Degree Project in Technology

First cycle, 15 credits

Evaluating Machine Learning Models for Minimizing Downtime through Operator Task Scheduling

A Case Study in Computer Numerical Control (CNC) Production Lines

KAAN ÖZSAN

Evaluating Machine Learning Models for Minimizing Downtime through Operator Task Scheduling

A Case Study in Computer Numerical Control (CNC) Production Lines

KAAN ÖZSAN

Degree Programme in Computer Engineering
Date: June 4, 2025

Supervisor: Yasaman Khorsandmanesh
Examiner: Håkan Olsson
School of Electrical Engineering and Computer Science
Host company: Scania AB
Swedish title: Utvärdering av maskininlärningsmodeller för att minimera driftstopp genom operatörsuppgiftsschemaläggning i CNC-produktionslinjer

Abstract

In computer numeric control (CNC) production lines, recurring tasks such as tool changes and quality inspections are still manually performed by operators. When CNC machine (MC) cycles become misaligned, these tasks may overlap, leading to increased downtime, reduced throughput, and greater operator workload. This thesis explores how machine learning can support operator task scheduling by predicting the timing and overlap of these interventions.

To address limitations in historical MC data, a simulation program was developed based on a mathematical model of MC behavior, operator activities, and automation logic. The simulation generated synthetic signals that were used to train and evaluate machine learning (ML) models capable of forecasting operator-dependent events. The study compares three model architectures Long short-term memory (LSTM), Convolutional neural network (CNN), and Temporal convolutional network (TCN)—using performance metrics including F1-score, precision, recall, Hamming loss, and AUC.

The results show that no single model consistently outperformed the others, but architectural differences affected performance depending on task frequency and machine position in the line. LSTM models demonstrated advantages in handling long-range dependencies, particularly under continuous learning conditions. However, low label frequency and data sparsity limited overall performance.

Despite these challenges, the findings suggest that predictive scheduling with ML can enhance production line efficiency if supported by better signal tracking and model specialization. The study also outlines future research directions, including model modularization, graph-based learning, and reinforcement learning for task optimization.

Keywords

Machine Throughput, CNC Production Lines, Operator Task Scheduling, Machine Learning, Multi-Label Predictions, Temporal Convolutional Network (TCN), Convolutional Neural Networks (CNN). Long-Short Term Memory (LSTM), Machine Learning Model Comparison, Time Series Forecasting,Hyperparameter Optimization

Sammanfattning

I CNC-produktionslinjer utförs återkommande uppgifter som verktygsbyten och kvalitetskontroller fortfarande manuellt av operatörer. När maskinernas cykler inte är synkroniserade kan dessa uppgifter överlappa, vilket leder till ökad stilleståndstid, minskad genomströmning och högre arbetsbelastning för operatören. Denna uppsats undersöker hur maskininlärning kan stödja operatörers arbetsplanering genom att förutsäga tidpunkt och överlappning av dessa insatser.

För att hantera bristen på historisk maskindata utvecklades en simuleringsmiljö baserad på en matematisk modell av maskinbeteende, operatörsaktiviteter och automationslogik. Simuleringen genererade syntetiska signaler som användes för att träna och utvärdera maskininlärningsmodeller för att förutsäga operatörsberoende händelser. Studien jämför tre modellarkitekturer – LSTM, CNN och Temporal Convolutional Networks (TCN) – med hjälp av prestandamått som F1-poäng, precision, recall, Hamming-förlust och AUC.

Resultaten visar att ingen enskild modell konsekvent överträffade de andra, men arkitektoniska skillnader påverkade prestandan beroende på uppgiftens frekvens och maskinens position i linjen. LSTM-modeller uppvisade fördelar i hantering av långsiktiga beroenden, särskilt under kontinuerliga inlärningsförhållanden. Dock begränsade låg etikettfrekvens och datagleshet den övergripande prestandan.

Trots dessa utmaningar tyder resultaten på att prediktiv schemaläggning med hjälp av maskininlärning kan förbättra effektiviteten i produktionslinjer, förutsatt att bättre signalspårning och modelspecialisering införs. Studien pekar även ut framtida forskningsområden, inklusive modellmodularisering, grafbaserad inlärning och förstärkningsinlärning för optimering av operatörsuppgifter.

Nyckelord

Maskinprestanda, CNC-produktionslinjer, Operatörsplanering, Maskininlärning, Flermärkesklassificering, Temporal Convolutional Network (TCN), Konvolutionella neurala nätverk (CNN), Long Short-Term Memory (LSTM), Jämförelse av maskininlärningsmodeller, Tidsserieförutsägelse, Hyperparametertuning

Acknowledgments

I would like to thank my wife, Lina Westerlund, and my daughters, Emmylou, Athena, and Isolde, for their patience and support throughout this journey. I am also grateful to Professor Leif Lindbäck for offering some of the most enjoyable courses and projects during my education, but most importantly, for his invaluable guidance on how an engineer should approach decision-making, whether in design or problem-solving. Additionally, I extend my thanks to Fredric Lilliengrip for embracing my project idea and making its execution possible at Scania.

Stockholm, June 2025
Kaan Özsan

Contents

List of Figures

List of Tables

Listings

List of acronyms and abbreviations

AUC	area under curve
CNC	computer numeric control
CNN	Convolutional neural network
CT	cycle time
FN	false negative
FP	false positive
FPR	false positive rate
HMI	human–machine interface
IQR	interquartile range
LSTM	Long short-term memory
MAE	mean absolute error
MC	CNC machine
ML	machine learning
MSE	mean squared error
NVA	non-value-added
PLC	programmable logic controller
RMSE	root mean squared error
ROC	receiver operating characteristic
SCADA	supervisory control and data acquisition
SPC	statistical process control box
TCN	Temporal convolutional network
TN	true negative
TP	true positive
TPR	true positive rate

Chapter 1

Introduction

The fourth industrial revolution has introduced concepts such as smart factories, predictive maintenance, and intelligent sensor networks into modern manufacturing. With the increasing deployment of automation and sensor-equipped computer numeric control (CNC), production lines now generate large volumes of real-time operational data [1]. This data enables a range of applications, including adaptive scheduling and flow optimization. However, these advances also introduce new challenges, such as determining which signals to monitor and how to convert raw sensor data into interpretable operational insights. These decisions significantly affect how data is interpreted by decision-makers and influence the ability to make informed operational choices, such as identifying machine degradation or anticipating maintenance requirements.

This thesis focuses on the operator task scheduling problem in a CNC production environment. Operators are responsible for performing several periodic maintenance tasks throughout their workday. The study compares different machine learning approaches for estimating task durations and predicting potential overlaps between maintenance activities. The objective is to evaluate the strengths and limitations of various models using multiple performance metrics. The outcome of this work is a real-time decision-support tool designed to assist production line operators in prioritizing tasks more effectively and minimizing downtime caused by periodic maintenance.

1.1 Overview of Production Line

A CNC production line consists of multiple interconnected CNC machines (MCs), an automation system, statistical process control boxs (SPCs), and

buffers. Raw parts enter the line through the input station, while completed parts are discharged at the output station. Each MC performs a specific machining operation and operates independently once loaded with a part.

Since the late 1970s, MCs have been widely controlled by programmable logic controllers (PLCs), with a notable early implementation introduced by Allen–Bradley[2]. A PLC is an industrial-grade controller designed to manage low-level machine operations, such as spindle control, axis motion, and I / O signal processing, with high reliability and real-time responsiveness. Positioned between the hardware and the operator, the PLC also serves as the bridge to higher-level supervisory systems. Interaction with the machine is made possible through a human–machine interface (HMI), which provides operators with a graphical user interface to monitor processes, adjust parameters, and respond to alarms. At the highest control layer, a supervisory control and data acquisition (SCADA) system integrates multiple PLCs and HMIs across a plant or network, enabling centralized monitoring, data acquisition, and long-term process management[2].

MCs with different cycle times (CTs) introduce variability in overall line throughput. To maintain continuity in the production process, intermediate buffers are strategically placed between MCs to absorb throughput instability. Buffers prevent upstream blocking and downstream starvation, thereby improving overall flow stability.

To ensure product quality, the line is equipped with SPCs. These units are typically placed after specific MCs where quality checks are required and temporarily remove parts from the production flow for inspection. Based on the inspection outcome, parts are either scrapped or reintroduced into the line in a forward-only manner. Although the overall flow between machines is unidirectional, routing between a SPC and its associated buffer may be bidirectional. This local interaction is not considered a violation of the forward-only flow constraint, as it occurs within the same inspection stage rather than between distinct machining operations.

Automation governs part transportation and task scheduling across the line. This includes loading and unloading MCs, quality inspection routing, and buffer management. Each part processed in the line adheres to a predefined sequence of operations without the possibility of skipping steps or re-entering previously completed operations. An illustration of this routing logic, including the relationship between MCs, buffers, and SPCs, is shown in Figure 1.1.

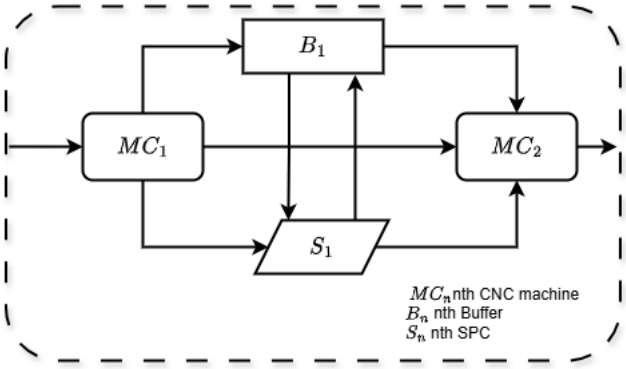


Figure 1.1: Illustration of part routing between machines, buffers, and inspection stations.

While automation handles part movement throughout the line, maintenance tasks for MCs are managed by the line operator. Each MC requires various types of maintenance at different intervals, with particular emphasis on periodic activities such as tool changes and quality checks. Tool changes can cause immediate production stops, while quality checks introduce temporary delays due to inspection and routing. These operator-dependent tasks introduce variability into the production flow and require timely decision-making, especially when they occur simultaneously. Consequently, this study investigates how machine learning models can support operator task scheduling by predicting maintenance-related events and optimizing task priorities to minimize downtime. The impacts of tool changes and quality inspections on production flow are discussed in more detail in Chapter 2 (Theory), while machine-to-machine dependencies are formally modeled in Chapter 3 (Method).

1.2 Research Objective and Questions

1.2.1 Research Objective

The objective of this thesis is to evaluate the performance of various machine learning models for predicting operator workload in CNC production lines, with the ultimate goal of minimizing downtime through intelligent task scheduling. The study adopts a human-centered perspective, focusing on how predictive insights can improve decision-making and alleviate operational bottlenecks.

In addition to real-time production signals, the research will assess the availability and quality of historical data from the CNC production line. If suitable historical data exists, it will be used to train and evaluate machine learning models. However, in the absence of reliable historical datasets, a simulation environment will be developed to replicate machine interactions and generate synthetic signals. This simulated data will emulate key production events—such as tool changes, task overlaps, and machine halts—allowing for a robust assessment of model performance under realistic and diverse operational scenarios.

1.2.2 Research Question

The central research question is: Which machine learning models perform best in predicting and scheduling operator tasks in a CNC production line environment, based on metrics such as accuracy, precision, latency, and robustness?

Sub-questions include:

- How do different models compare in handling real-time or simulated data for predictive task planning?
- Which model performs best at classifying operator task occurrences relevant to operator scheduling?
- Which machine learning approach, in the context of continuous learning, results in the least downtime during model updates and retraining?

1.3 Benefits, Ethics, and Sustainability

1.3.1 Benefits

This thesis aims to enhance production efficiency by minimizing downtime through predictive operator task scheduling using machine learning. The approach supports better workload distribution, potentially reducing operator stress and improving overall throughput. Additionally, the developed simulation and evaluation framework offers long-term value by being adaptable to various industrial environments beyond the case study.

