 98%. Because this model can look at both directions in a sequence, it seems to catch some useful patterns in the signal over time.

CNN–LSTM–Attention

This model learned fast and did a good job. The validation curves were very smooth, which shows that the model didn't just memorize the training data but learned the actual patterns.

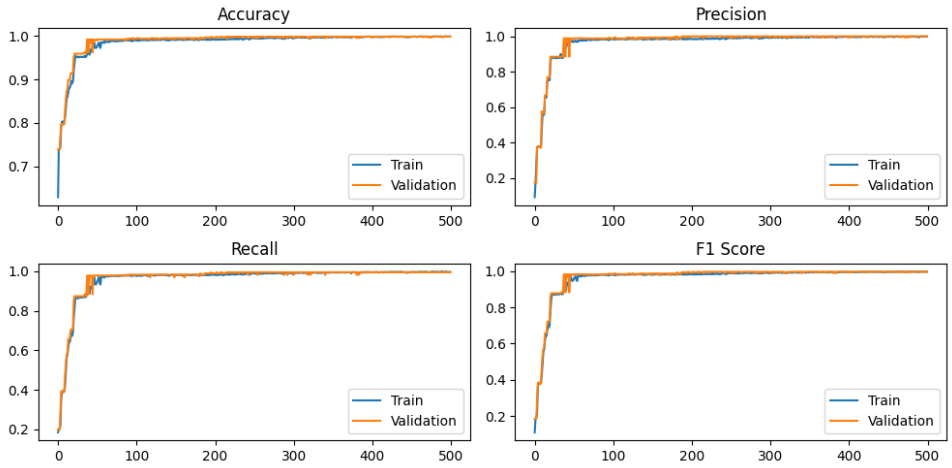

Figure 4.2: Training and validation performance curves of the BiGRU model over 500 epochs.

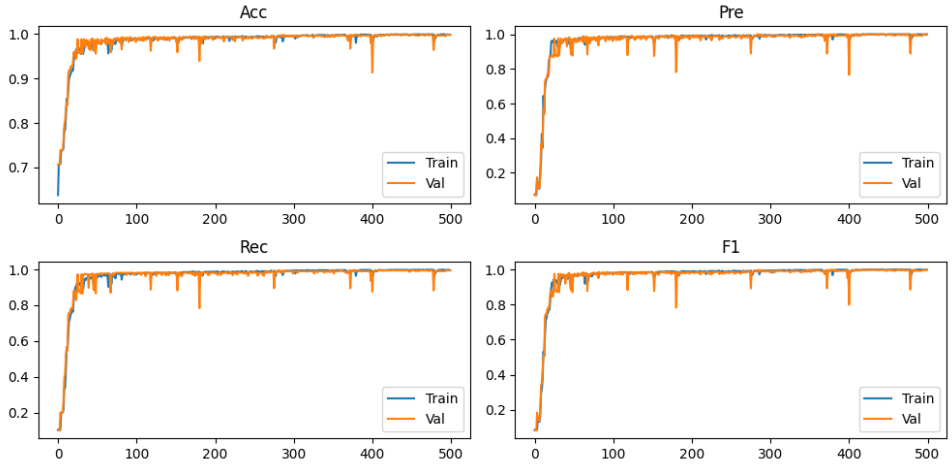

Figure 4.3: Training and validation performance curves of the CNN–LSTM–Attention model over 500 epochs.

The attention layer likely helped it focus on the important parts of the current signal, which made the model more confident in its decisions.

4.2 Confusion Matrix Analysis

Confusion matrices help us see which classes the models confused with others. They're good for spotting which fault types are harder to tell apart.

