**UNIVERSITY OF MISKOLC**
**FACULTY OF MECHANICAL ENGINEERING AND INFORMATICS**


# IMPROVING THE EFFICIENCY OF MAINTENANCE PROCESS IN MANUFACTURING SYSTEMS USING INDUSTRY 4.0 TOOLS

**PH.D. DISSERTATION**

**Prepared by:**
Marc Philip Hermans
MSc in Mechanical Engineering

**JÓZSEF HATVANY DOCTORAL SCHOOL**
**FOR COMPUTER SCIENCE AND ENGINEERING**
**Institute of Logistics**

**Head of Doctoral School:**
Prof. Dr. habil. Kovács László
University Professor

**Head of Topic Group:**
Prof. Dr. habil. Illés Béla
Professor Emeritus

**Scientific Supervisor:**
Prof. Dr. habil Tamás Péter
Director of the Institute, University Professor

Miskolc
2026

# DECLARATION

I, the undersigned, hereby declare that I have prepared this doctoral dissertation independently and have used only the sources specified therein. Any parts that have been taken, either verbatim or in content, from other sources—even if paraphrased—have been clearly indicated with appropriate references. The reviews of the dissertation and the official record of the defense will be made available at a later date at the Dean's Office of the University of Miskolc.

Miskolc, April 8, 2026

Hermans Marc Philip

# Use of Artificial Intelligence Tools

Artificial intelligence tools (specifically large language models) were used during the preparation of this dissertation to support language refinement, structural organization, and the clarification of explanations.

In addition, AI-assisted coding practices—sometimes referred to as "vibe coding"—were employed during the development of selected software components and data processing pipelines. In this context, AI tools were used to accelerate prototyping, suggest implementation patterns, and improve code readability. All generated code was critically reviewed, validated, and, where necessary, adapted by the author to ensure correctness, consistency, and alignment with the research objectives.

All scientific content, including the development of concepts, models, and conclusions, is the original work of the author. AI tools were used solely as assistive instruments and did not replace independent scientific reasoning, analysis, or interpretation.

The author takes full responsibility for the content, implementation, and integrity of this dissertation.

Miskolc, April 8, 2026

Hermans Marc Philip

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all those who, through their support and encouragement, contributed to the completion of this dissertation.

This dissertation was prepared at the Institute of Logistics of the University of Miskolc within the framework of the Doctoral School of Computer Science of the Hatvany József Informatics Sciences. First and foremost, I would like to express my deepest appreciation to Prof. Dr. Tamás Péter, my scientific supervisor, whose professional guidance, helpful attitude, and continuous support greatly contributed to the successful completion of my work.

I would also like to thank Prof. Dr. László Kovács and Prof. Dr. Jenő Szigeti, who, as leaders of the Doctoral School of Computer Science of the Hatvany József Informatics Sciences, continuously supported my work with valuable advice and professional guidance.

My thanks are extended to all colleagues of the Institute of Logistics at the University of Miskolc, whose professional and moral support made the completion of this dissertation possible.

I am grateful to Emese Homonnai, staff member of the Dean's Office of the Faculty of Mechanical Engineering and Informatics, as well as to Dr. János Juhász, for their assistance in handling all administrative matters throughout my doctoral studies and research.

Finally, I would like to express my sincere gratitude to my wife Adrienn, my children, my parents—particularly my father, Hermans John Dominique, a retired professor, for his invaluable proofreading and his guidance in mathematical matters, especially regarding Laplace transformations—my sister, and my friends for their patience, encouragement, and unwavering support throughout this journey.

# Supervisor's Recommendation

Ensuring efficient and sustainable operation of modern manufacturing systems requires a sound scientific basis for maintenance decision-making, particularly in the context of Industry 4.0. The present doctoral dissertation provides a novel and well-founded contribution to this challenge.

The candidate addresses a complex and highly relevant industrial problem by shifting maintenance decision-making from deterministic approaches toward a probabilistic framework. A key strength of the work is its integrated approach, combining Overall Equipment Effectiveness (OEE), degradation modeling, and Bayesian decision logic into a coherent methodology.

The scientific contributions are significant. The dissertation reinterprets OEE as a probabilistic performance indicator capturing system degradation dynamics. Furthermore, it introduces a Weibull-based, sawtooth-shaped hazard model that reflects real industrial processes. The proposed Bayesian neural network framework enables risk-based maintenance decisions with explicit treatment of uncertainty.

The research is methodologically sound and validated through discrete-event simulation, ensuring reproducibility and scientific reliability. An additional advantage is the limited data requirement, allowing application even in environments with minimal sensor infrastructure.

The candidate conducted the research independently and at a high professional level, with results published in reputable scientific venues. The dissertation is well-structured, logically developed, and supported by rigorous scientific reasoning.

In conclusion, the dissertation meets the scientific requirements, presents novel and valuable results, and contributes meaningfully to maintenance logistics and Industry 4.0-based decision support. I recommend the dissertation for public defense and support the awarding of the PhD degree.

Miskolc, 08 April 2026

Prof. Dr. Tamás Péter
University Professor
Head of Institute
Supervisor

# Contents

# 1 Economic and Scientific Relevance

Predictive maintenance in serial production is receiving increasing economic and scientific attention due to global disruptions such as the energy crisis, climate change, and supply chain volatility. These developments highlight the need for maintenance strategies that not only improve operational reliability but also support data-driven decision-making under uncertainty. These developments intensify the pressure on manufacturing companies to improve operational efficiency while simultaneously reducing energy consumption and material waste. Consequently, maintenance strategies have become a critical factor in achieving stable and sustainable production systems.

Maintenance has long been recognized as a cornerstone of manufacturing reliability, yet its strategic potential remains underutilized in many industrial settings. Historically, maintenance activities have been predominantly reactive—performed only after failures occur—or preventive, based on predefined time intervals rather than actual system conditions. Although preventive maintenance reduces unexpected breakdowns, fixed interval scheduling often leads either to premature maintenance actions or to insufficient protection against failures.

Predictive maintenance is frequently proposed as the solution to this challenge. However, its implementation is far from trivial. Predictive maintenance depends heavily on extensive sensor infrastructure, sophisticated data acquisition systems, and advanced analytics methods such as machine learning and signal processing. Beyond hardware investments, these systems require continuous retraining of personnel and adaptation to rapidly evolving analytical technologies. These barriers render predictive maintenance economically and organizationally challenging for many small and medium-sized enterprises. Even large manufacturers operating legacy equipment may find such systems difficult to deploy at scale [1].