1.3.2 Ethics

This project ensures ethical data handling by using only operational signals that do not contain any personally identifiable information. All real-time data presented in the thesis is proportionally randomized relative to the original data to prevent any potential leakage of sensitive company information. In two cases, tool change and quality check events, operator activity is timestamped to measure task duration. However, these timings are collected without recording which operator performed the task or when the event specifically occurred. No individual profiling is conducted, and all data remains fully anonymized throughout model training and analysis.

1.3.3 Sustainability

Reducing downtime contributes to sustainability by improving resource efficiency, lowering energy consumption, and minimizing material waste caused by production interruptions. Furthermore, by incorporating predictive planning into operator tasks, the project supports the development of smarter, more resilient manufacturing systems that align with long-term environmental and economic sustainability goals.

1.4 Organization

The following sections outline the structure of the report:

Chapter 2 presents the theoretical background relevant to CNC production lines, machine throughput modeling, and the fundamentals of machine learning methods applied in predictive scheduling. This chapter provides the necessary foundation for understanding the problem domain and the techniques used throughout the project.

Chapter 3 describes the methodology employed in the thesis, including data collection strategies, simulation setup, model training processes, evaluation metrics, and experimental design. It outlines how real-world constraints were addressed and how the different machine learning models were benchmarked.

Chapter 4 presents the results of the machine learning model evaluations, comparing their performance in predicting operator tasks across various metrics such as accuracy, precision, latency, and robustness. Results from both historical and simulated data scenarios are discussed.

Chapter 5 summarizes the key findings, discusses their implications for CNC production environments, and reflects on the limitations of the study. It also suggests directions for future research, particularly regarding continuous learning.

The complete implementation is available at: github.

Chapter 2

Theory

Section 2.1 introduces a mathematical model of machine throughput, including tool changes and quality checks as discrete time penalties. Section 2.2 outlines machine data flow in Industry 4.0 and the roles of PLC, SCADA, MES, and ERP. Section 2.3 defines the metrics and statistical tools used to evaluate model performance and signal behavior.

2.1 Mathematical Description of Machine Throughput

A fundamental expression for CT, denoted t_c, is typically composed of two main components. The first is the processing time t_m, which refers to the time spent cutting material. The second is the handling time t_h, representing the time required to load and unload parts from the machine. Unless otherwise stated, all-time variables are measured in seconds.

$$t_c = t_h + t_m. \tag{2.1}$$

However, in practical CNC operations, additional factors such as tool replacement introduce interruptions that are not captured by Equation (2.1), and therefore must be considered to improve the accuracy of CT modeling in MCs. A primary example is tool replacement. The duration required for a tool replacement is denoted t_t, and tool life is represented by N, which indicates the number of parts that can be machined before the tool must be replaced. By dividing the t_t by tool life, the average contribution of each tool change to the

cycle time is given by t_t/N. Incorporating this into the model yields:

$$t_c = t_h + t_m + \frac{t_t}{N}. \tag{2.2}$$

This formulation illustrates how machining time, tool life, and part handling durations jointly influence average throughput. It assumes uniform tool wear and consistent production conditions. In particular, reductions in t_m or increases in N lead directly to lower t_c, emphasizing the importance of tool optimization strategies to minimize downtime and improve production efficiency [3].

A comprehensive model for CNC process planning has been proposed in prior work to optimize CT per part using a combination of genetic algorithms and multi-criteria decision-making techniques such as Analytic Hierarchy Process method [4].

The model accounts for both machining (value-added) and non-value-added (NVA) activities. This study focuses on operator-interactive tasks, so machining activities are considered out of scope. Instead, this work focuses on extending the model to explicitly capture real-time discrete machine events, such as tool changes and quality inspections. These events generally generate corresponding operator tasks. This enhancement improves the accuracy of throughput estimation, supports short-term scheduling decisions, and improves operator task allocation.

Within this model, tool change is treated not as a foundational operation but as a composite operator task resulting from replacing tools or inserts. This task is quantified as a discrete, repeatable time penalty in the process sequence, formalized as:

$$\mathrm{Tach}_m^{\mathrm{tot}} = \sum_{n=2}^{N} \mathrm{X}_{\mathrm{acc}}^{n,m} \times \mathrm{Tach}_{\mathrm{unit}}. \tag{2.3}$$

Here, $\mathrm{Tach}_m^{\mathrm{tot}}$ represents the total tool change time for the machining process m, calculated by summing the constant tool change duration $\mathrm{Tach}^{\mathrm{unit}}$ each time a tool change is required. This is indicated by the binary variable $\mathrm{X}_{\mathrm{acc}}^{n,m}$, which is set to 1 if a tool change occurs between operations $n-1$ and n, and 0 otherwise. The model also includes a detailed formulation for insert change time, which is considered out of scope. This component accounts for the cumulative time penalties associated with scheduled insert replacements and tool wear.

Although quality checks are not explicitly modeled as standalone time

components, the model incorporates a risk index to account for quality-related concerns such as dimensional deviations or surface defects [4]. The risk associated with each operation is quantified by:

$$IR_{op}^{\mathrm{m,n}} = \frac{R_{op}^{\mathrm{m,n}}}{R_{\mathrm{threshold}}^{m,n}}. \tag{2.4}$$

In this formula, higher values of the risk index $IRop_{\mathrm{m,n}}$ indicate an increased likelihood of machining-induced defects. An operation is generally considered safe if $IRop_{\mathrm{m,n}} < 1$ [4]. In real-world manufacturing, these risks are typically managed through scheduled inspection intervals. The frequency of these intervals influences both product quality and scrap rate. These inspections may be performed through in-line measurement systems, automated inspection stations, or manual inspections. All inspections, except those conducted through in-line systems, are considered operator tasks and therefore require explicit scheduling.

2.2 Data flow in Industry 4.0

Industry 4.0 has accelerated the transition from analog to digital signal processing, enabling deeper insights into the operation of complex machinery. As part of this transformation, the PLC has become a central hub for collecting and processing machine signals. These signals are then transmitted through various industrial communication protocols to higher-level systems, including SCADA, MES, and ultimately, ERP[5].

This hierarchical flow of data, first conceptualized in the mid-1980s, is commonly referred to as the automation pyramid. It is a five-level architecture that organizes data access and control responsibilities based on the needs and functions of different stakeholders within a manufacturing system. Each level transforms raw operational signals into increasingly abstract and strategic insights[5].

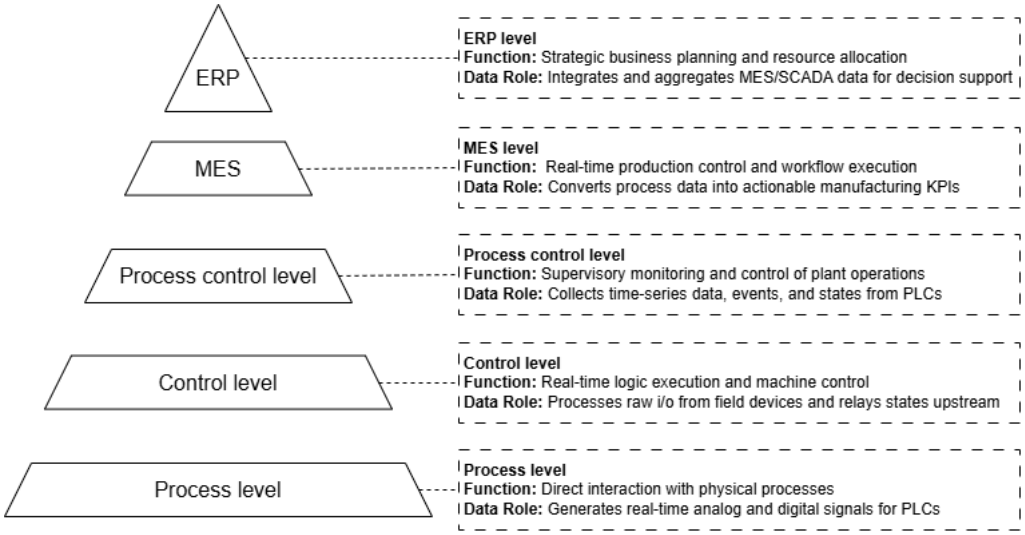


Figure 2.1: The Automation Pyramid illustrating layered data flow and control in Industry 4.0 environments.

As illustrated in Figure 2.1, each layer in the pyramid plays a critical role in the manufacturing data ecosystem. Of particular importance is the SCADA level, which serves as the supervisory control layer between machine-level logic and operational planning. SCADA systems typically poll each connected machine at one-second intervals to detect changes in operational state—commonly referred to as events. Upon detecting such an event, the SCADA system transmits the relevant information to the MES layer and updates associated databases[5].

This real-time data is subsequently accessed by ERP systems to support high-level enterprise functions such as business intelligence, strategic planning, and resource optimization. Understanding this hierarchical structure is essential for leveraging operational data in predictive scheduling systems and improving decision-making related to operator task prioritization[5].

2.3 Evaluation Metrics and Mathematical Formulations

This section outlines the quantitative tools used to evaluate model performance and interpret signal characteristics. It is divided into three parts: metrics for classification tasks, metrics for continuous predictions, and statistical measures that support theoretical and data-driven analysis. Together, these

components provide the mathematical basis for model assessment and signal evaluation throughout the study.

2.3.1 Performance Measures for Class Label Predictions

Each prediction made by a machine learning model results in one of four possible outcomes: true positive (TP), true negative (TN), false positive (FP), or false negative (FN). A TP occurs when the model correctly predicts the positive class, and a TN when it correctly predicts the negative class. FP refers to a negative instance incorrectly classified as positive, while FN refers to a positive instance incorrectly classified as negative.

These outcomes are typically summarized in a confusion matrix, shown in Table 2.1, which serves as the foundation for calculating various evaluation metrics.

Table 2.1: Tabular representation of classification outcomes used to derive evaluation metrics.

model / actually	**Positive prediction**	**Negative prediction**
True label	TP	TN
False label	FP	FN

The confusion matrix provides the basis for several standard evaluation metrics derived from TP, TN, FP, and FN. The most common of these are accuracy ...

- **Accuracy** is a widely used metric that quantifies the proportion of correct predictions out of all predictions made by the model [6]. It is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.5}$$

While accuracy is intuitive and easy to interpret, it can be misleading

in cases where the class distribution is highly imbalanced [6]. For instance, if only 1% of the dataset contains positive labels, a model that predicts all samples as negative would still achieve 99% accuracy, despite completely failing to identify any positive instances. In such scenarios, accuracy does not reflect the model's ability to detect the minority class and should be interpreted cautiously and complemented with additional metrics.

- **Precision**

  Precision measures the proportion of positive predictions that are actually correct. It is particularly useful when the cost of false positives is high, as it directly quantifies the model's ability to avoid incorrect positive predictions [6]. The formula for precision is:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{2.6}$$

  A high precision score indicates that positive predictions are likely to be correct. However, precision does not account for FN, which can lead to a misinterpretation of model performance [6]. For instance, consider a dataset that contains 100 positive instances. If a model predicts only one instance as positive and classifies the remaining 99 positive instances as negative, it will still achieve a precision of 100%, despite having missed the vast majority of actual positives. Therefore, precision should always be interpreted in conjunction with recall, as they provide complementary insights into classification performance.

- **Recall (Sensitivity)**

  Recall quantifies the proportion of actual positive instances that are correctly identified by the model. Recall is also known as the true positive rate (TPR) [6]. It is defined as:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{2.7}$$

  High recall indicates that the model identifies most of the actual positive instances, making it particularly important in applications where detecting positive cases is the primary objective. However, recall does not take false positives into account. As a result, a model that labels many instances as positive might achieve high recall by catching

most true positives, even if it also includes a large number of incorrect predictions.

For example, a model that predicts all samples as positive would achieve 100% recall, but it would likely have poor precision due to the high number of false positives. Therefore, recall should be interpreted alongside precision to provide a more balanced view of a model's performance [6].

- **F1-Score**

  The F1-score is the harmonic mean of precision and recall, offering a single measure that balances their trade-off. It is defined as:

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.8}$$

  The F1-score summarizes both precision and recall into a single metric, making it well-suited for evaluating classifiers on imbalanced datasets where accuracy alone can be misleading.

  However, the F1-score also has limitations. It assumes that precision and recall are equally important, which might not align with the actual cost structure of misclassification in a specific application [6].