MLP

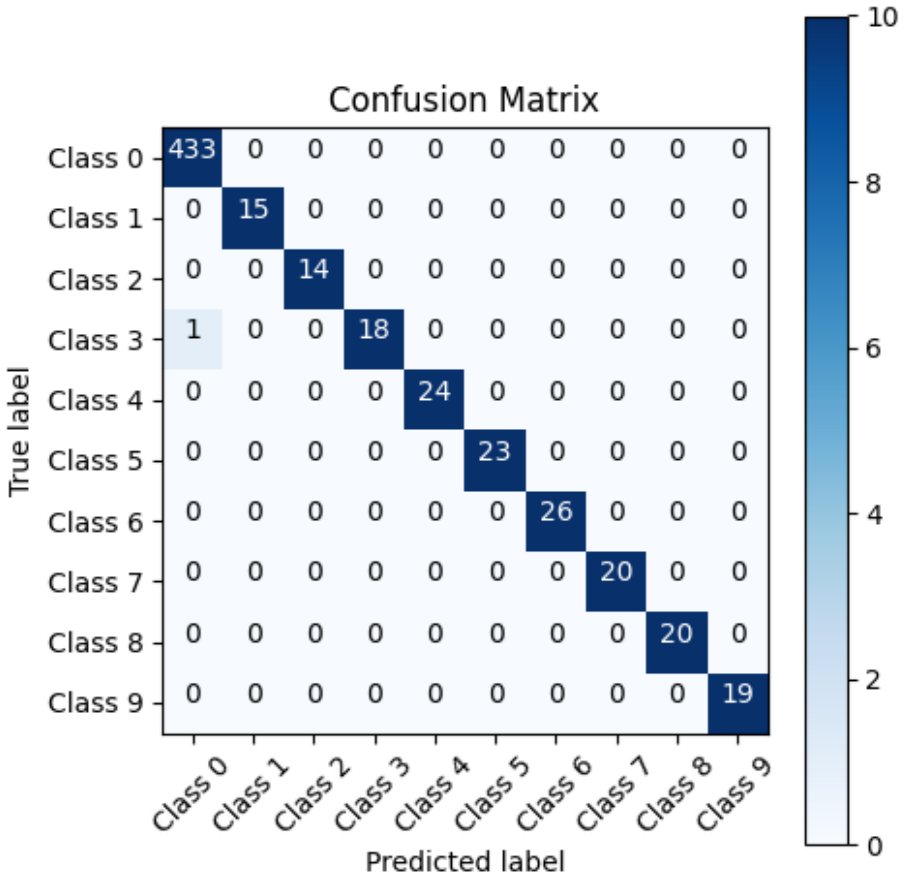

Figure 4.4: Confusion matrix of the MLP classifier.

MLP did well overall. The only small problem was that it sometimes confused Class 0 with Class 3. That's likely because their waveforms look a bit similar, especially near the locking phase. But overall, MLP gave us decent results for a basic model.

BiGRU

The BiGRU matrix shows very few wrong guesses, and those mistakes are spread out across different classes. That means the model didn't have a strong weakness for any particular class and handled them all quite evenly. This may be because it captures time-based changes in the signal well.

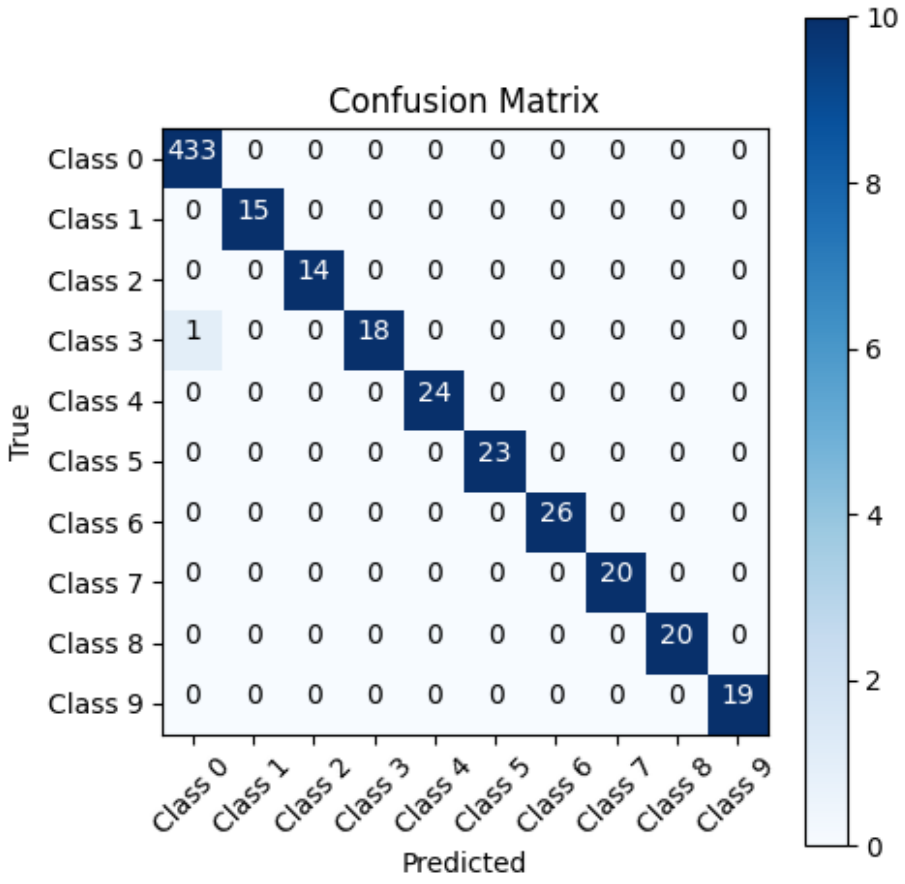

Figure 4.5: Confusion matrix of the BiGRU classifier.