In response to these limitations, this dissertation explores an alternative direction: leveraging simulation and probabilistic modeling—particularly Bayesian methods—to enable predictive maintenance decision-making without relying on extensive sensor infrastructures. The proposed approach integrates production performance indicators, degradation modelling, and probabilistic inference into a unified analytical framework. This approach facilitates broader applicability, including older or partially observable produc-

tion lines, while still enabling uncertainty-aware maintenance decision support.

Industry 4.0 technologies such as Discrete Event Simulation (DES) and probabilistic machine learning methods provide the technological foundation for such an approach. These tools enable the modelling of degradation processes, the reconstruction of production performance indicators from operational data, and the estimation of probabilistic maintenance decisions. These tools enable probabilistic and simulation-driven maintenance decision-making that moves beyond rigid time-based policies toward dynamic, reliability-centered maintenance strategies. Improved maintenance planning can significantly reduce unplanned downtime and therefore contribute to improved energy efficiency and waste reduction in manufacturing systems [2].

For example, a production line consisting of ten machines that reduces unplanned downtime from 300 to 210 hours per year avoids approximately 7,200 kWh of wasted energy. At an electricity cost of 0.15 €/kWh, this corresponds to savings of roughly 1,080 € annually. In addition, improved production stability can reduce scrap rates from 5% to 3%, which corresponds to a reduction of approximately 40% in scrap volume in high-volume production environments.

Recent global disruptions, including the energy crisis triggered by geopolitical conflict and the increasing urgency of climate change, have intensified the demand for operational efficiency and sustainability. Within this context, maintenance logistics plays a pivotal role. Minimizing unplanned downtime not only safeguards productivity but also contributes directly to energy efficiency and waste reduction.

The core motivation of this dissertation is therefore to bridge the gap between traditional preventive maintenance and predictive maintenance approaches by integrating production performance measurement, reliability theory, simulation-based modeling, and probabilistic machine learning into a unified analytical framework.The detailed analysis of existing research and the resulting research gap are presented in Chapter 2.

This research is guided by the following fundamental scientific question:

> To what extent can maintenance decisions in serial production systems be supported by probabilistic models that estimate the risk of performance degradation based on production data?

Addressing this question requires the integration of multiple research fields, including production performance measurement, reliability engineering, and probabilistic machine learning. The literature review presented in Chapter 2 examines existing research on Overall Equipment Effectiveness (OEE), Total Productive Maintenance (TPM), reliability modelling, and Digital Twin technologies in order to identify the research gap motivating the probabilistic maintenance framework developed in this dissertation. The explicit formulation of this gap is provided in Section 2.6.

## Research Question and Sub-Questions

The overall aim of this dissertation is to investigate how probabilistic methods, simulation modeling, and Bayesian inference can be integrated into maintenance logistics in order to support maintenance decision-making in serial production environments.

**Central Research Question**

> How can Overall Equipment Effectiveness (OEE), degradation modelling based on reliability theory, and Bayesian probabilistic inference be integrated into a coherent framework that supports maintenance decision-making in serial production systems?

**Sub-Questions**

1. How can OEE be redefined and calculated in a way that is consistent, comparable across production systems, and relevant for maintenance decision-making?
2. What are the limitations of traditional hazard models (such as the Negative Exponential distribution) for describing failure behavior in production systems, and how can alternative approaches such as Weibull-based or sawtooth hazard models provide a more realistic representation of degradation dynamics?
3. How can Discrete Event Simulation (DES) be used to generate synthetic production data and evaluate maintenance strategies under controlled and reproducible operating conditions?
4. How can Bayesian Neural Networks incorporate uncertainty into maintenance planning and estimate the probability that maintenance can be safely postponed without violating production performance thresholds?

The methodological framework used to address these questions—combining timestamp-based OEE reconstruction, degradation modelling using Weibull-based hazard functions, discrete-event simulation, and Bayesian probabilistic inference—is introduced in Chapter 3. Subsequent chapters develop the theoretical models, simulation environment, and probabilistic decision framework that together form the basis of the proposed maintenance decision-support approach.

# 2 Literature Review and Research Aims

## 2.1 Conceptual Foundations

To ensure clarity and consistency, the key concepts used throughout this research are defined explicitly and positioned within the relevant scientific literature.

### 2.1.1 Maintenance Logistics

Maintenance logistics refers to the planning and coordination of maintenance activities, including scheduling, resource allocation, spare parts provisioning, and technician deployment. The concept has been discussed in the context of maintenance management and service logistics, where it emphasizes the efficient organization of support activities for technical systems [3].

In contrast to traditional maintenance approaches, this research interprets maintenance logistics as an integrated decision-making process that combines operational and logistical considerations under uncertainty, thereby extending classical planning-focused definitions.

### 2.1.2 Probabilistic Interpretation of OEE

Overall Equipment Effectiveness (OEE), originally introduced within the Total Productive Maintenance (TPM) framework, is traditionally defined as a deterministic performance indicator composed of availability, performance, and quality [4, 5].

In this research, OEE is reinterpreted as a time-based stochastic signal derived from production timestamps, enabling its use as a probabilistic descriptor of system performance rather than a purely retrospective KPI.

Two forms are distinguished:

- **Prior OEE:** Expected performance before the machine, assuming ideal conditions,
- **Posterior OEE:** Observed performance after the machine, reflecting actual system behaviour.

The deviation between prior and posterior OEE serves as an indicator of degradation and forms a key input to the Bayesian decision framework. This distinction introduces a probabilistic interpretation that departs from classical OEE formulations and enables integration into predictive decision-making models.

### 2.1.3 Bayesian Decision Rule

The maintenance decision is formulated as a probabilistic rule based on the likelihood that system performance will fall below a predefined threshold, following principles of Bayesian decision theory [6, 7, 8]. The probability represents a predictive posterior conditioned on observed system evidence and the absence of maintenance action.

$$P(OEE_{t+1} < OEE_{\text{planned}} \mid \text{evidence}, \neg M_t) \tag{2.1}$$

Maintenance is performed if this probability exceeds a predefined confidence threshold; otherwise, it is postponed. This formulation aligns maintenance planning with risk-based decision-making under uncertainty, rather than fixed-interval scheduling.