- **Receiver Operating Characteristic and Area Under Curve**

  The receiver operating characteristic (ROC) curve is a graphical representation of a classifier's diagnostic ability across various threshold settings. It plots the TPR against the false positive rate (FPR) at different decision thresholds. The area under curve (AUC) summarizes this plot into a single scalar value that represents the model's overall ability to distinguish between classes.

  A model with an AUC of 1.0 perfectly separates positive and negative instances, while a model with an AUC of 0.5 indicates random guessing. ROC-AUC is particularly valuable because it evaluates performance independently of a specific decision threshold, making it useful for comparing models in early stages of selection.

  In highly imbalanced datasets, ROC-AUC may overestimate model performance. Since the FPR is defined as $FP/(FP + TN)$, a large number of negative instances causes small increases in false positives to have minimal effect on the FPR. This can result in deceptively high AUC values despite poor detection of the minority class [6].

2.3.2 Performance Measures for Continuous Predictions

This subsection describes evaluation metrics commonly used to assess the accuracy of regression models in continuous prediction tasks. Covered metrics include Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error. These measures quantify the deviation between predicted and actual values and offer different perspectives on model performance, particularly in how they respond to large errors and outliers.

- **Mean Squared Error**

  Mean squared error (MSE) is one of the most widely used performance measures for evaluating regression models. It quantifies the average of the squared differences between predicted and actual values. Formally, it is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2. \tag{2.9}$$

  MSE is particularly popular due to its mathematical properties. It is differentiable and "well-behaved", making it convenient for optimization in many machine learning algorithms. Squaring the errors ensures that larger discrepancies are penalized more heavily than smaller ones, which helps highlight cases where the model performs poorly [6].

  However, this sensitivity to large errors is also a notable drawback, especially for datasets that contain a large number of outliers [6].

- **Root Mean Squared Error**

  Root mean squared error (RMSE) is a related metric that takes the square root of MSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2}. \tag{2.10}$$

  By reintroducing the original scale of the target variable, RMSE facilitates more intuitive comparisons between predicted and actual values. Like MSE, it emphasizes larger errors but expresses the result in the same units as the output variable, improving interpretability [6].

RMSE is especially useful when large errors are particularly undesirable. However, its sensitivity to outliers makes RMSE less reliable in datasets with frequent noise or anomalies. As a result, it is often used alongside other metrics to ensure a balanced evaluation of model performance [6].

- **Mean Absolute Error**

  Mean absolute error (MAE) measures the average magnitude of prediction errors without considering their direction. It calculates the average absolute difference between predicted and actual values and is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i| \tag{2.11}$$

  Unlike MSE and RMSE, which square the errors and thus disproportionately penalize larger deviations, MAE treats all errors linearly. This makes it more robust in the presence of outliers, as it does not exaggerate the effect of large prediction errors [6].

  MAE is easy to interpret because it retains the same units as the target variable and provides an intuitive sense of average error magnitude. However, its linearity can be a limitation when large errors are especially undesirable. In such cases, RMSE may be preferable due to its sensitivity to large deviations [6].

2.3.3 Statistical and Theoretical Evaluation Tools

This subsection describes statistical tools for evaluating data variability, uncertainty, and distributional properties. Covered metrics include Shannon entropy, standard error, confidence intervals based on the Student's t-distribution, percentiles, and the interquartile range (IQR). These measures are used to characterize signal behavior in the presence of noise and limited data.

- **Entropy** Shannon entropy quantifies the uncertainty in a probability distribution by measuring the average information content of its outcomes:

$$\text{H}(X) = -\sum_{x\in X} p(x)\log_2 p(x) \tag{2.12}$$

where $p(x)$ is the probability of outcome x. Entropy is maximized under a uniform distribution.

In data analysis, entropy is used to assess the informational complexity or unpredictability of a signal. For time-series data—where noise and nonlinearity are prevalent—variants such as Approximate Entropy, Sample Entropy, and Permutation Entropy provide improved robustness to noise and small sample sizes [7].

- **Standard Error and the Central Limit Theorem**

  The Central Limit Theorem (CLT) states that, given a sufficiently large sample size n, the distribution of the sample mean $\bar{x}$ tends toward a normal distribution, regardless of the shape of the underlying population distribution. This result enables the use of normal-based statistical methods even when the original data are non-normal.

  Under the CLT, the sample mean $\bar{x}$ is not a fixed value but a random variable with its own distribution. The variability of this distribution is quantified by the standard error (SE), defined as:

$$SE = \frac{s}{\sqrt{n}} \tag{2.13}$$

  In this equation, s is the sample standard deviation and n is the sample size. The standard error estimates the standard deviation of the sampling distribution of $\bar{x}$. It reflects how much the sample mean varies due to sampling variability. Because of the CLT, this variability becomes predictable and approximately normal as n increases, making SE a critical component in constructing confidence intervals and performing hypothesis tests [8].

- **Confidence Interval with Student's *t*-Distribution**

  Building on the standard error introduced above a confidence interval quantifies the uncertainty associated with estimating the population mean μ from a sample. If the population standard deviation is unknown and the sample size is small, the Student's *t*-distribution is used instead of the normal distribution.

  The $100(1-\alpha)\%$ confidence interval for the population mean is given by:

$$\bar{x} \pm t_{\alpha/2,\, n-1} \cdot \text{SE} \tag{2.14}$$

Here, $\bar{x}$ is the sample mean, s is the sample standard deviation, n is the sample size, and $t_{\alpha/2,\,n-1}$ is the critical value from the *t*-distribution with $n-1$ degrees of freedom. This interval reflects the increased uncertainty from estimating the population standard deviation based on limited data [8].

- **Percentiles** A percentile indicates the value below which a given percentage of observations fall. The Pth percentile, denoted x_P, can be calculated from ordered data as:

$$x_P = (1 - w)x_j + wx_{j+1} \tag{2.15}$$

In this formula, x_j and x_{j+1} are adjacent ordered values surrounding the percentile rank, and w is a weight between 0 and 1 applied for linear interpolation. Percentiles, including the 50th percentile (median), are commonly used to identify outliers [8].

- **Interquartile Range (IQR)**

The IQR is a robust measure of statistical dispersion, representing the spread of the middle 50% of a dataset. It is defined as the difference between the 75th percentile (Q_3) and the 25th percentile (Q_1):

$$\text{IQR} = Q_3 - Q_1 \tag{2.16}$$

By excluding both extremes, the IQR is less influenced by outliers than the full range or standard deviation, making it especially suitable for skewed distributions [8].

2.4 Temporal Convolutional Networks

(Temporal convolutional network (TCN)s are designed for sequence modeling using a purely convolutional architecture. Unlike recurrent models, TCNs process data in parallel and capture long-range dependencies through dilated convolutions [9]. This architecture avoids common training issues such as vanishing gradients, and supports efficient training on modern hardware.

The TCN can be implemented modularly to support multiple architecture variants and hyperparameter settings. These variants are typically based on a shared set of design principles: causal convolutions, dilated filters, residual connections, and temporal pooling [10]. These components enable the model

to handle multivariate time series data with varying prediction horizons and label configurations.

2.4.1 Causal and Dilated Convolutions

A TCN uses causal convolutions to ensure that the output at time step t is only influenced by inputs at time t and earlier. This is enforced by using causal padding, which aligns the receptive field without leaking future information.

To expand the temporal receptive field without increasing model depth or kernel size, TCNs apply dilated convolutions. These introduce gaps between kernel elements, allowing the network to access inputs from increasingly distant past time steps. The operation is defined as:

$$(x *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d \cdot i} \tag{2.17}$$

where f is the convolution filter of size k, d is the dilation rate, and s is the current time index. By increasing d across layers, the model can capture long-range temporal dependencies more efficiently than with standard convolutions.

2.4.2 Residual Connections and Skip Paths

Residual connections are used to support deeper networks by stabilizing gradient flow during training. A residual block typically applies a transformation to the input and adds it back to the original input, optionally adjusting dimensions with a 1x1 convolution [11]. This allows the model to learn differences from the identity function rather than full mappings, which often leads to faster convergence.

In some configurations, intermediate outputs are aggregated through skip connections before the final layers. This gives the model access to multi-scale temporal features and helps preserve useful signals across the network depth.

2.4.3 Temporal Pooling and Output Layer

After the convolutional stack, the temporal dimension is reduced using a global average pooling layer. This converts the variable-length sequence output into a fixed-size feature vector, enabling dense layers to produce final predictions. For binary multi-label classification tasks, the final layer typically uses a sigmoid activation function applied to each output unit.

2.4.4 Training Configuration

The training process involves several configurable hyperparameters, including:

- Convolution kernel size and dilation rates
- Number of filters and dropout rate
- Activation function and padding type
- Optimizer, loss function, batch size, and epoch count

Training can also include label-specific weighting for imbalanced targets. Each convolutional layer is followed by dropout and batch normalization to improve generalization and speed up convergence [10]. TCNs, convolutional structure, support parallel training and inference, offering a significant performance advantage over recurrent alternatives.

2.5 Convolutional Neural Networks

Convolutional neural network (CNN) can also be applied to sequence modeling by applying one-dimensional convolutions along the temporal axis. Many of the concepts used in CNN-based architectures, such as causal convolutions, dilation, and residual blocks, also appear in TCNs. Therefore, the key principles behind convolutional sequence modeling—such as receptive fields, parallel training, and temporal pooling are described in section 2.4.

In this study, a 1D CNN is implemented with configurable architecture depth, kernel size, number of filters, and pooling method. The goal is to assess how shallow convolutional models perform on the same sequence prediction task, without using dilation or residual structures.

2.5.1 Convolutional Layers and Feature Extraction

Each convolutional layer applies a filter over local temporal windows in the input sequence. For an input $\mathbf{x} \in R^{T \times F}$, the output is computed as:

$$y_t = \sum_{i=0}^{k-1} f_i \cdot x_{t+i} \tag{2.18}$$

Unlike TCNs, the CNN model used here applies all convolutions with a fixed dilation rate of 1 and standard padding. All layers are causal only if

explicitly configured. The model uses ‘same‘ padding by default, preserving the input length.

2.5.2 Batch Normalization and Dropout

To stabilize training and reduce overfitting, batch normalization and dropout are optionally applied after each convolution. These mechanisms help the model generalize better to unseen data by normalizing activation distributions and introducing random deactivation during training [12, 13].

2.5.3 Temporal Pooling and Output Layer

After the final convolutional layer, the temporal output can either be flattened or reduced using global average pooling. This step condenses the temporal dimension and creates a fixed-size vector that can be passed to a dense output layer. A sigmoid activation function is used to support multi-label binary classification tasks [14].

2.5.4 Training Configuration

The training setup includes configurable hyperparameters such as kernel size, number of filters, activation function, dropout rate, optimizer, batch size, and epoch count. The model is trained using binary cross-entropy loss or a weighted variant when class imbalance is detected. As with TCNs, the convolutional structure enables parallel training over time steps, making the CNN model computationally efficient on modern hardware.

Chapter 3

Method

Section 3.1 extends the throughput model to include quality checks, tool changes, and machine dependencies, starting with an isolated machine model and adding line-level interactions via a readiness function. Section 3.2 outlines the limitations of MES data and describes how simulation, field studies, and preprocessing were used to generate training data. Section 3.3 details the data pipeline, model framework, and training setup, including hardware/software, data splitting, and architecture-specific components. Section 3.4 presents the validation process, covering averaging methods, Hamming loss, grid search, model selection, and continuous learning evaluation.

3.1 Mathematical Description of Machine Throughput

The knowledge gained in Section 2.1 highlights the negative effects of periodic maintenance on the throughput of an isolated MC. Previous models demonstrate how minimizing tool replacement time or reducing overall cycle time (CT) can improve MC throughput. However, they have limitations in their ability to represent interruptions and delays that affect MC throughput in real time. Therefore, an updated mathematical model is required. The model should capture time-dependent interruptions and account for MC dependencies within a production line.