CNN–LSTM–Attention

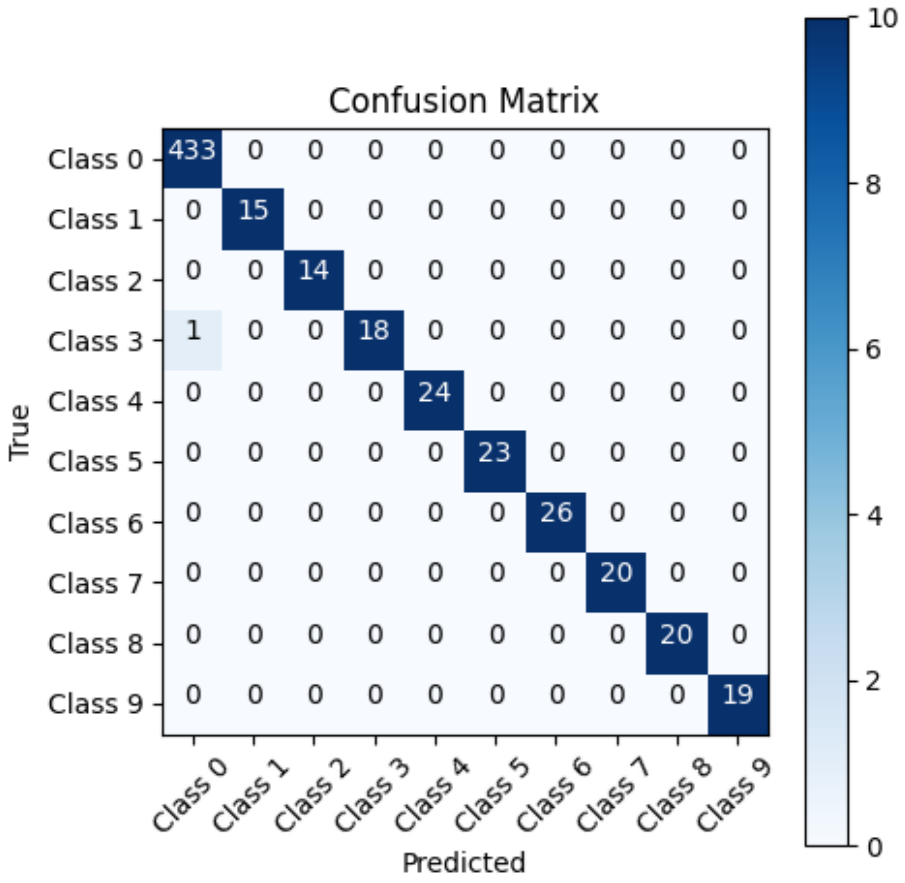

Figure 4.6: Confusion matrix of the CNN–LSTM–Attention classifier.

This model almost never made mistakes. The confusion matrix is very clean, meaning each class was predicted correctly. It probably benefits from combining spatial, time-based, and attention features, which gives it more information to work with.

4.3 Per-Class Evaluation Metrics

To see how well each model handled each class, we looked at the precision, recall, and F1-score for all 10 classes. The bar charts make it easier to compare their strengths and weaknesses.

MLP

MLP did quite well across the board. One class (Class 3) had slightly lower recall, which means the model sometimes failed to catch it. But the gap was small, and overall, the model treated all classes fairly well.