### 2.1.4 Role of the Digital Twin

The discrete-event simulation model acts as a *generative digital twin*, producing synthetic data that reflects system behaviour under controlled conditions. The digital twin concept has been widely defined as a virtual representation of a physical system used for monitoring, simulation, and optimization [9, 10].

Unlike fully integrated cyber-physical twins, the focus here is on data generation and scenario exploration, positioning the digital twin as an experimental and analytical instrument rather than a real-time synchronization mechanism.

### 2.1.5 System State vs. Maintenance Synchronization

In contrast to traditional condition monitoring and prognostics approaches, which aim to estimate the physical degradation state or remaining useful life of a system [11, 12], the Bayesian Neural Network does not directly estimate the physical state of the system. Instead, it evaluates the degree of synchronization between system degradation and maintenance actions.

This distinction is critical: the model does not predict “how damaged” a system is, but rather whether the current maintenance strategy remains aligned with the underlying degradation dynamics. This represents a shift from state estimation toward policy alignment, forming a central conceptual contribution of this research.

## 2.2 Rethinking TPM: A New Perspective on Maintenance Efficiency

In Arno Koch's book, *OEE for the production team* [5], Total Productive Maintenance (TPM) is described as a form of time loss resulting from maintenance activities during which machines remain available but are not producing. In contrast, numerous theses and research works [13, 14, 15], as well as industrial practice, interpret TPM as a mechanism that transforms unpredictable failures into more controllable and partially predictable events. From a production efficiency perspective, these two interpretations appear contradictory and highlight the fundamental tension between short-term production output and long-term system reliability. While minimizing maintenance activities can temporarily increase production efficiency, it simultaneously increases exposure to unexpected production losses.

To reconcile this dilemma, companies often attempt to transition from preventive maintenance toward predictive maintenance strategies. Although predictive maintenance can effectively prevent equipment failures, it frequently fails to provide sufficient advance planning time for maintenance preparation in serial production environments with very short cycle times.

As illustrated in Figure 2.1, predictive maintenance requires extensive sensor infrastructure and complex data analysis procedures. These systems rely on advanced analytics and domain expertise that may exceed the capabilities or interest of maintenance personnel.

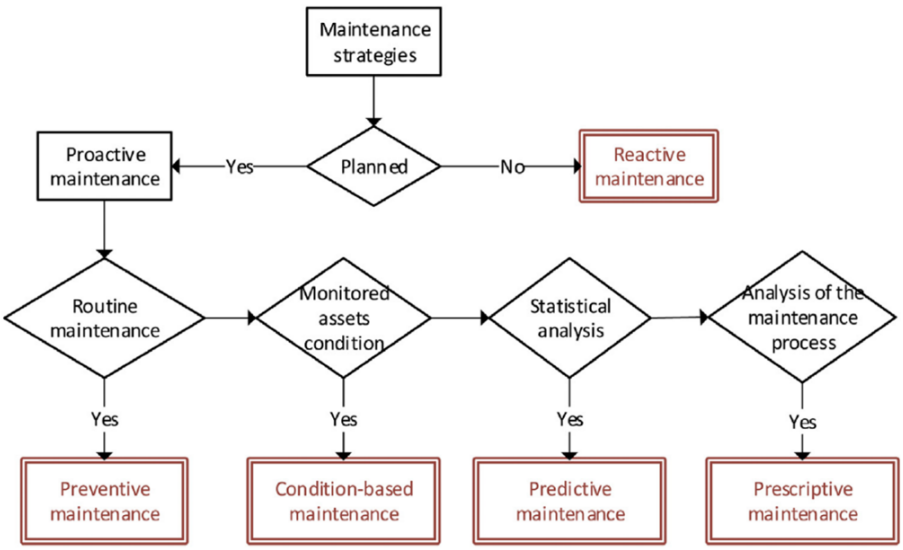



Figure 2.1: Maintenance strategies diagram.

Rather than simply replacing preventive maintenance with predictive maintenance, this

research investigates an intermediate concept in which preventive maintenance strategies evolve toward probabilistic decision-making based on operational performance indicators. In this context, the concept of an *executable digital twin* becomes particularly relevant. By using a digital twin not only as a simulation or planning tool but as a real-time monitoring and advisory system, maintenance decisions can be supported using operational performance indicators [16].

To establish such a framework, three scientific questions must be investigated:

1. **Reconstruction of Production Performance Indicators**

   The first research focus investigates whether Overall Equipment Effectiveness (OEE) can be reconstructed objectively from production timestamp data in a way that enables consistent comparison across production systems.

2. **Modelling Degradation in Maintained Production Systems**

   The second research focus examines how degradation processes in periodically maintained production systems can be represented using reliability models, particularly hazard rate functions derived from Weibull distributions.

3. **Probabilistic Maintenance Decision Framework**

   The third research focus investigates how probabilistic inference methods can be used to support maintenance decision-making by estimating the probability that production performance will fall below a predefined threshold.

Through the integration of these elements, the proposed framework seeks to combine performance measurement, maintenance strategy, and digital twin technologies into a unified decision-support approach for maintenance management.

The evolution of maintenance strategies can broadly be described as a progression from corrective maintenance (fail-and-fix), through preventive maintenance (scheduled interventions), to predictive maintenance (condition-based strategies). Despite considerable technological progress, predictive maintenance remains difficult to implement in many serial production environments.

Key barriers include the complexity of data acquisition, uncertainty in failure modeling, and the difficulty of integrating predictive maintenance outputs with production logistics.

A fundamental concept in reliability engineering is the hazard rate, which describes the probability of system failure over time. Hazard rates are commonly modeled using either the Negative Exponential distribution, which assumes a constant failure rate, or the Weibull distribution, which allows time-dependent degradation behavior.

The Weibull distribution is particularly suitable for modelling wear-out processes because its hazard rate allows increasing failure intensity when the shape parameter $\beta > 1$.

In this dissertation, a sawtooth-shaped hazard structure is used to represent degradation processes in periodically maintained systems, where the hazard rate increases between maintenance events and is partially reset after maintenance interventions. This approach enables a more realistic representation of maintenance-induced reliability dynamics [17].

In addition, Bayesian inference methods provide a probabilistic framework for estimating the likelihood and uncertainty of future system degradation based on production data. Bayesian Neural Networks (BNNs) are particularly well suited for maintenance decision support because they produce probabilistic predictions and explicit uncertainty estimates. Unlike conventional neural networks, BNNs quantify prediction confidence, which is essential in industrial decision environments where maintenance actions carry operational risk [18, 19].