3.1.1 Isolated Machine

An isolated MC operates independently, with no external dependencies except for loading and unloading parts, periodic maintenance, and scheduled quality

checks. It models the throughput of a MC operating under ideal flow conditions. These conditions include:

- No raw part shortages
- Constant availability of containers for produced parts
- Spindle speed and feed rate at 100%
- Fully operational MC (i.e., no unexpected downtime)
- Setup and configuration completed for the current part type

Under these assumptions, MC throughput may experience delays due to quality checks and interruptions resulting from required tool replacements.

3.1.1.1 Throughput Delays

Periodic quality inspections cause production delays at fixed intervals of k_{q} parts. This inspection interval is externally controlled. During each inspection, the operator evaluates part quality using either an automated measuring MC or manual tools, which requires a duration of T_{q}. If the part passes inspection (with probability $\mathrm{p_r}$), no further corrective action is taken. The time penalty associated with this quality control activity is expressed as:

$$T_{\mathrm{QC}}(i) = \delta_{\mathrm{q}}(i) \times T_{\mathrm{q}} - \delta_{\mathrm{q}}(i) \times T_{\mathrm{q}} \times p_{\mathrm{r}} \tag{3.1}$$

where

$$\delta_{\mathrm{q}}(i) = \begin{cases} 1, & \text{if } i \equiv 0 \pmod{k_{\mathrm{q}}} \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

The indicator function $\delta_{\mathrm{q}}(i)$ evaluates to 1 whenever the i-th part triggers an inspection. The subtraction term reflects the partial overlap between the quality control time and the machining time that would have been utilized if the inspection had not occurred.

3.1.1.2 Throughput Interruptions

Tool replacement, similar to quality inspection, occurs at fixed intervals denoted by k_{t}, and requires a duration of T_{t}. However, tool replacement differs from inspection in two key aspects. First, this activity interrupts the MCs throughput entirely, meaning no machining occurs and no parts are produced during the replacement period. Unlike inspections, which are

externally triggered, tool wear is monitored internally by the MC itself. When a tool reaches its wear limit, the MC automatically halts and awaits operator intervention to complete the replacement task. After replacement, the first part produced is subject to a mandatory quality check. The time penalty associated with this task is expressed as:

$$T_{\text{Tool}}(i) = \delta_{\text{t}}(i) \times (T_{\text{t}} + T_{\text{q}}) \tag{3.3}$$

where

$$\delta_{\text{t}}(i) = \begin{cases} 1, & \text{if } i \equiv 0 \pmod{k_{\text{t}}} \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

3.1.1.3 Isolated throughput model

The isolated MC model takes as input the number of parts, n, and returns the total time required to produce them. Under normal operating conditions, the MC processes each part in its CT plus the duration required for loading and unloading, denoted by T_l. Any interruptions or delays are modeled as additional time penalties and applied separately to the base processing time.

$$m_i(x) = \sum_{i=1}^{x} (CT + T_l + T_{\text{QC}}(i) + T_{\text{Tool}}(i)) \tag{3.5}$$

3.1.2 Machine in a line Environment

While the isolated MC model captures internal interruptions, extending it to a production line introduces additional complexity due to interactions with neighboring components, as described in Section 1.1.

Each MC in the line interacts with both upstream and downstream elements, such as adjacent MCs, buffers, and the automation system. These components jointly determine the MCs operational availability and ultimately influence overall line throughput.

An isolated MC operates within a discrete interaction plane, where tasks such as machining and tool life management occur in fixed cycles. In contrast, external dependencies, including automation and task scheduling, belong to a continuous interaction plane. Signals are sent by each MC to the automation system to request loading or unloading operations. While the execution of these tasks occurs in discrete time, their scheduling and prioritization across multiple MCs happen continuously.

The automation system uses a busy-waiting mechanism, running an infinite loop that continuously monitors the status of all components in the production line. The position of a component in this loop influences its scheduling priority, with tasks queued and serviced based on predefined logic. Once a task is received, the automation system executes a sequence of operations, with durations that may vary depending on the specific requirements of the subsequent production step.

This scheduling behavior introduces nondeterministic readiness, which can be modeled by an exponential distribution. To incorporate a deterministic decision rule, a probabilistic readiness function is defined as:

$$\theta_i(t) = \begin{cases} 1, & \text{if } \lambda_i e^{-\lambda_i t} \geq \text{threshold} \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

where $i \in \{\mathrm{B_{in}}, \mathrm{B_{out}}, MC_{i-1}, MC_{i+1}\}$. Here, λ_i denotes the service rate of the automation system for component i and $\theta_{\mathrm{i}}(t)$ reflects whether the component is available at time t. Due to their shared structural characteristics and influence on MC behavior, these dependencies are grouped into a unified dependency set:$D_i(t) = \{\theta_{B_{\mathrm{in}}}(t),\ \theta_{B_{\mathrm{out}}}(t),\ \theta_{m_{i-1}}(t),\ \theta_{m_{i+1}}(t)\}$

The overall readiness function is then defined as

$$\Theta_{\mathrm{i}}(t) = \prod_{x \in D_{\mathrm{i}}(t)} \theta_x(t) \tag{3.7}$$

The extended MC model is defined conditionally on the readiness function $\Theta_{\mathrm{i}}(t)$, and is notated as $\tilde{m}_i(n, \Theta_{\mathrm{i}}(t), t)$:

$$\tilde{m}_i(n, \Theta_{\mathrm{i}}(t), t) = \begin{cases} m_i(n), & \text{if } \Theta_i(t) = 1 \\ \infty, & \text{otherwise} \end{cases} \tag{3.8}$$

Accordingly, the model presented in Equation (3.8) provides the formal basis for constructing a digital representation of the production line. This formalism also underpins the simulation model used to generate synthetic time series data, which serves as input for training machine learning (ML) models.

3.2 Data Collection and Simulation Configuration

Data for this study was initially collected from the Manufacturing Execution System (MES) layer, as defined by the automation pyramid described in

section 2.2. While the SCADA level captures a broad range of MC signals used by operators to monitor the production line in real time, these signals are not persistently stored and are therefore unavailable for historical analysis. To evaluate the suitability of MES data for predictive modeling, an entropy analysis, as defined in eq. (2.12), was conducted to measure the informational content of the available signals, indicating their variability and potential predictive value.

The analysis revealed that several important MC signals are not stored long-term in the MES database. As shown in table 3.1, key components exhibited complete information loss relative to the entropy required to capture the signal variability needed for effective model training. This suggests that MES data alone may be insufficient to support the development of reliable predictive models for downtime or operator task scheduling.

Table 3.1: Entropy values for selected components. Required, collected, and stored entropy in bits, with calculated information loss.

	Required entropy (bits)	**Collected entropy (bits)**	**Stored entropy (bits)**	**Information loss**
MC_1	7.90	10.48	0.00	100.00%
MC_2	4.71	8.41	0.00	100.00%
MC_4	5.84	8.17	0.00	100.00%
MC_5	4.32	4.32	0.00	100.00%
MC_6	7.01	9.34	0.00	100.00%
in_line	3.32	3.32	0.00	100.00%
out_line	3.46	3.46	0.00	100.00%

Due to the observed information loss, a production line simulation program was developed to generate synthetic signals that reflect realistic MC states and operator interventions. The simulation environment is based on the mathematical model of production line dynamics described in eq. (3.8). Its configuration required discrete event data to mimic real-world conditions. Historical data from the MES layer was used to gather CT for each MC. Other configuration data points, such as inspection durations, inspection intervals, and the overall structure of the production line, were missing and needed to complete the simulation configuration. To address this issue, several field studies were conducted on the affected production line.

The duration of the tool replacement task was determined by conducting several benchmarks involving different operators across various shifts. This

benchmarking was also conducted on other production lines with similar tooling and similar tool replacement tasks. The collected data was analyzed statistically, and a representative tool replacement duration was defined and used in the simulation configuration.

Historical data from the MES layer includes machine state values spanning approximately two years. For this study, the most recent six months of data were extracted and analyzed to estimate the CT for each MC. The dataset contains a significant amount of noise, which required preprocessing. To clean the data, the following strategies were applied and compared: IQR filtering, z-score filtering, and percentile-based removal (2.5%). Table 4.1 shows the results of these methods, and fig. 4.1 visualizes the differences.

The CT values were calculated through field studies by comparing the cleaned CT values against observed machine behavior. Among the tested approaches, the IQR based method produced estimates that most closely matched actual cycle times.

3.3 Data Handling and Model Development

3.3.1 Hardware and Software Environment

The experiments in this study were carried out on the hardware and software environment detailed in Tables 3.2 and 3.3. These configurations were used throughout model development, data preprocessing, training, and evaluation. Notably, TensorFlow version 2.19.0 was used, which—like other versions after 2.10.0—does not support GPU acceleration in certain Windows setups. As a result, all computations were executed on the CPU. This may have impacted training time and performance. The results reported in table 4.1 could potentially improve with GPU utilization.

Property	Value
OS Name	Microsoft Windows 11 Enterprise
OS Version	10.0.22631 N/A Build 22631
System Manufacturer	HP
System Model	HP ZBook Power 15.6 inch G10 Mobile Workstation PC
System Type	x64-based PC
Processor(s)	13th Gen Intel(R) Core(TM) i7-13800H, 2.5 GHz, 14 cores, 20 threads
BIOS Version	HP V97 Ver. 01.02.02, 2023-09-21
Total Physical Memory	32.0 GB
Virtual Memory (Max Size)	38.9 GB

Table 3.2: Hardware and operating system specifications of the workstation used during model development, testing, and evaluation.

Library	Version
pandas	2.2.3
numpy	2.1.3
joblib	1.4.2
json	2.0.9
TensorFlow	2.19.0
matplotlib	3.10.0
scikit-learn	1.6.1

Table 3.3: Software library versions used during the implementation and experimentation phases.

3.3.2 Splitting Data

Cross-validation traditionally assumes that data points are independent and identically distributed, meaning there is no correlation between them. Several types of cross-validation exist, such as Leave-One-Out (LOO), Leave-P-Out, and K-Fold cross-validation. However, this key assumption of independence

does not hold for time series data, where each data point is often correlated with its neighboring points [6][15].

Due to this temporal dependency, time series data require a different approach to splitting. In scikit-learn, the TimeSeriesSplit function is specifically designed to handle time-dependent data. This method splits the data set into sequential training and testing sets, where each new fold includes all prior observations and adds a new set of test instances [16]. This behavior is illustrated in Figure 3.1 While TimeSeriesSplit preserves temporal ordering, it may introduce certain challenges when evaluating model performance.

One concern is data leakage, which occurs when information from future observations becomes accessible during the model training phase. This typically leads to overly optimistic performance metrics and reduces the model's generalization ability. In time series contexts, data leakage arises when the training data includes patterns or signals that would not be available at prediction time.

Another issue is the imbalance in training set sizes across different folds. For example, the first fold is trained on a smaller portion of the data, approximately total data$/k) \times$ train_portion, whereas the final fold may use nearly the entire training set. This discrepancy can make comparisons between early and late models unfair. Early models may suffer from underfitting due to limited data, while later models may overfit due to excessive exposure to the dataset. As a result, model performance may vary not due to intrinsic model quality but due to the uneven data distribution across folds.

To address the issues of data leakage and unequal training set sizes introduced by expanding-window validation strategies, a sliding window approach was developed for this study, Figure 3.1 illustrates the sliding window method. The aim of this method is to create a fair comparison across models by ensuring that each model is trained on an equal number of data points and validated on a consistent test set size.

In this approach, the dataset is segmented into multiple consecutive windows of fixed duration. Each window is then split into training and testing subsets using the same proportions. Unlike the expanding-window technique, the training set does not grow over time, which ensures that no model benefits from access to significantly more data than others. Furthermore, because each window contains a self-contained time segment without forward overlap, the risk of data leakage is minimized. This makes the evaluation of each model more consistent and reliable, particularly in the context of time-dependent sequences.

However, this approach also has certain limitations. Since only a subset

of the total dataset is used in each window, the full dataset is not exploited as efficiently as in cumulative training strategies. As a result, valuable information available in earlier or later segments may be excluded from training. Additionally, due to the temporal nature of the data, each sliding window may capture slightly different statistical properties. This means that some models might be trained on relatively noisy or unstable segments, while others benefit from more stable periods. These temporal distribution shifts across windows can affect performance consistency and should be carefully considered during result interpretation results.