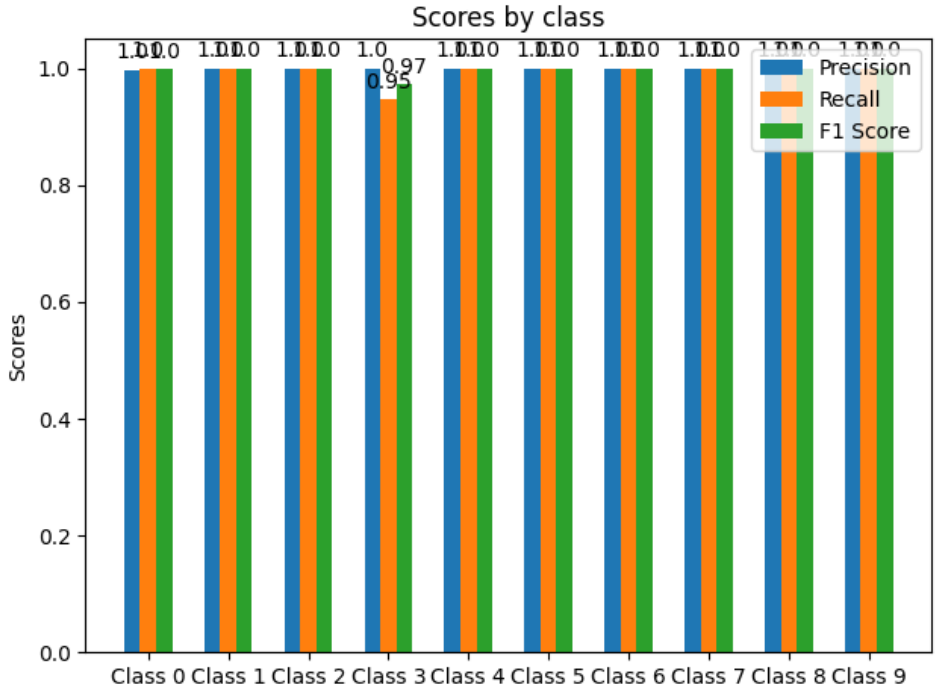

Figure 4.7: Per-class evaluation metrics for MLP.

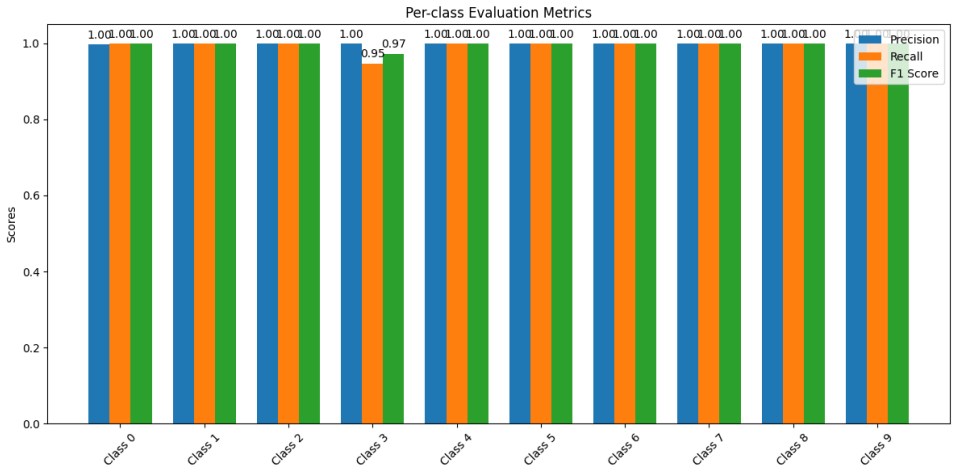

Figure 4.8: Per-class evaluation metrics for BiGRU.

BiGRU

BiGRU showed even better results on the tougher classes like Class 3 and 5. Its ability to look at how signals change over time really helped here, especially for waveform patterns that shift gradually.

CNN–LSTM–Attention

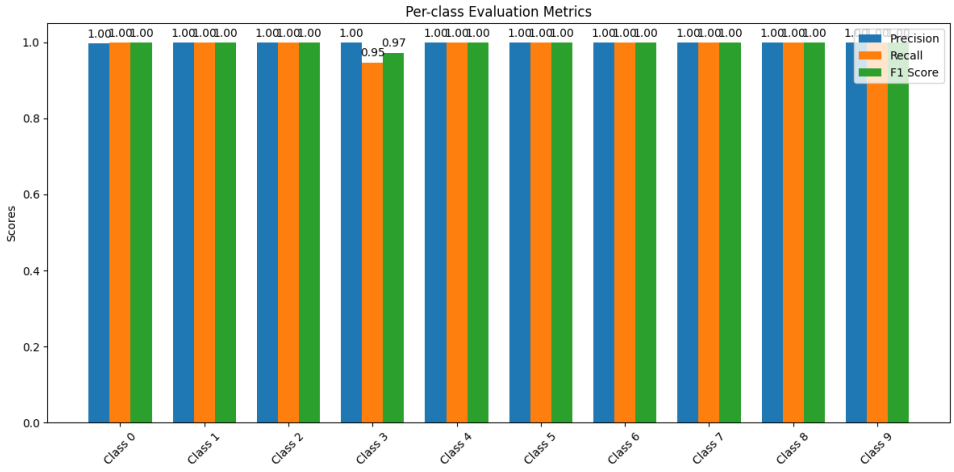

Figure 4.9: Per-class evaluation metrics for CNN–LSTM–Attention.