## 2.3 Methodical Approach

This literature review investigates research on Overall Equipment Effectiveness (OEE), Total Productive Maintenance (TPM), and Digital Twin technologies using the ScienceDirect database. [20] These domains were selected because of their relevance for maintenance optimization and production system performance.

The decision to focus on ScienceDirect was based on several factors:

1. **Quality of Sources**

   ScienceDirect provides access to a large collection of peer-reviewed journals and scientific publications, ensuring reliable and high-quality literature sources.

2. **Historical Coverage**

   Understanding the development of OEE, TPM, and Digital Twin concepts requires access to literature spanning several decades. ScienceDirect offers extensive archival coverage for this purpose.

3. **Language Consistency**

   Only English-language publications were included to avoid introducing additional translation complexity into the research process.

Search queries were restricted to the subject areas of Engineering, Decision Sciences, Mathematics, and Computer Science. The following keyword groups were applied:

- **Overall Equipment Effectiveness (OEE)** [4, 21, 22]
- **Total Productive Maintenance (TPM)** [4, 23, 24]

- **Digital Twin Technologies (DT)** [25, 26, 27]

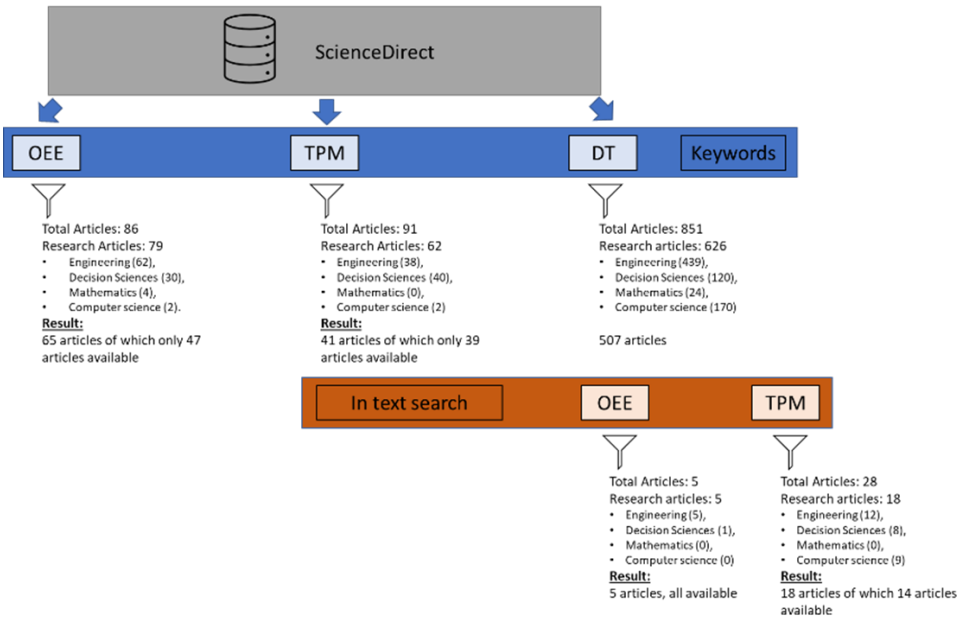



Figure 2.2: Filtering of articles across domains and subdomains.

## Search Results and Refinement

The initial search produced:

- OEE: 86 articles
- TPM: 91 articles
- Digital Twin: 851 articles

After applying additional filters and relevance criteria, 101 articles remained for detailed evaluation.

## Categorization and Scoring

Articles were classified and scored according to thematic relevance:

- Score 3 — Direct contribution to OEE calculation, TPM task mapping, or executable digital twin concepts.
- Score 2 — Indirect but significant relevance.
- Score 1 — Peripheral or weak connection to research topics.

| Keyword | Catagory | Nr of articles | Ranking |
|---|---|---|---|
| OEE | Calculating OEE | 14 | 3 |
| | OEE and Fuzzy | 1 | 2 |
| | OEE, TPM | 3 | 2 |
| | OEE and DES | 1 | 1 |
| | OEE and maintenance manufacturing | 1 | 1 |
| | OEE and maintenance status | 1 | 1 |
| | case study | 12 | 0 |
| | Digital Dashboard | 1 | 0 |
| | Improve the effectiveness | 2 | 0 |
| | Ind 4.0 and Improvement | 1 | 0 |
| | OEE and AM | 1 | 0 |
| | OEE and CO2 Emissions | 1 | 0 |
| | OEE and cyber security | 1 | 0 |
| | OEE and Energy | 1 | 0 |
| | OEE and flexibility | 1 | 0 |
| | OEE and FMEA | 1 | 0 |
| | OEE and Scheduling | 1 | 0 |
| | OEE and time margins | 1 | 0 |
| | Performance increase | 1 | 0 |
| | Transfer of knowledge | 1 | 0 |
| Total | | 47 | 21 |

| Keyword | Catagory | Nr of articles | Ranking |
|---|---|---|---|
| DT & OEE | DT and Bottleneck analyses | 1 | 0 |
| | case study | 2 | 0 |
| | DT and 5G | 1 | 0 |
| | DT and ICT | 1 | 0 |
| Total | | 5 | 0 |

| Keyword | Catagory | Nr of articles | Ranking |
|---|---|---|---|
| TPM | TPM standardizing | 4 | 3 |
| | TPM History | 1 | 1 |
| | TPM management | 1 | 1 |
| | Calculating OEE | 1 | 0 |
| | case study | 12 | 0 |
| | Implementing TPM | 1 | 0 |
| | Key_OEE | 10 | 0 |
| | TPM and CO2emissions | 2 | 0 |
| | TPM and I4.0 | 1 | 0 |
| | TPM and Tele maintenance | 1 | 0 |
| | TPM and TQM | 4 | 0 |
| | TPM problem identification | 1 | 0 |
| | TPM, SPC, TPM standardizing | 1 | 0 |
| Total | | 40 | 6 |

| Keyword | Catagory | Nr of articles | Ranking |
|---|---|---|---|
| DT & TPM | DT and maintenance | 2 | 1 |
| | case study | 4 | 0 |
| | DT and benefits | 1 | 0 |
| | DT and Jobshop | 1 | 0 |
| | DT and Sustainability | 1 | 0 |
| | DT Challenges | 1 | 0 |
| | DT cyber security | 1 | 0 |
| | DT Digital Thread | 1 | 0 |
| | DT general | 1 | 0 |
| | DT review | 1 | 0 |
| Total | | 14 | 2 |

Figure 2.3: Grading mechanism used for scoring selected articles.