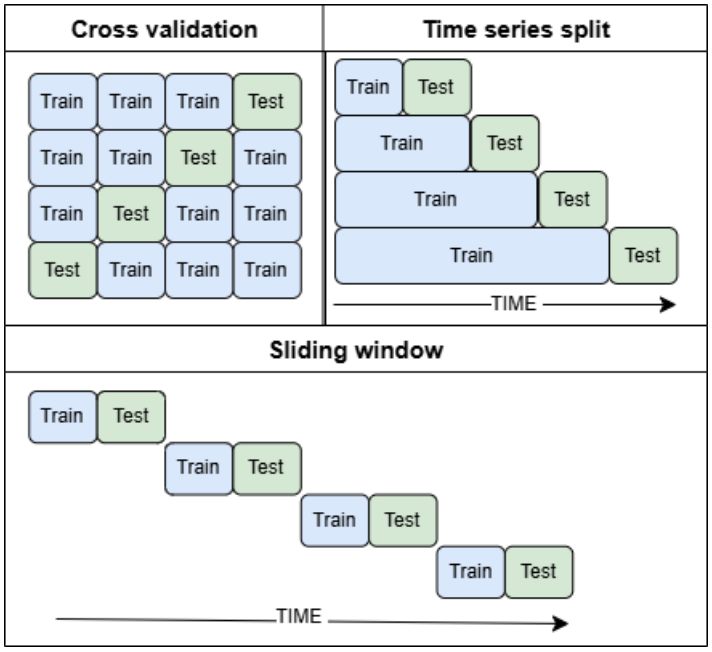


Figure 3.1: Comparison of K-Fold, TimeSeriesSplit, and Sliding Window strategies for model evaluation on time series data. The illustrated proportions of training and test sets are schematic; in practice, training sets are considerably larger than test sets. The visualization focuses on temporal sequencing rather than scale accuracy.

3.3.3 Data Transformation

Before model training, the raw machine signal data was transformed through a series of preprocessing steps to ensure consistent formatting, valid types, and semantically meaningful features. These transformations were implemented in the DataPreparation class, which includes column normalization, type

conversion, feature encoding, and label generation.

One-hot encoding was applied to handle categorical features, particularly machine state indicators such as MachineState. These indicators represent discrete operational modes (e.g., Working, Idle, Stopped) that have no inherent ordinal relationship. Encoding such states with integers (e.g., 1 for Working, 2 for Stopped) would imply an artificial hierarchy or magnitude, where assigning a higher numerical value to Stopped than Idle would incorrectly imply a greater severity or priority. To avoid introducing such artificial structure, one-hot encoding transforms each categorical state into a binary vector, representing each state as a separate column with values 0 or 1. This ensures all states are treated as equally distinct without implying any relative ordering. Although one-hot encoding preserves semantic correctness, it increases the dimensionality of the feature space; for instance, the original machine state feature occupied a single dimension, whereas the transformed version spans four binary dimensions.

After cleaning and encoding, the data was pivoted to a wide format where each timestamped row contains feature values for all machines. This layout facilitates the use of temporal models like LSTM and TCN by ensuring that each input sequence captures the full production line state at each time step.

Finally, label columns were derived to indicate the occurrence of operator-dependent tasks (tool changes and quality checks), based on event traces in the historical signal data. These labels are aligned with the forecast horizon during model training.

3.3.4 Model Training

To enable structured and reusable model development, a centralized ModelFactory class was implemented following the factory design pattern. This design encapsulates the initialization and coordination of internal modules such as data preparation, evaluation, plotting, and model instantiation. The use of a factory structure promotes separation of concerns and provides a unified interface for model selection, training, and evaluation.

ModelFactory internally manages several components:

- **DataPreparation:** Handles all preprocessing logic, including timestamp normalization, feature encoding, pivoting of machine-specific data, and label generation. This module standardizes the structure and semantics of input data across models.
- **ModelEvaluating:** Computes multi-label classification metrics such

as micro/macro precision, recall, F1-score, exact match accuracy, and Hamming score. This abstraction simplifies model comparison and ensures consistency in evaluation across experiments.

- **ModelPlotting:** Provides visualization utilities such as per-label confusion matrices and training loss/accuracy curves. This helps interpret model behavior and assess learning progression.

To support multiple temporal architectures while avoiding duplicated logic, a BaseModel class was introduced and inherited by each model variant. This class implements shared methods for sequence generation, training configuration, history tracking, and prediction buffering. The common interface enables consistent training workflows while preserving the flexibility to define architecture-specific layers.

Each subclass overrides the build() method to specify its own network topology:

- LSTMModel: Uses one or more stacked (optionally bidirectional) LSTM layers followed by dense output layers.
- CNNModel: Applies multiple 1D convolutional layers with dropout and pooling to capture short-range temporal patterns.
- TCNModel: Implements dilated causal convolutions with optional residual connections and skip layers to support long-range temporal dependencies.

All models operate on sliding window sequences extracted from pivoted time-series input, predicting binary labels for operator-dependent events such as tool changes and quality checks. During training, binary cross-entropy loss is minimized, optionally with class weights to handle imbalance, using the Adam optimizer. Evaluation is performed over a fixed number of epochs, and predictions are assessed both on validation and test sets.

The object-oriented structure not only improves maintainability but also enables rapid experimentation across model types. Switching between architectures requires no changes to the surrounding pipeline, as models conform to a shared interface. This design supports grid search, performance comparison, and modular extension in a structured and reproducible manner.

3.4 Model Validation

3.4.1 Macro and Micro Averaging

The classification metrics introduced earlier in section 2.3.1 are aggregated using various averaging strategies to summarize the overall model performance in multiclass and multilabel settings.

- **Micro averaging**
  Micro averaging aggregates contributions from all classes by summing the individual true positives, false positives, and false negatives before computing each metric. This method reflects the classifier's overall ability across all instances, giving higher weight to more frequent classes. It is especially useful in imbalanced datasets or when the total number of correctly classified instances is the primary concern.

$$\text{Accuracy}_{\text{micro}} = \frac{\sum TP + \sum TN}{\sum TP + \sum FP + \sum TN + \sum FN} \tag{3.9}$$

$$\text{Precision}_{\text{micro}} = \frac{\sum TP}{\sum TP + \sum FP} \tag{3.10}$$

$$\text{Recall}_{\text{micro}} = \frac{\sum TP}{\sum TP + \sum FN} \tag{3.11}$$

$$\text{F1-score}_{\text{micro}} = \frac{2 \cdot \sum Tp}{2 \cdot \sum TP + \sum FP + \sum FN} \tag{3.12}$$

  Micro-averaging may obscure poor performance on minority classes because it is dominated by the majority class.

- **Macro averaging**
  Macro averaging, in contrast, computes each metric independently for every class and then takes the unweighted average. This approach treats all classes equally, regardless of their frequency in the dataset. It is particularly suited for understanding model performance across all classes without favoring the majority class.

$$\text{Accuracy}_{\text{macro}} = \frac{1}{n} \sum_{i=1}^{n} \frac{TP_i + TN_i}{TP_i + FP_i + TN_i + FN_i} \tag{3.13}$$

$$\text{Precision}_{\text{macro}} = \frac{1}{n} \sum_{i=1}^{n} \frac{TP_i}{TP_i + FP_i} \tag{3.14}$$

$$\text{Recall}_{\text{macro}} = \frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i + FN_i} \tag{3.15}$$

$$\text{F1-score}_{\text{macro}} = \frac{1}{n}\sum_{i=1}^{n} F1_i \tag{3.16}$$

Macro averaging can give a misleading impression of overall model performance if some classes have very few samples, as it assigns equal weight to all classes regardless of their size or importance.

3.4.2 Hamming Loss

Hamming loss measures the fraction of labels that are incorrectly predicted. It is particularly useful in multi-label classification, where each instance may be associated with multiple labels. Unlike subset accuracy, which requires all labels to be correctly predicted for a sample to be counted as correct, Hamming loss penalizes individual label mistakes independently. The score ranges from 0 (perfect classification) to 1 (all labels incorrect)[17].

$$\text{Hamming Loss} = \frac{1}{n_{\text{samples}} \cdot n_{\text{labels}}}\sum_{i=1}^{n_{\text{samples}}}\sum_{j=1}^{n_{\text{labels}}} 1\left[y_{i,j} \neq \hat{y}_{i,j}\right] \tag{3.17}$$

Here, $n_{samples}$ represents the total number of instances, n labels is the total number of labels, $y_{i,j}$ denotes the true label for sample i and label j, $\hat{y}_{i,j}$ is the predicted label for sample i and label j, and $1\left[y_{i,j} \neq \hat{y}_{i,j}\right]$ is the indicator function, which evaluates to 1 if the condition is true and 0 otherwise.

A related metric, the Hamming score, is defined as:

$$\text{Hamming Score} = 1 - \text{Hamming Loss} \tag{3.18}$$

Hamming loss provides a more balanced view in multi-label settings, especially when some labels are rare or when partial correctness should be rewarded. It complements other metrics by revealing how often individual label predictions are wrong, independent of the full-label correctness for a sample.

3.4.3 Hyperparameter Tuning with Grid Search

Hyperparameter tuning was conducted using grid search to refine model performance and investigate the effects of architectural choices. This method systematically tests all predefined combinations of selected hyperparameter values, with each configuration evaluated against validation performance. The total number of models trained during grid search is calculated as $\prod_{i=1}^{k} N_i$, where k is the number of hyperparameters and N_i is the number of options for the i-th hyperparameter. Specifically, 192 models were created for both the TCN and CNN architectures (parameters for these can be reviewed in table 4.3 and table 4.4). For the Long short-term memory (LSTM) model, 96 configurations were trained and validated, resulting in a grand total of 480 unique models created across all architectures. While this approach is computationally demanding, its extensive coverage of the parameter space and facilitation of reproducibility justify the cost.

To minimize runtime overhead during tuning, all data preprocessing steps were executed outside the model training loop. This allowed the model to operate directly on preprocessed input sequences, eliminating redundant reshaping or formatting operations during each run. Observed runtimes, summarized in table 4.9, support the efficiency of this design choice.

The label weights were computed to address class imbalance by assigning higher importance to underrepresented labels. Weights were derived using inverse label frequency with smoothing, normalized to ensure numerical stability with a mean value near one. Additional scaling was applied to emphasize tool-related events and de-emphasize more frequent quality signals.

Each configuration was trained on three months of simulated time-series data and subsequently evaluated. Evaluation results are detailed in section 4.2.

All model versions were trained under consistent conditions, enabling a controlled comparison of architectural changes and their impact on performance.

3.4.4 Model Selection

Grid search resulted in several models for each ML architecture. To identify the best models, a filtering and ranking procedure was applied. Configurations were grouped then sorted by macro-F1 and micro-F1 scores in descending order and using result of eq. (3.17) in ascendong order. The top ten models from each base model were selected for detailed inspection.

These results suggest that performance is sensitive to the interaction between temporal parameters (window_size, forecast_horizon) and model

depth (via num_filters and kernel_size). Additionally, label weighting had a mixed impact depending on the version.

The selection process provided a clear set of high-performing configurations, which were then used for further qualitative and quantitative analysis in the next sections.

3.4.5 Model Performance with Continuous Learning

To examine how different model architectures behave under continuous learning scenarios, a rolling evaluation strategy was applied. This approach aims to simulate a realistic deployment setting where models are incrementally updated as new data becomes available over time. The evaluation was structured to measure changes in predictive performance and computational efficiency as models are retrained and tested on sequential data intervals.

The evaluation began by training each model using a three-month dataset, selected based on the best-performing hyperparameter configurations identified during earlier grid search. An initial validation was performed on a separate holdout set to establish a performance baseline.

The remaining data was then segmented into ten-day intervals, forming a sequence of time windows. Each window included a new training portion with 90% of the data and an associated test portion comprising the remaining 10%. This structure simulates a real-world deployment scenario in which new data becomes gradually available for model retraining.

For each interval, the following procedure was repeated:

1. The model was trained on the current interval's training set.

2. Predictions were generated for the test set by transforming it into input sequences using the model's sliding window configuration.