This model scored almost perfect across all classes. The attention mechanism likely helped it focus on subtle but important signal parts, making it very reliable, even on rare fault types.

4.4 Conclusion

We tested three models—MLP, BiGRU, and CNN–LSTM–Attention—on a fault classification task using current signals. All three worked well, with accuracy above 98%.
MLP gave a solid baseline. It's simple and fast, and did well even without sequence learning. But it had a bit of trouble when faults had similar signal shapes.
BiGRU was better at learning from time-based patterns. It handled class variation more smoothly and worked well on more complex examples.
CNN–LSTM–Attention performed the best. It used features from space, time, and attention to make very accurate predictions. It was the most stable and consistent model overall.
In short, while all three models are usable, CNN–LSTM–Attention is the most suitable for real-life use, especially in important railway systems.

4.5 Ideas for Future Work

Even though the current models worked well, there's still room to improve.

1. Cross-Entropy Loss Analysis

All models used standard cross-entropy loss. Future work could look into how this loss behaves per class and try other versions like focal loss [8] or label smoothing to deal with class imbalance or tough examples.

2. Ablation Studies

To better understand how each part of the model helps, we can try removing some parts:

- Try the model without attention and see what changes.
- Use a one-way GRU instead of BiGRU.
- Remove CNN or LSTM to check their individual effects.

These tests could help simplify the model while keeping good accuracy.

3. Explainability and Saliency Analysis

We can try methods like saliency maps to see which parts of the signal the model is focusing on. This would help users and engineers trust the system more.

4. Online Learning and Domain Adaptation

To handle real-time use, future versions can support online learning so the model improves as more data comes in. Also, domain adaptation could help it work on different machines or fault types.

5. Energy-Efficient Deployment

If we want to run the model on embedded systems, we can try shrinking it using pruning, quantization, or distillation. These can reduce memory and speed up inference.

6. Integration with Maintenance Systems

Finally, if the model is connected to the railway system directly, it could help send alerts, plan repairs, and improve safety by turning predictions into actions.

Chapter 5

Deployment Considerations and Practical Integration

After proving that the model works well in experiments, the next big step is making sure it also works in real-world railway environments. This chapter explains how we can move the fault detection system from a lab setup to a real-world deployment. It covers how the system is built, how it connects with railway maintenance tasks, what safety rules it must follow, and how we can keep it running smoothly over time.

5.1 System Architecture for Field Deployment

Figure 5.1 shows the overall design for how this system could work in the field. There are five main parts: collecting current signals, preprocessing the data, running the model, classifying faults, and saving the results.

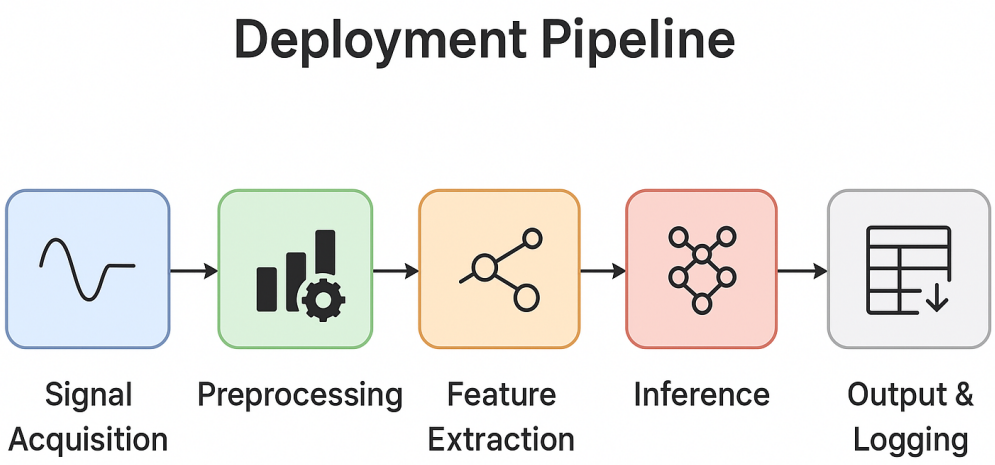

Figure 5.1: Proposed deployment pipeline from current signal acquisition to diagnostic output.