## 2.4 Main Findings

After evaluation, 29 articles were selected for deeper analysis. The grouping and filtering process is illustrated in Figure 2.4.

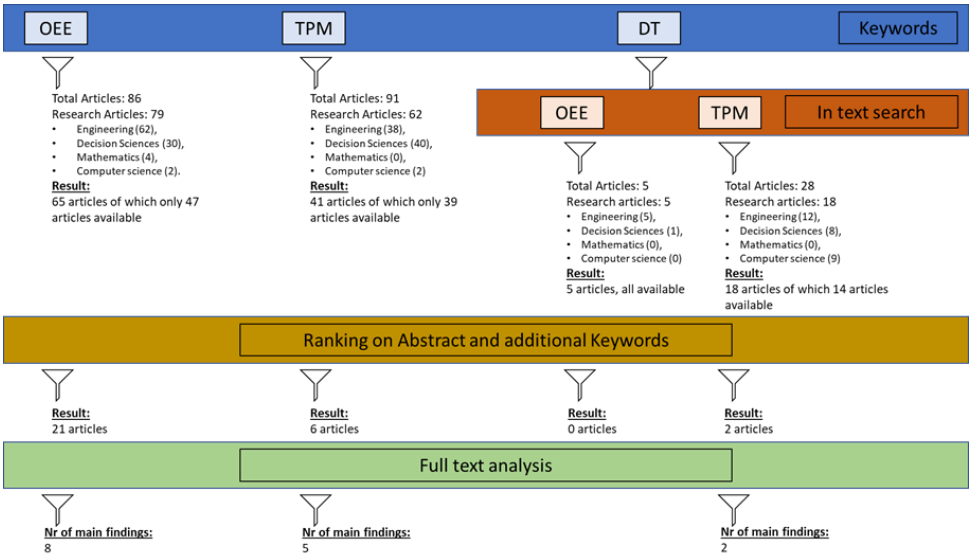



Figure 2.4: Article grouping by thematic topic.

The analysis revealed that OEE research focuses primarily on performance measurement and process optimization. TPM literature emphasizes maintenance organization and

operational improvement, while digital twin research mainly addresses monitoring and predictive analytics.

Although these domains share conceptual connections, the literature rarely integrates production performance indicators, degradation modelling, and probabilistic decision-making into a unified analytical framework.

## 2.5 Results and Discussion

The reviewed literature demonstrates that OEE calculations are highly sensitive to the definition and classification of production losses. Different methodologies produce different OEE values, highlighting the need for standardized calculation methods.

Consistency between real systems and digital twin models is essential to enable comparability, model validation, and reliable decision support.

Several studies indirectly support the integration of OEE, TPM, and digital twin technologies. However, no study explicitly connects OEE degradation with maintenance decision-making within a digital twin environment.

## 2.6 Research Gap and Positioning of the Thesis

The literature review reveals that the research areas of OEE, TPM, and Digital Twin technologies have evolved largely independently.

OEE research primarily focuses on performance measurement and operational evaluation, but rarely interprets OEE as a dynamic indicator of system deterioration.

TPM research concentrates on maintenance organization and implementation strategies but provides limited mechanisms for linking production performance indicators to maintenance timing.

Digital Twin research focuses mainly on monitoring, simulation, and predictive maintenance applications, but rarely incorporates production performance indicators such as OEE into maintenance decision frameworks.

As a consequence, existing research does not provide an integrated framework that combines probabilistic performance indicators, degradation modelling, and risk-aware maintenance decision-making.

This dissertation addresses this gap by developing a probabilistic maintenance decision framework that integrates OEE-based performance reconstruction, degradation modelling using hazard functions, discrete-event simulation, and Bayesian probabilistic inference.

The methodological framework used to operationalize this approach is presented in Chapter 3.

## 2.7 Positioning within Existing Literature

To clearly distinguish the contribution of this research from existing approaches, Table 2.1 summarizes the key differences between the state of the art and the proposed framework.

| **Aspect** | **State of the Art** | **Contribution of this Research** |
|---|---|---|
| Maintenance Modeling | Deterministic PM, CBM, or RUL-based prediction | Probabilistic decision-making based on confidence thresholds |
| Use of OEE | Retrospective KPI for performance monitoring | Time-based probabilistic signal with prior/-posterior interpretation |
| Machine Learning Output | Point estimates (e.g., RUL prediction) | Full predictive distributions using Bayesian Neural Networks |
| Handling of Uncertainty | Often implicit or ignored | Explicit modelling and propagation of uncertainty |
| Data Requirements | Sensor-intensive data collection | Minimal data approach supported by DES-generated synthetic data |
| Decision Logic | Threshold-based or heuristic rules | Bayesian decision rule based on probability of performance degradation |

Table 2.1: Comparison of Existing Approaches and Proposed Contribution

The comparison highlights that the primary novelty of this research lies not in isolated methodological elements, but in their integration into a unified probabilistic decision-support framework.

### 2.7.1 Research Design

This research follows a Design Science Research (DSR) paradigm, aiming to develop and evaluate an artifact that addresses a relevant industrial problem. The identified problem is the lack of decision-support mechanisms in maintenance planning that explicitly incorporate uncertainty.

The proposed artifact is a Bayesian Neural Network (BNN)-driven decision framework that estimates the probability of performance degradation and supports maintenance postponement decisions based on quantified confidence.

The research process consists of three stages:

1. **Problem formalization:** Maintenance is reframed as a probabilistic decision problem based on OEE degradation and uncertainty.

2. **Artifact development:** A simulation-based data generation pipeline (DES) and a BNN model are developed to estimate the probability of failure under postponed maintenance.

3. **Evaluation:** The artifact is evaluated using synthetic production scenarios, focusing on cost-risk trade-offs, maintenance regimes, and system stability.

The contribution of this research lies in integrating probabilistic modelling, simulation-based data generation, and maintenance decision logic into a unified framework that is both operationally interpretable and adaptable to industrial environments.

### 2.7.2 Scientific Contribution

This thesis contributes to the field of maintenance logistics and predictive maintenance in three key aspects:

- **Conceptual contribution:** Maintenance is reframed from a deterministic scheduling problem into a probabilistic confidence-based decision process, shifting the focus from failure prediction to the quantification of safe postponement.