3. Evaluation metrics—including macro and micro F1-score, precision, recall, Hamming score, and runtime duration for both training and inference—were recorded.

4. The model was then updated using the test data, enabling a continuous learning cycle across time windows.

When results are obtained for an interval, the corresponding testing data is subsequently incorporated into the model's training set for the next iteration. This continuous integration of new data influences the model's performance on subsequent windows. This approach yields conservative but realistic

performance estimates by ensuring gradual adaptation and reducing the risk of data leakage or overfitting.

By storing and visualizing evaluation results across all intervals, this method allows for a detailed analysis of how model performance evolves with repeated exposure to new data. The results reflect not only the model's predictive capacity but also its computational efficiency and resilience under incremental retraining. This setup enables a fair comparison of different architectures under consistent conditions, as explained in section 3.3.

Chapter 4

Results

This chapter presents the results of the conducted experiments. Section 4.1 summarizes label frequencies in the dataset. Section 4.2 presents grid search results, visualizing tested parameter configurations and identifying the best-performing settings for each model. Section 4.3 reports model performance under continuous learning using the selected configurations. Section 4.4 provides a comparative analysis of all models based on key performance metrics.

4.1 Label Distribution Overview

Table 4.1: Comparison of statistical properties for raw and cleaned cycle time (CT) data. The table presents min, max, mean, standard deviation, and error margin values for each data cleaning method.

	Min (s)	**Max (s)**	**Mean (s)**	**Std (s)**	**Error Margin (s)**
Raw Data	35.2	595708.9	1275.1	23846.1	±2218.9
IQR Cleaned	158.0	183.3	169.8	3.2	±0.4
Percentile Cleaned	38.3	1493.8	209.1	175.9	±16.8
Z-score Cleaned	35.2	25473.3	322.4	1210.4	±112.7

Figure 4.1: Data cleaning

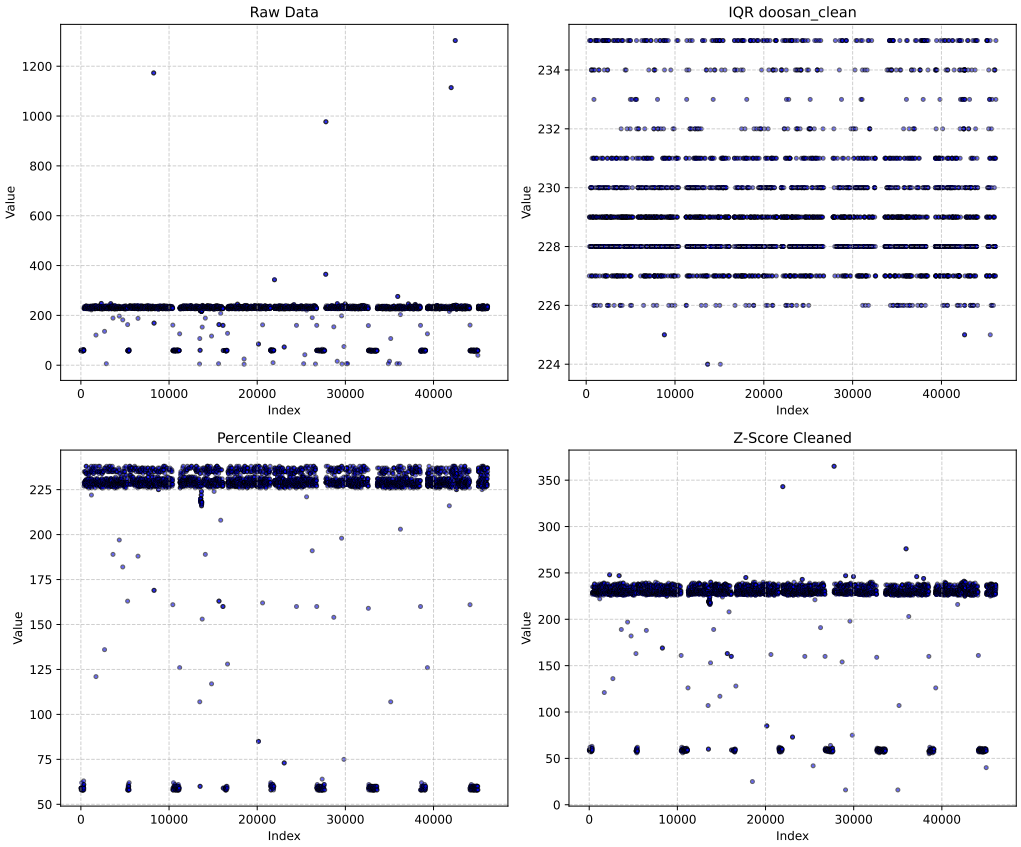


Table 4.2: Label frequency in 3 months data

Machine	Tool replacement	Quality inspection
MC_1	0.4%	0.9%
MC_2	0.9%	1.7%
MC_3	0.2%	0.4%
MC_4	0.4%	0.8%
MC_5	0.1%	0.2%
MC_6	0.3%	0.5%

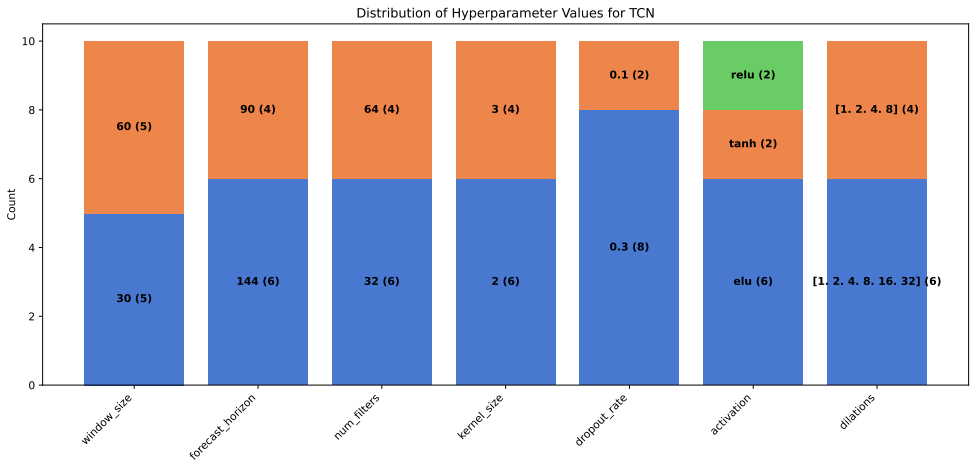
4.2 Grid Search Results

4.2.1 TCN

Table 4.3: Parameter ranges used in the grid search for tuning the TCN model

Parameter	Value
window_size	[30, 60]
forecast_horizon	[90, 144]
num_filters	[32, 64]
kernel_size	[2, 3]
dropout_rate	[0.1, 0.3]
activation	['relu', 'tanh', 'elu']
dilations	[[1, 2, 4, 8], [1, 2, 4, 8, 16, 32]]

Figure 4.2: Parameter Frequencies Among Top 10 Grid Search Results for the TCN Model

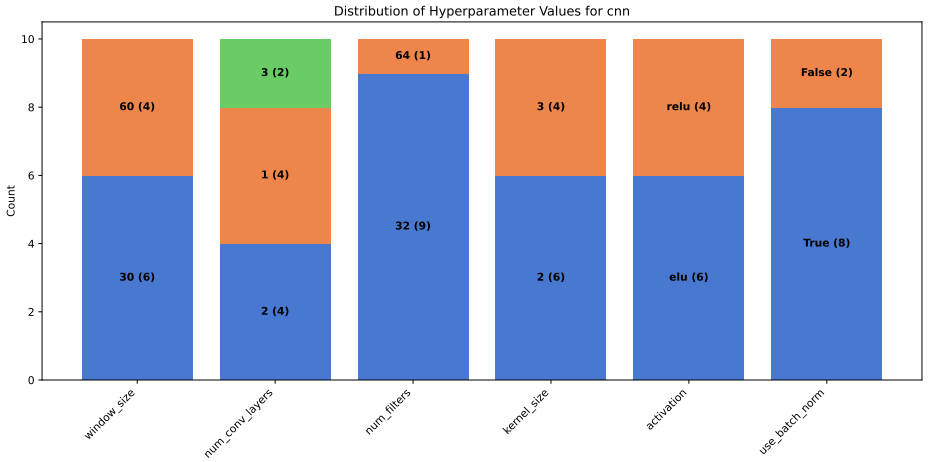


4.2.2 CNN

Table 4.4: Parameter ranges used in the grid search for tuning the CNN model

Parameter	Value
window_size	[30, 60]
num_conv_layers	[1, 2, 3]
num_filters	[32, 64]
kernel_size	[2, 3]
dropout_rate	[0.1, 0.3]
activation	['relu', 'elu']
use_batch_norm	[True, False]
use_global_pooling	[True]
version	['V1']
batch_size	[32]
epochs	[10]

Figure 4.3: Parameter Frequencies Among Top 10 Grid Search Results for the CNN Model

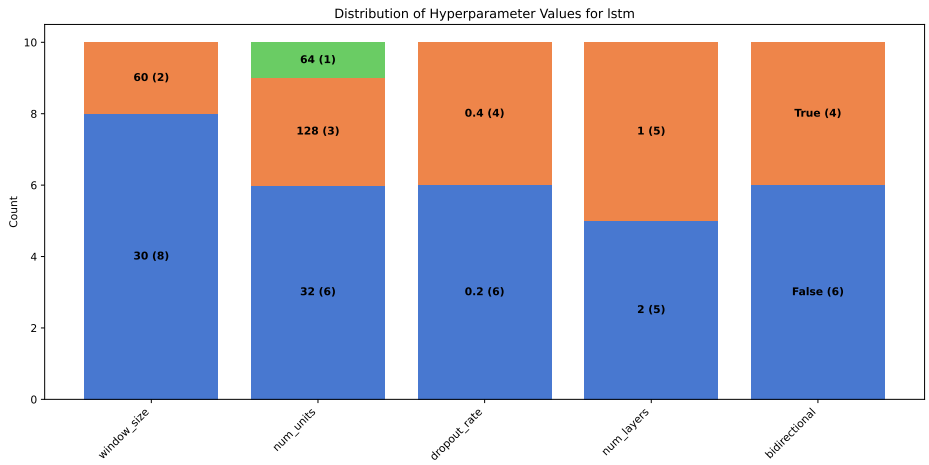


4.2.3 LSTM

Table 4.5: Parameter ranges used in the grid search for tuning the LSTM model

Parameter	Value
arch	['lstm']
window_size	[30, 60]
forecast_horizon	[1, 10]
num_units	[32, 64, 128]
dropout_rate	[0.2, 0.4]
num_layers	[1, 2]
bidirectional	[False, True]
activation	['tanh']
optimizer	['adam']
loss	['binary_crossentropy']
batch_size	[32]
epochs	[20]

Figure 4.4: Parameter Frequencies Among Top 10 Grid Search Results for the LSTM Model



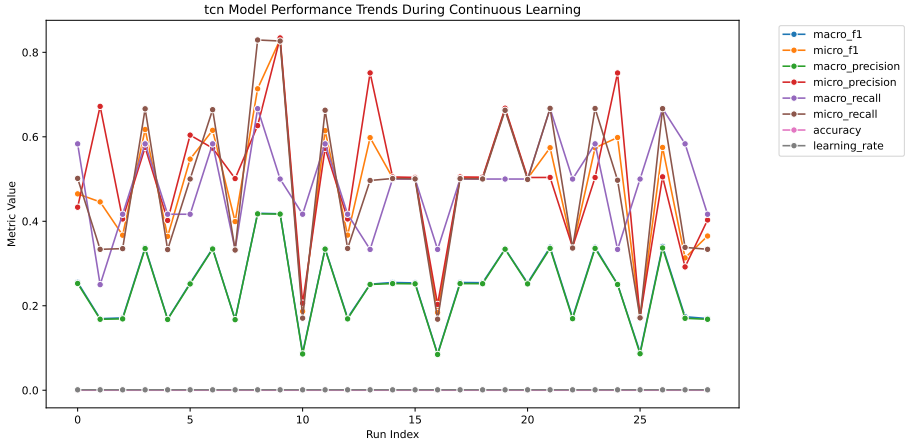
4.3 Model Results

4.3.1 TCN model

Table 4.6: Best performing hyperparameter configuration identified for the TCN model during grid search