In practice, current signals are recorded automatically while the switch machine is working. An industrial sensor collects the signal, and a local data logger turns it into digital format. Then, an edge device (such as an ARM chip) processes the signal in real-time and runs the trained model to detect faults. The result can be shown on-site, and if needed, uploaded to a central system for maintenance tracking.

5.2 Integration with Existing Maintenance Workflows

Currently, railway maintenance teams mostly rely on regular inspections and people reporting issues manually. Our system can help by detecting problems in real-time, which means teams can respond faster and fix things before they become serious.
To make this fit into what railway staff already use, the system can connect with existing platforms that handle maintenance tasks. When a fault is found, the system can send out a notice with the fault type, the time it happened, and a picture of the waveform. This helps teams plan repairs more quickly and organize the parts and workers they need.

5.3 Compliance with Safety and Industry Standards

Since railway systems are critical for safety, any system we install must meet strict industry standards. We need to consider the following:

- **Data Integrity:** The data should be sent securely, using something like TLS, to make sure it's not lost or changed on the way.
- **Fail-Safe Behavior:** If there's a hardware failure or the model isn't confident, the system should default to a safe mode and alert operators just in case.
- **Model Interpretability:** The system should give clear reasons for its decisions. For example, it can use attention maps or basic rule-based outputs to help engineers understand the results.
- **Regulatory Compliance:** The system must follow official railway safety guidelines, such as TB/T 3084 in China or EN 50126 in Europe.

5.4 Monitoring, Updates, and System Maintenance

Once the system is deployed, we have to make sure it keeps working well. This includes keeping track of performance, updating the model, and checking the hardware.

- **Detecting Model Drift:** If the model's predictions start to shift or show unusual trends, we can look into whether the data has changed or the model needs to be updated.
- **Comprehensive Logging:** Save not just the final result but also the original signal and intermediate data, which helps with troubleshooting or future retraining.
- **Lifecycle Management:** We can automate the process of collecting new data and retraining the model. Updates can even be sent to devices wirelessly.
- **Hardware Health Checks:** The system should keep an eye on the sensors, processor, and communication parts to catch problems early.

These steps help make the system stable and reliable even as conditions change.

5.5 Resource-Aware Deployment and Optimization Strategies

Because the model runs on small devices, we need to make sure it doesn't use too much power or memory. There are a few ways to make this more efficient:

- **Quantization:** Reduce the model's precision (like using 8-bit integers) so it runs faster and takes up less memory, usually without losing much accuracy.
- **Pruning and Distillation:** Remove parts of the model that aren't very useful, or train a smaller model to act like the big one.
- **Batch Inference:** If the system doesn't need instant results, we can process several inputs at once to improve efficiency.

These tricks make sure the model stays fast and lightweight while still performing well.

5.6 Summary

This chapter explained how to turn our lab-tested fault detection system into something that can actually work on real railways. The system is modular, meaning it can be broken down into manageable parts. It works in real-time and fits into the tools railway staff already use. By paying attention to safety, system health, and speed, the solution is well-prepared for practical use. With this, we take a big step toward using AI to help maintain railway systems more efficiently and safely.

References

[1] C. Chen, X. Li, K. Huang, Z. Xu, and M. Mei. A convolutional autoencoder-based fault detection method for metro railway turnout. *Computational Modeling in Engineering & Sciences*, 136:471–485, 2023.

[2] W. contributors. Multilayer perceptron. `https://en.wikipedia.org/wiki/Multilayer_perceptron`, 2024. Accessed 19 May 2025.

[3] Dewesoft d.o.o. Guide to fft analysis (fast fourier transform), 2023. Accessed 18 May 2025.

[4] F. Duan, Y. Zhang, and C. Wang. Few-shot fault diagnosis of switch machines based on balanced regularised prototypical networks. *Engineering Applications of Artificial Intelligence*, 125:108847, 2024.

[5] M. A. El-Naggar and A. H. Elsayed. Investigating the impact of data normalization methods on predicting power load time series. *Energy Reports*, 10:3325–3337, 2024.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[7] M. Li, X. Hei, and W. Ji. A fault-diagnosis method for railway turnout systems based on improved autoencoder and data augmentation. *Sensors*, 22(23):9438, 2022.

[8] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[9] H. Liu, Y. Wang, and K. Zhang. Threshold-based fault detection for railway turnout machines. *Reliability Engineering & System Safety*, 199:106912, 2020.

[10] S. Nóbrega. Evaluating the impact of outlier treatment in time-series, 2024. Medium article, accessed 18 May 2025.