- **Methodological contribution:** A novel integration of Discrete Event Simulation (DES) and Bayesian Neural Networks (BNNs) is proposed, enabling the generation of controlled degradation scenarios and the learning of probabilistic maintenance policies under uncertainty.

- **Operational contribution:** The proposed framework links probabilistic predictions directly to maintenance and logistics decisions, including intervention timing, spare parts provisioning, and resource allocation, thereby extending predictive maintenance from diagnosis to actionable decision support.

# 3 Research Overview, Aims, and Methodological Framework

Building on the research gap identified in Chapter 2 and formalized in Section 2.6, this chapter presents the overall research objectives and the methodological framework used to investigate probabilistic maintenance decision-making in serial production systems.

## 3.1 Research Paradigm and Design Science Framework

This research adopts a *Design Science Research* (DSR) paradigm [28], which is particularly suited for addressing complex engineering problems through the development of purposeful artifacts. In contrast to purely descriptive or explanatory research, DSR focuses on the creation and evaluation of innovative solutions that extend the existing body of knowledge.

The central objective of this work is the development of a probabilistic decision-support artifact for maintenance planning in serial production systems. This artifact integrates measurement, modelling, and inference mechanisms into a unified framework capable of supporting maintenance decisions under uncertainty.

Within the DSR paradigm, these components collectively form the *research artifact*, which is designed to support maintenance decisions under uncertainty.

The research follows an iterative build–evaluate cycle typical for Design Science Research, where the artifact is progressively refined through controlled experimental evaluation.

The evaluation is primarily simulation-based, enabling controlled experimentation across varying degradation patterns and maintenance policies while preserving reproducibility.

By framing the research within DSR, the contribution of this work is not limited to model development, but extends to the design of a generalizable decision-support artifact for maintenance logistics under uncertainty.

## 3.2 General Research Objective

The overarching objective of this dissertation is the formulation and investigation of a probabilistic, timestamp-driven framework for maintenance decision-making in serial production environments.

More specifically, the dissertation seeks to:

- Establish a timestamp-based reconstruction of OEE that is independent of predefined performance parameters or ideal cycle times.
- Model degradation dynamics using reliability theory and analyze their influence on the temporal evolution of production performance indicators.
- Integrate degradation modelling and reconstructed OEE signals into a Bayesian probabilistic framework capable of estimating the probability that maintenance actions can be safely postponed.
- Validate the proposed framework using discrete-event simulation (DES) datasets generated under controlled degradation scenarios.

## 3.3 Research Hypothesis

The central hypothesis of this research is that maintenance decision-making in serial production systems can be reformulated as a probabilistic confidence decision problem rather than a deterministic scheduling problem.

More precisely, the hypothesis states that maintenance decisions can be transformed from interval-based scheduling policies into probabilistic confidence threshold decisions when Overall Equipment Effectiveness (OEE) is reconstructed from timestamp-based production data and interpreted within a Bayesian inference framework.

## 3.4 Specific Research Objectives

These objectives operationalize the research hypothesis and guide the development and evaluation of the proposed decision-support artifact.

To operationalize this hypothesis, the dissertation pursues the following specific objectives:

1. Investigate whether OEE can be reconstructed objectively from timestamped production logs without relying on predefined ideal cycle times.

2. Reinterpret OEE as a probabilistic measure representing the likelihood that a production system operates under ideal manufacturing conditions.

3. Analyze degradation dynamics in periodically maintained systems using Weibull-based hazard models and investigate the resulting sawtooth hazard structure produced by periodic maintenance resets.

4. Investigate the use of Bayesian Neural Networks (BNNs) to estimate the probability that reconstructed OEE will fall below a predefined performance threshold within a future decision interval.

5. Analyze the behaviour of the resulting Bayesian decision rule under different maintenance regimes, including aggressive maintenance, insufficient maintenance, and stable maintenance strategies.

## 3.5 Methodological Architecture

In contrast to the conceptual Design Science Research framing introduced earlier, this section describes the concrete technical realization of the proposed research artifact.

The methodological framework follows a layered structure that integrates measurement theory, reliability modelling, simulation experimentation, and probabilistic machine learning.

The technical implementation of the research artifact consists of four interrelated analytical components:

1. Timestamp-based reconstruction of production performance (OEE),

2. Mathematical modelling of system degradation,

3. Simulation-based generation of experimental datasets,

4. Bayesian inference for maintenance decision support.

### 3.5.1 Data Foundation

The research relies exclusively on synthetic datasets generated through discrete-event simulation.

Synthetic production logs are generated using discrete-event simulation implemented in Siemens Plant Simulation.

This controlled experimental environment allows systematic variation of:

- Failure distributions (Negative Exponential versus Weibull),
- Maintenance interval strategies,
- Degradation intensity parameters.

The use of synthetic data enables the isolation of causal relationships between degradation processes, OEE evolution, and maintenance decision outcomes under statistically controlled conditions.

### 3.5.2 Timestamp-Based OEE Reconstruction

OEE is reconstructed exclusively from timestamped production data.

Cycle times are defined as:

$$CT_i = t_i - t_{i-1} \tag{3.1}$$

where

$$CT_i = \text{cycle time of part } i \tag{3.2}$$

$$t_i = \text{timestamp of part } i \tag{3.3}$$

Performance losses are detected using robust statistical measures, including median-based cycle time estimation and interquartile range analysis. [1]

Extreme cycle time deviations are interpreted as availability losses, while moderate deviations correspond to performance losses.

This formulation eliminates the need for predefined ideal cycle times and preserves objectivity across different production environments.

### 3.5.3 Degradation Modeling

To represent realistic failure behaviour beyond constant hazard assumptions, a Weibull-based hazard model with increasing hazard rate ($\beta > 1$) is considered.

[1] The conceptual design of the timestamp-based OEE reconstruction was partly inspired by practical insights gained from the author's professional experience with the Audi Hungaria production data acquisition system (BDE). Due to internal data governance and confidentiality restrictions, the original industrial datasets could not be used within this dissertation.

Consequently, all analytical results and validation presented in this work are based exclusively on synthetically generated datasets produced through discrete-event simulation.

$$h(t) = \frac{\beta}{\eta^{\beta}} t^{\beta-1} \tag{3.4}$$

where

$$h(t) = \text{hazard rate at time } t \tag{3.5}$$

$$\eta = \text{scale parameter} \tag{3.6}$$

$$\beta = \text{shape parameter} \tag{3.7}$$

In the simulation experiments presented in this dissertation, the representative case $\beta = 2$ is used, resulting in a linearly increasing hazard rate over time. This behaviour forms the basis for a sawtooth degradation model in which periodic maintenance resets the hazard rate.