Parameter	Value
window_size	30
forecast_horizon	144
num_filters	32
kernel_size	2
dropout_rate	0.3
activation	elu
dilations	[1. 2. 4. 8. 16. 32]

Figure 4.5: Changes in TCN model performance metrics over successive batches during continuous learning

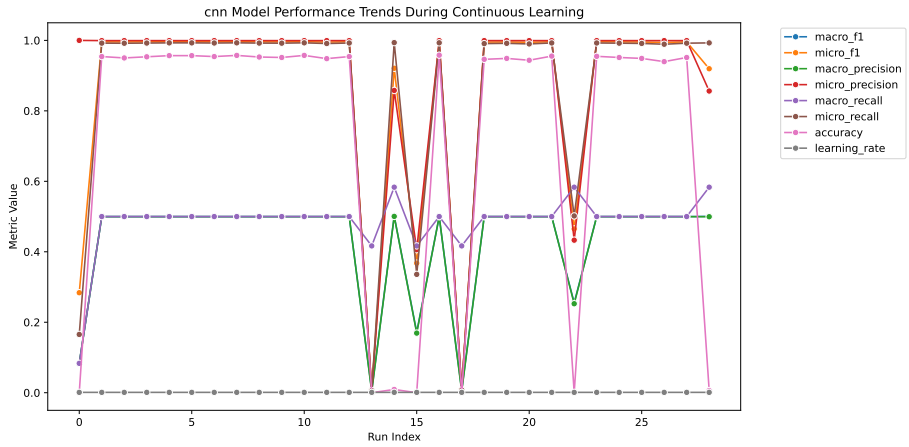


4.3.2 CNN model

Table 4.7: Best performing hyperparameter configuration identified for the CNN model during grid search

Parameter	Value
window_size	30
num_conv_layers	3
num_filters	32
kernel_size	2
dropout_rate	0.1
activation	relu
use_batch_norm	True
use_global_pooling	True
version	V1
batch_size	32
epochs	10

Figure 4.6: Changes in CNN model performance metrics over successive batches during continuous learning

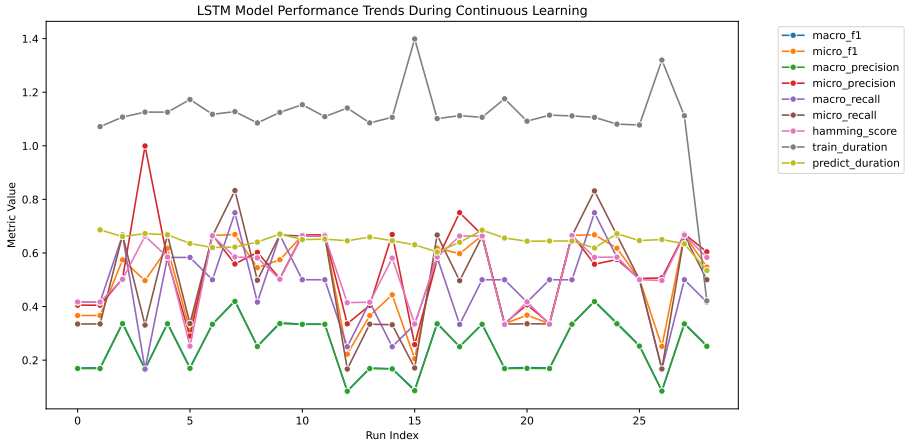


4.3.3 LSTM model

Table 4.8: Best performing hyperparameter configuration identified for the LSTM model during grid search

Parameter	Value
arch	lstm
window_size	60
forecast_horizon	10
num_units	128
dropout_rate	0.4
num_layers	2
bidirectional	True
activation	tanh
optimizer	adam
loss	binary_crossentropy
batch_size	32
epochs	20

Figure 4.7: Changes in LSTM model performance metrics over successive batches during continuous learning



4.4 Model Comparison

Table 4.9: Comparison of execution times (in seconds) for data preparation and model training across 1-day, 5-day, and 1-month datasets.

Size / Models	1 day	5 days	1 month
Data preparation	0.21 ± 0.006	1.78 ± 0.024	30.13 ± 0.3
TCN	0.12 ± 0.005	0.41 ± 0.006	2.34 ± 0.02
CNN	0.09 ± 0.004	0.39 ± 0.007	2.34 ± 0.01
LSTM	0.09 ± 0.004	0.39 ± 0.005	2.33 ± 0.02

Figure 4.8: Overall Metric results by model

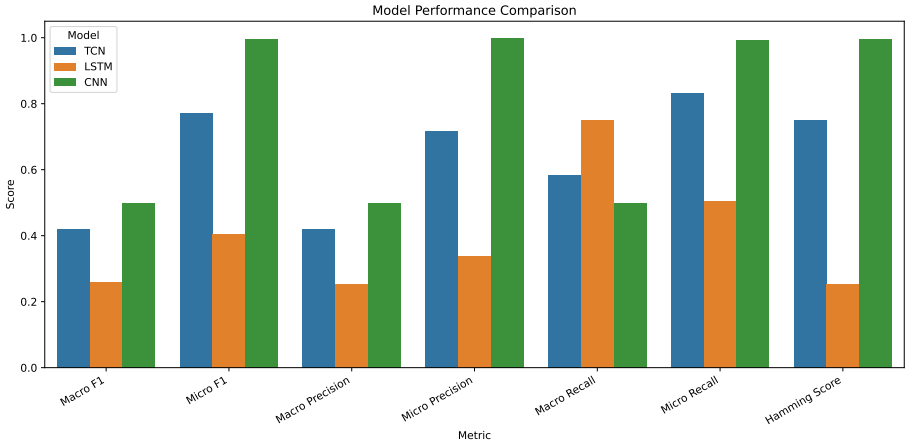


Figure 4.9: Label level Metric results by model

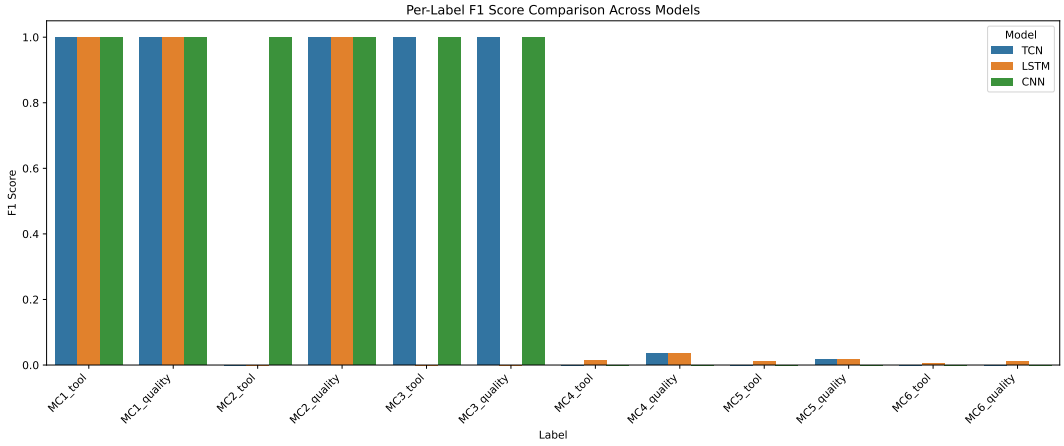


Figure 4.10: Models predictions frequency

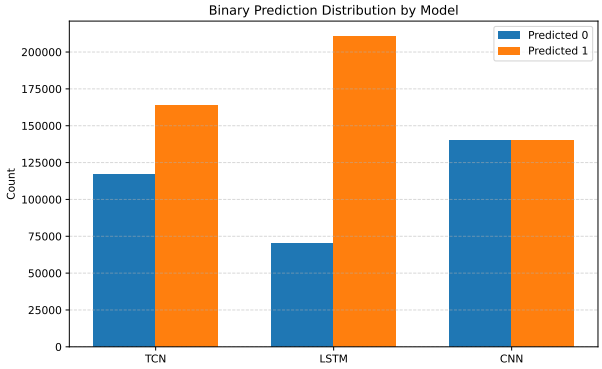


Figure 4.11: Micro F1 score during cont laerning model comparition

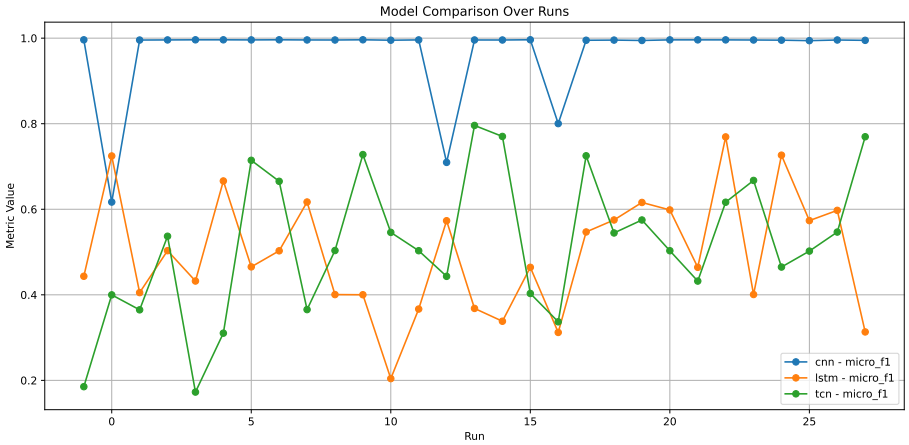
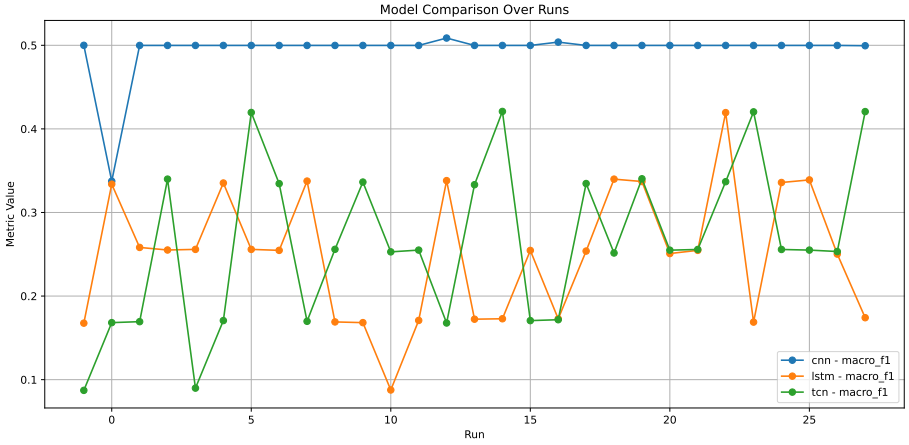


Figure 4.12: Macro F1 score during cont laerning model comparition



Chapter 5

Conclusion

This chapter presents the final conclusions of the study. Section 5.1 reviews the use of simulated data and its limitations. Section 5.2 summarizes model performance across tasks and machines, including results from continuous learning. and reflects on the study's limitations Section 5.3 outlines future directions, such as using real-world data and exploring alternative model architectures.

5.1 Data

5.1.1 Simulation

As outlined in section 3.2, a simulation was developed to generate the necessary MC signals for this study, addressing the lack of historical data. This chapter presents conclusions drawn from the simulation, considering observed production line behavior.

Scania has other ongoing projects focused on simulating different production lines. These projects primarily emphasize automation and maintenance-related delays and estimate throughput by assuming a constant efficiency factor. For instance, the expected output of a line is typically assumed to be 70% or 80% of its maximum theoretical throughput. However, the complete omission of MC signal-level detail in such simulations rendered them unsuitable for the specific requirements of this study.

The simulation performs well in capturing the atomic behavior of different entities and the dependencies between MCs and automation systems. It models machine-level tasks with clear execution boundaries and reflects the general priority of actions—starting with MCs, followed by automation, and

finally operator tasks. This structure aligns with the physical layout and logical flow of the production line. Despite these strengths, several limitations remain, particularly concerning the assumed readiness and responsiveness of the automation system and the operator. For instance, the simulation assumes that automation is always ready to perform loading and unloading actions, consistent with Scania's interpretation of automation capabilities.