[11] I. A. of Public Transport. Rail transport statistics 2023. `https://www.uitp.org`, 2023. Accessed 12 May 2024.

[12] A. Sugiana, A. Nurenda, and S. Purba. Current-signal-based fault diagnosis of railway point machines using machine learning. *Applied Sciences*, 14(1):267, 2024.

[13] X. Sun, Y. Cao, T. Tang, and Y. Sun. A review of fault diagnosis for railway turnout systems. *Applied Sciences*, 13(22):12375, 2023.

[14] Y. Tian, Q. Zhou, and W. Zhang. Deep cnn–lstm network for railway switch motor current analysis. *IEEE Transactions on Industrial Electronics*, 68(11):10543–10552, 2021.

[15] Wikipedia contributors. Spectral centroid. `https://en.wikipedia.org/wiki/Spectral_centroid`, 2024. Revision 12 May 2024, accessed 18 May 2025.

[16] S. Zhang, F. Wang, and Q. Li. Railway switch fault diagnosis based on multi-head channel self-attention and residual deep cnn. *Transportation Safety & Environment*, 5:tdac045, 2024.

[17] S. Zhou, S. Guo, B. Du, and G. Jun. A hybrid framework for multivariate time series forecasting of daily urban water demand using attention-based convolutional neural network and long short-term memory network. *Sustainability*, 14(18):11234, 2022.

Appendix A

Self-appraisal

A.1 Critical self-evaluation

The development of this project, which focused on current signal classification using machine learning models, has been an extremely valuable experience. Throughout the process, I not only applied theoretical knowledge of deep learning architectures such as CNN, LSTM, and attention mechanisms, but also encountered various challenges that enhanced my problem-solving and implementation capabilities.

One of the key achievements was gaining a deeper understanding of complex architectures like CNN and LSTM, which I had previously explored only at a superficial level. Implementing these models in the context of time-series signal processing, and understanding their respective roles, was a significant milestone. The model achieved an accuracy of over 98%, exceeding my expectations and validating the use of both time-domain and frequency-domain feature extraction.

However, the process was not without difficulties. Hyperparameter tuning—particularly for the LSTM and CNN–LSTM–Attention models—was time-consuming and technically demanding. Adjusting learning rates, batch sizes, and architectural depth required extensive experimentation. Another notable challenge was the computational demand of training deep models, which was constrained by my hardware setup. This presents a clear opportunity for improvement in future work through the use of distributed or cloud-based training platforms.

Although the attention mechanism improved interpretability by highlighting important temporal segments of the signal, it remains difficult to fully understand the decision-making process of deep models. This lack of full transparency still poses a limitation in safety-critical applications.

In summary, this project significantly contributed to my growth as a data scientist. It reinforced not only technical depth in neural architectures, but also a broader understanding of how to manage real-world data—data that is often noisy, imbalanced, and incomplete.

A.2 Personal reflection and lessons learned

Looking back on this project, I have gained both technical insights and a renewed respect for the iterative nature of machine learning. Initially, I underestimated the complexity of training deep models. The first few training attempts were suboptimal, which emphasized the importance of patience, experimentation, and incremental improvement.

Data preprocessing turned out to be one of the most impactful stages. I had to carefully handle issues like class imbalance, missing values, and inconsistent signal lengths. Applying feature extraction and data augmentation techniques helped boost the quality and stability of the input data.

Another important takeaway was the value of feedback and discussion. While I conducted the

bulk of development independently, occasional discussions with mentors and classmates provided critical perspectives that shaped my model design choices. These collaborative exchanges helped prevent technical missteps and introduced new directions.
Finally, I came to appreciate that model accuracy alone is insufficient. Interpretability and explainability are crucial, especially in domains like fault detection where predictions must be trusted and justified. The attention mechanism was a good start, but further efforts toward transparency will be needed for real-world deployment.

A.3 Legal, Social, Ethical and Professional Issues

A.3.1 Legal Issues

One of the key legal concerns is data privacy. Although the dataset used in this project was anonymized and obtained from previously collected and curated records, future real-world deployments that involve live monitoring data must strictly adhere to national and international data protection regulations. For instance, the General Data Protection Regulation (GDPR) in the European Union imposes comprehensive rules for collecting, processing, and storing personal and sensitive information, including metadata or indirect identifiers. In industrial contexts, sensor data may not initially appear sensitive, but when combined with operational logs, time stamps, or location tags, it could be linked back to identifiable infrastructure components or personnel. To ensure legal compliance, robust data governance policies must be implemented, including encryption of stored data, access control mechanisms, audit trails, and data minimization principles. Legal accountability also extends to liability in the event of system failures caused by algorithmic decisions, which must be clearly addressed in contractual agreements.