The model reflects realistic midlife degradation patterns and allows analytical comparison with Negative Exponential failure processes.

### 3.5.4 Discrete Event Simulation

Discrete-event simulation is used to generate controlled degradation scenarios.

The simulation environment implemented in Siemens Plant Simulation enables:

- comparison of Negative Exponential and Weibull failure structures,
- evaluation of aggressive, insufficient, and balanced maintenance regimes,
- sensitivity analysis of maintenance interval strategies.

The simulation model therefore functions as an experimental laboratory for generating statistically consistent training and validation datasets for the probabilistic models used in this research.

### 3.5.5 Bayesian Decision Modeling

Maintenance decision-making is formulated as a probabilistic confidence problem based on predicted production performance.

The decision rule can be expressed as:

$$P(OEE_{t+1} < OEE_{threshold} \mid Evidence, \neg M_t) \tag{3.8}$$

where

$$OEE_{t+1} = \text{predicted OEE in the next decision interval} \tag{3.9}$$

$$OEE_{threshold} = \text{critical performance threshold} \tag{3.10}$$

$$M_t = \text{maintenance action at time } t \tag{3.11}$$

A Bayesian Neural Network estimates this probability together with its associated prediction uncertainty.

Maintenance intervention is triggered when:

$$P > \tau \tag{3.12}$$

where

$$\tau = \text{predefined risk tolerance threshold} \tag{3.13}$$

This formulation transforms preventive maintenance from fixed interval scheduling into a probabilistic confidence-based decision strategy.

## 3.6 Scientific Contribution

The methodological contribution of this research lies not in individual components, but in their integration into a coherent probabilistic decision-support framework.

This framework combines the following elements:

- a universally applicable timestamp-based OEE formulation,
- a mathematically consistent degradation model,
- a probabilistic maintenance decision rule based on Bayesian inference,
- and a simulation-based experimental validation framework.

Together, these elements establish a formal link between classical maintenance concepts and probabilistic modelling approaches, resulting in an analytically grounded framework for maintenance decision support in serial production systems.

The detailed scientific contributions are formalized and evaluated in Chapter 10.

## 3.7 Traceability of Research Gap, Questions, and Contributions

To conclude the methodological framework, the identified research gaps are explicitly linked to research questions, corresponding chapters, and resulting contributions.

Table 3.1: Traceability Matrix Linking Research Gap, Questions, and Results

| **Research Gap** | **Research Question** | **Chapter** | **Result / Contribution** |
|---|---|---|---|
| Lack of uncertainty-aware maintenance decision models | RQ1 | Chapters 3–5 | Bayesian decision framework for maintenance postponement |
| Limited integration of OEE into predictive models | RQ2 | Chapters 4–6 | Probabilistic interpretation of OEE as performance signal |
| Weak linkage between prediction and operational logistics | RQ3 | Chapters 6–8 | Maintenance regime classification and decision support |

This mapping establishes a direct traceability between problem definition, methodological design, and resulting contributions, ensuring a coherent and verifiable research structure.

The framework introduced in this chapter represents the operationalization of the thesis contribution, translating probabilistic modelling into a concrete decision logic applicable to industrial maintenance planning.

# 4 Development of a General OEE Calculator.

## 4.1 Introduction

In modern discrete event simulation and production systems analysis, accurately capturing the dynamic behavior of machines, processes, and lines is essential. [29, 30] This work explores the mathematical and theoretical foundations underlying the modeling and evaluation of production systems, focusing on difference calculus, Little's Law, Overall Equipment Effectiveness (OEE), and their integration into probabilistic maintenance decision-making frameworks.

## 4.2 Probabilistic Foundation of Manufacturing Performance

Manufacturing systems operate under inherent uncertainty, where equipment effectiveness emerges from the complex interaction of availability, performance, and quality factors. Traditional OEE approaches treat these as deterministic ratios, but this perspective fails to capture the probabilistic nature of real manufacturing environments.

### 4.2.1 Reinterpreting OEE as a Probability Measure

Rather than viewing OEE as a simple multiplicative ratio, I propose reinterpreting it as a probability measure that quantifies the likelihood of achieving ideal manufacturing conditions:

$$\text{OEE} = P(\text{ideal manufacturing conditions}) \tag{4.1}$$

This probabilistic interpretation recognizes that manufacturing effectiveness is fundamentally about the system's ability to consistently operate under optimal conditions. Each component of the traditional OEE formula can be understood as a conditional probability:

$$\text{Availability} = P(\text{equipment operational} \mid \text{scheduled time}) \tag{4.2}$$

$$\text{Performance} = P(\text{optimal speed} \mid \text{equipment operational}) \tag{4.3}$$

$$\text{Quality} = P(\text{conforming output} \mid \text{production completed}) \tag{4.4}$$

The overall OEE then represents the joint probability of all ideal conditions occurring simultaneously:

$$\begin{aligned} P(\text{maintenance needed} \mid \text{OEE observations}) &\propto \text{Likelihood} \times \text{Prior} \\ = P(\text{OEE observations} \mid \text{maintenance needed}) &\times P(\text{maintenance needed}) \end{aligned} \tag{4.5}$$

### 4.2.2 Implications for Discrete Manufacturing Systems

In discrete manufacturing environments, this probabilistic perspective aligns naturally with the event-driven nature of production systems. Each production cycle represents a discrete trial where the system either achieves or fails to achieve ideal conditions. The statistical OEE approach leverages this discrete structure by:

- Using empirical distributions of cycle times rather than theoretical specifications
- Treating performance variations as probabilistic events rather than deterministic deviations
- Enabling uncertainty quantification essential for maintenance decision-making

### 4.2.3 Connection to Bayesian Reasoning

The probabilistic formulation naturally extends to Bayesian frameworks where prior beliefs about system performance can be updated with observed data. This creates a foundation for adaptive maintenance strategies where:

$$\begin{aligned} P(\text{maintenance needed} \mid \text{OEE observations}) &\propto \\ P(\text{OEE observations} \mid \text{maintenance needed}) &\times P(\text{maintenance needed}) \end{aligned} \tag{4.6}$$

This Bayesian perspective enables the integration of OEE signals with maintenance decision logic, forming the theoretical bridge to the predictive maintenance framework developed in subsequent chapters.