In reality, automation systems require time to transition between tasks on different MCs. For example, if automation performs a part change on MC 1 and a part change request arises at MC 4 during that period, it must complete the task at MC 1 and then travel to begin the task at MC 4. This inter-task transportation time is not represented in the current simulation. Depending on the difference in CT across MCs, these delays may or may not have a noticeable impact on line throughput.

A similar simplification applies to operators. Once a task is completed, the simulation assumes the operator can immediately proceed to the next one. In practice, operators must first determine which task to perform next and may need to travel between task locations, introducing delays not currently modeled. Additionally, the simulation does not consider prioritization logic from the operator's perspective, such as deciding between multiple pending tasks. These assumptions about readiness and availability lead to a simplified view of automation and operator responsiveness, even though the simulation accurately models core task execution.

As previously discussed in section 3.2, the investigation of historical data unexpectedly revealed seasonal patterns in throughput. For example, tool breakage occurred more frequently during winter months, which in some cases caused tool life counters to reset earlier than expected. As a preventive measure, some critical MCs operated with shortened tool life intervals during colder periods to minimize tool-breakage-related downtime. These seasonal effects are not currently modeled in the simulation.

These limitations indicate potential areas for future research and simulation refinement. Incorporating automation transition times, modeling operator prioritization logic, and simulating seasonal variation in tool behavior could improve the realism and predictive value of the simulation.

5.2 Model comparison

5.2.1 Performance on Test Dataset

The results of this study do not indicate a single model that consistently outperforms the others. While model performance was relatively similar for the first MC, performance degraded on subsequent MCs downstream in the production line. This degradation is likely due to the internal dependencies between MCs, which reduce the model's ability to learn generalizable patterns across the full production line. As shown in Table 4.2, many target labels occur infrequently, limiting the learning signal available for supervised training. Despite these general challenges, some architectural characteristics influenced performance on individual MCs, for example, the behavior of the LSTM model.

Among the models, only the LSTM demonstrated measurable performance on MC6, the final machine in the line, based on evaluation metrics in Figure 4.9. Specifically, it achieved scores of 0.0113 for the quality task and 0.0038 for the tool task, whereas both CNN and TCN produced zero for these labels. While these values are low and not practically useful on their own, they nonetheless indicate that the LSTM model demonstrated limited predictive capacity, unlike the other models, which failed to learn any discernible pattern for these labels. This aligns with the architectural design of recurrent networks like LSTM which is capable of retaining long-term dependencies, an advantage in scenarios with sparse or delayed signal patterns. To further investigate model-specific tendencies, the distribution of predicted operator tasks was analyzed in Figure 4.10.

Figure 4.10 shows the distribution of positive and negative predictions for each model, where a positive prediction indicates the occurrence of an operator task (e.g., tool replacement or quality check). Given the inherent rarity of these tasks, as shown in table 4.2, an unbiased model would be expected to produce predominantly negative predictions. Despite this, the LSTM model made more frequent positive predictions (i.e., task occurrences) than the other models. The TCN showed a similar tendency, with slightly more negative predictions than the LSTM, though the majority of its predictions were still positive. The CNN, in contrast, produced the highest number of negative predictions among the models. However, its positive and negative predictions were numerically equal, indicating a significant tendency to over-predict rare events and generate a high number of false positives.

These predictive tendencies directly influenced overall performance. As

illustrated in Figure 4.11 and Figure 4.12, CNN achieved the highest F1 scores among the models. This result warrants further analysis to understand its underlying causes. Figure 4.8 shows that CNN also yielded the highest Hamming score, which reflects lower accuracy across all labels. This implies that the model may have been overly sensitive to rare events, leading to a higher number of false positives—even though it achieved strong precision and recall.

5.2.2 Evaluation of Models Under Continuous Learning

The results from the continuous learning evaluation indicate that both CNN and LSTM demonstrated slight improvements in performance as they were incrementally retrained with new data. Although no model showed a dramatic increase, the trend suggests that more data contributes to better generalization, though the effect remains limited. In contrast, the TCN exhibited a slight decrease in performance over time. The performance trajectories of each model are shown in fig. 4.6, fig. 4.5, and fig. 4.7.

However, the rate of improvement was limited. This can be attributed to the low density of learnable patterns in the dataset. The rarity of operator tasks and the interdependencies between MCs reduce the number of independent training examples available to the model. As a result, while the continuous learning setup reflects a realistic deployment scenario, it also highlights the challenge of training effective models under sparse and interdependent conditions.

While overall improvements were observed, certain data batches negatively affected model performance. These fluctuations indicate the presence of edge cases or shifts in the underlying data distribution that hinder model generalization. For instance, the TCN exhibited significant performance drops on the 10th and 14th batches, and again around the 25th, and was unable to regain its earlier performance. This decline is likely due to the TCN's sensitivity to abrupt changes, as it is designed to capture localized temporal patterns.

In contrast to the TCN's instability, the LSTM model exhibited greater robustness. Despite being exposed to the same data batches, the LSTM model showed greater resilience, maintaining relatively stable performance throughout the continuous learning process. Minor performance declines were still observed, though they were less pronounced than in the CNN and TCN models.

This behavior may be attributed to the LSTM's ability to retain longer

temporal dependencies, making it better suited to handle subtle changes in input distribution. These findings reinforce the conclusion that while continuous learning can yield incremental benefits, maintaining performance over time depends largely on a model's robustness to changes in data distribution.

5.2.3 Overall Comparison and Discussion

In summary, model performance was highly dependent on both task frequency and machine position in the production line. No architecture outperformed the others consistently across all metrics or machines. CNN demonstrated high precision and recall but struggled with over-prediction. LSTM showed strength in modeling long-range dependencies, particularly on later-stage machines. TCN, while consistent, lacked the adaptability observed in the other two models.

Continuous learning results suggest that model performance can improve with more data, but the effect is constrained by data sparsity and task imbalance. These findings reinforce the importance of not only architectural choices but also dataset design and task framing in industrial machine learning applications.

5.2.4 Limitations and Implications

Despite the insights gained, this study has several limitations. Prediction performance, particularly on the final MC, was low in absolute terms, with some models failing to produce any correct predictions for certain tasks. This highlights the difficulty of modeling MC behavior under data imbalance and limited tracked signals.

Furthermore, all data in this study was generated through simulation, using real-world configuration parameters collected through time studies of different operator tasks. However, simulations do not capture the uncertainty and variability present in real-world production environments, as discussed in section 5.1.1.

The simulated signals were designed to reflect the limited set of signals that are actually tracked in the production environment. In the real system, the only available automation signal is the part number, which remains constant during production and only changes when the product type changes. This results in zero information entropy, as calculated using eq. (2.12), and does not provide any meaningful input for learning time-dependent production behavior.

Despite this, the automation system plays a critical role in coordinating the production flow and linking the behavior of different MCs, as described in Section 3.1.2. The absence of richer automation signals therefore constrains the model's ability to capture MC to MC dependencies within the production line.

These constraints underscore the need for more adaptable model architectures and domain-aware input features. In particular, addressing signal limitations and MC-specific behavior requires models that are better aligned with the structure and complexity of the production environment. These challenges open several promising directions discussed in section 5.3.

5.3 Future work

The applied ML approach can be extended from a single-model to a multi-model system to improve prediction performance. One approach is to split the problem by task type, using separate, specialized models for different operator tasks such as quality checks or tool replacements. Alternatively, the problem can be approached by treating each MC as an individual unit, training a dedicated model for its specific operator interactions. This allows each architecture to be precisely optimized for the MC it serves. To capture dependencies across MCs, prediction outputs from upstream models could be passed as features into downstream ones, enabling task forecasting that reflects the full production flow.

Beyond architectural extensions, alternative ML paradigms can also be explored. The interdependencies between MCs suggest a graph-like structure, making Graph Neural Networks (GNNs) a promising choice for capturing the dynamics of the production line. Additionally, reinforcement learning offers a promising framework for learning task scheduling policies based on throughput-oriented reward signals. Instead of directly predicting operator tasks, a reinforcement learning model could learn to schedule them based on expected hourly throughput as a reward signal, gradually optimizing its policy through interaction with the simulated environment.

Chapter 6

Appendix

here appendix lives

Bibliography

[1] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*. doi: 10.1109/HICSS.2016.488 pp. 3928–3937, ISSN: 1530-1605. [Online]. Available: https://ieeexplore.ieee.org/document/7427673/?arnumber=7427673 [Page 1.]

[2] S. G. McCrady, *1. Introduction*. ISBN 978-0-12-417000-1. [Online]. Available: https://learning.oreilly.com/library/view/designing-scada-application/9780124170001/xhtml/CHP001.html [Page 2.]

[3] K. Vasilko and Z. Murčinková, "Reduction in total production cycle time by the tool holder for the automated cutting insert quick exchange and by the double cutting tool holder," vol. 7, no. 3, p. 99. doi: 10.3390/jmmp7030099 Number: 3 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2504-4494/7/3/99 [Page 8.]

[4] L. Delolme, A.-L. Antomarchi, S. Durieux, and E. Duc, "Decision-making for multi-criteria optimization of process planning," vol. 20, no. 8, p. 806. doi: 10.1051/meca/2020040 Number: 8 Publisher: EDP Sciences. [Online]. Available: https://www.mechanics-industry.org/articles/meca/abs/2019/08/mi190304/mi190304.html [Pages 8 and 9.]

[5] W. Babel, "Automation pyramid and solutions business," in *Industry 4.0, China 2025, IoT*. Springer, Wiesbaden, pp. 75–147. ISBN 978-3-658-37852-3. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-658-37852-3_4 [Pages 9 and 10.]

[6] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, "Chapter 5 - credibility: Evaluating what's been learned," in *Data Mining (Fourth Edition)*, I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, Eds. Morgan Kaufmann, pp. 161–203. ISBN 978-0-12-804291-5. [Online]. Available: https:

//www.sciencedirect.com/science/article/pii/B9780128042915000052 [Pages 11, 12, 13, 14, 15, and 28.]

[7] K. Keller, *Entropy Measures for Data Analysis: Theory, Algorithms and Applications*. MDPI - Multidisciplinary Digital Publishing Institute. ISBN 978-3-03928-032-2 978-3-03928-033-9 [Page 16.]

[8] P. Bruce and A. Bruce, *Practical Statistics for Data Scientists*. O'Reilly Media Inc. ISBN 978-1-4919-5295-5. [Online]. Available: https://learning.oreilly.com/library/view/practical-statistics-for/9781491952955/ [Pages 16 and 17.]

[9] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." [Online]. Available: http://arxiv.org/abs/1803.01271 [Page 17.]

[10] Y. Wang, Z. Liu, D. Hu, and M. Zhang, "Multivariate time series prediction based on optimized temporal convolutional networks with stacked auto-encoders," in *Proceedings of The Eleventh Asian Conference on Machine Learning*. PMLR, pp. 157–172, ISSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v101/wang19c.html [Pages 17 and 19.]

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi: 10.1109/CVPR.2016.90 pp. 770–778, ISSN: 1063-6919. [Online]. Available: https://ieeexplore.ieee.org/document/7780459/ [Page 18.]

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift." [Online]. Available: http://arxiv.org/abs/1502.03167 [Page 20.]

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," vol. 15, no. 56, pp. 1929–1958. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html [Page 20.]

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," vol. 86, no. 11, pp. 2278–2324. doi: 10.1109/5.726791. [Online]. Available: https://ieeexplore.ieee.org/document/726791/ [Page 20.]

[15] 3.1. cross-validation: evaluating estimator performance. [Online]. Available: https://scikit-learn/stable/modules/cross_validation.html [Page 28.]

[16] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice (3rd ed)*. [Online]. Available: https://otexts.com/fpp3/ [Page 28.]

[17] tfa.metrics.HammingLoss | TensorFlow addons. [Online]. Available: https://www.tensorflow.org/addons/api_docs/python/tfa/metrics/HammingLoss [Page 33.]