A.3.2 Social Issues

Automation in fault detection brings both societal benefits and risks. On the one hand, AI-driven systems can significantly improve diagnostic speed, reduce human error, and enhance railway safety. On the other hand, there is a possibility of workforce displacement, especially for field engineers and manual inspection staff whose roles may be partially automated. This raises broader concerns about employment shifts in industrial sectors and the need for reskilling or upskilling programs to help affected workers transition to new roles in system oversight, data analysis, or AI maintenance. Moreover, the digital divide remains a pressing issue. Not all railway systems—especially in developing regions—have the necessary digital infrastructure, computational resources, or stable connectivity to support advanced AI deployment. This technological gap may lead to unequal access to safety enhancements and infrastructure modernization, reinforcing existing inequalities. Therefore, inclusive deployment planning and equitable technology access are important considerations for the social sustainability of such systems.

A.3.3 Ethical Issues

A central ethical issue lies in ensuring that the deployed AI system performs reliably and equitably across all fault classes. In practice, railway faults are imbalanced: some rare but dangerous failures may occur infrequently and thus be underrepresented in training data. If these categories are not adequately learned by the model, the system might overlook critical anomalies, leading to safety risks. This raises the question of accountability—who is to blame if the system fails to detect a fault? Is it the developer who trained the model, the vendor who integrated the system, or the operator who trusted the output? To address this, ethical AI development must involve rigorous bias testing, transparency in decision-making, and post-deployment monitoring. Additionally, explainability plays a critical ethical role. Stakeholders—including engineers, inspectors, and regulators—must be able to understand how and why the system makes certain decisions, especially in safety-critical environments. Incorporating explainable AI techniques can bridge the gap between algorithmic output and human trust, enabling responsible decision-making.

A.3.4 Professional Issues

This project has reinforced several core responsibilities that define professionalism in the AI and machine learning domain. First and foremost is the commitment to lifelong learning: the field is evolving at a rapid pace, and staying current with state-of-the-art methods, frameworks, and ethical standards is not optional—it is essential. Secondly, practitioners must ensure that their systems are not only technically effective but also socially responsible. In safety-critical industries like transportation, every stage—data preprocessing, model selection, training, evaluation, and deployment—must meet high standards of reliability, transparency, and documentation. Peer-reviewed validation, third-party audits, and compliance with international standards (such as ISO/IEC 25010 or EN 50126) are not just best practices; they are professional obligations. Additionally, communication is a key part of the professional role: machine learning engineers must be able to explain their systems to both technical and non-technical stakeholders. This project has also highlighted the importance of interdisciplinary collaboration. Real-world deployment involves coordination with domain experts, IT professionals, and legal advisors, underscoring that responsible AI development is not a solitary effort, but a team-driven, multidisciplinary pursuit.

Appendix B

External Material

This project is based on a real-world dataset collected from a railway signal maintenance department affiliated with the Shanghai Railway Bureau. Specifically, the electric current data were obtained from ZD6-type switch machines over a four-month monitoring period, ranging from August 1st, 2023 to December 1st, 2023. These signals were collected using a current monitoring device deployed in the field and represent actual operation cycles, including both normal and fault conditions.
The raw dataset contains 2,116 operation samples, each consisting of 200 evenly sampled current values. The dataset includes a variety of failure modes annotated by domain experts, covering issues such as relay malfunction, mechanical jamming, motor short circuits, and incomplete locking actions.
After acquisition, all data preprocessing and augmentation procedures were independently designed and implemented by the author. This includes:

- **Missing value handling:** Linear interpolation was applied to fill short gaps in sensor readings.
- **Outlier filtering:** Signals were clipped using percentile thresholds to reduce the effect of extreme spikes.
- **Data augmentation:** Gaussian noise and amplitude scaling were introduced to augment minority classes and improve class balance.

Feature extraction was also independently conducted by the author. Both time-domain statistics (e.g., RMS, skewness, crest factor) and frequency-domain descriptors (e.g., dominant harmonics, spectral centroid) were computed using NumPy and SciPy libraries.
The design, implementation, and validation of all three machine learning models—MLP, BiGRU, and CNN–LSTM–Attention—were carried out solely by the author. All models were implemented from scratch in PyTorch, without using any pretrained architectures.
No third-party datasets or pre-built software were used for core model training or feature computation. All scripts and modules were written and executed locally on a personal device for experimentation and evaluation purposes.