**Digital Twin Integration**

The probabilistic interpretation of OEE enables seamless integration with digital twin technologies. Since both the physical system and its digital counterpart operate under uncertainty, probability distributions provide a common language for comparison and validation. This standardization allows meaningful performance comparisons across different systems and supports real-time updating of digital twin parameters based on observed manufacturing data. [31]

The mathematical foundations developed in the following sections build upon this probabilistic perspective, showing how Little's Law and queuing theory provide the analytical tools to operationalize these concepts in practice.

## 4.3 Fundamentals of Difference Calculus in Production Modeling

In the field of discrete simulations, especially within production units operating on discrete events or time steps, finite difference calculus plays a crucial role. [32] It approximates the derivative of a function at specific points using discrete measurements, providing valuable insights into dynamic production behavior.

Key types include the forward difference

$$\Delta f(x) = f(x + h) - f(x) \tag{4.7}$$

which measures change over a forward step $h$, useful for monitoring hourly output shifts in $P(t)$ via

$$\Delta P(t) = P(t + 1) - P(t) \tag{4.8}$$

The backward difference

$$\nabla f(x) = f(x) - f(x - h) \tag{4.9}$$

compares current with previous outputs, and the central difference

$$\delta f(x) = f(x + \frac{h}{2}) - f(x - \frac{h}{2}) \tag{4.10}$$

balances forward and backward perspectives.

These finite differences approximate derivatives of production functions, identify growth trends, and support dynamic equation modeling, such as discrete, first-order non-

homogeneous difference equations [33]:

$$P(n+1) = A(n)P(n) + G(n) \tag{4.11}$$

Similarly, time dynamics can be modeled by

$$T(n+1) = B(n)T(n) + H(n) \tag{4.12}$$

where $B(n)$ and $H(n)$ incorporate process efficiency factors and external time adjustments. [34]

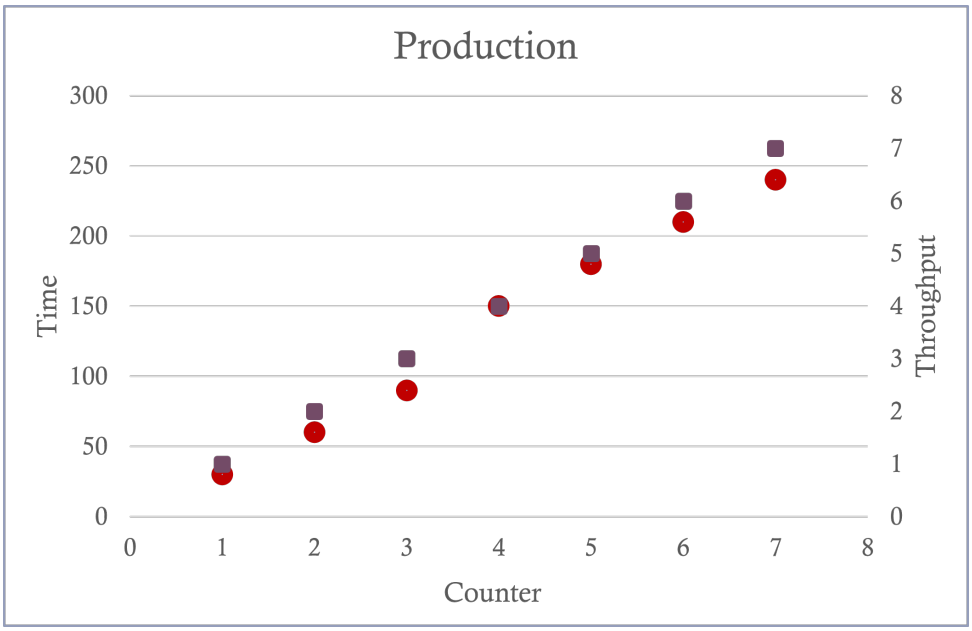



Figure 4.1: Production through the lens of discrete events.

**Recursive Definitions**

Production can be modeled as a piecewise or event-driven system characterized by two interrelated recursive functions:

$$\begin{cases} \text{Throughput}_N = F\left(\text{Throughput}_{N-1}, \text{Current Cycle Time}\right) \\ \text{Time}_N = F\left(\text{Time}_{N-1}, \text{Average Cycle Time}\right) \end{cases} \tag{4.13}$$

These functions form a discrete-time representation of production where each index N corresponds to a production event, possibly defined by the start or end of a processing cycle.

Although the above recursive functions form the foundational logic of discrete-event based simulation, their practical utility in day-to-day operations remains limited. For

general estimation and performance analysis, a more concise and universally applicable relation is desirable. Such a relation is provided by Little's Law, which offers a closed-form estimator linking average system load, throughput, and lead time.

## 4.4 Application of Little's Law in Discrete Manufacturing

### 4.4.1 Characteristics of Queuing Systems

The characteristics of queuing systems can be quantitatively analyzed using six core elements [35]:

1. **Arrival Pattern of Customers:** Described by the distribution of inter-arrival times. Patterns may be stationary or non-stationary. Customer behaviors such as balking, reneging, and jockeying reflect impatience.

2. **Service Pattern of Servers:** Service times are generally stochastic and can be individual or batch. Service behavior may depend on the number of waiting customers (state-dependence) or time (non-stationarity).

3. **Number of Servers:** A key design trade-off between customer delay and operational cost. Multi-server configurations can include shared queues or individual queues. Shared-queue systems tend to be more efficient.

4. **Queue Discipline:** Refers to the rule by which customers are selected for service, including FCFS, LCFS, RSS, processor sharing, polling, and priority disciplines (preemptive or non-preemptive).

5. **System Capacity:** Refers to the maximum number of customers allowed in the system. In finite systems, arrivals may be blocked upon reaching capacity.

6. **Stages of Service:** Queuing systems may consist of one or multiple service stages, possibly including feedback or recycling loops (e.g., rework or repeated routing).

### 4.4.2 Standard Notation (Kendall's Notation)

A queuing system is denoted using Kendall's notation as:

$$A/B/X/Y/Z$$

where:

- $A$: Inter-arrival time distribution,

- $B$: Service time distribution,
- $X$: Number of parallel servers,
- $Y$: System capacity (maximum number of entities allowed),
- $Z$: Queue discipline.

Queueing System Characteristics

| **Characteristic** | **Symbol** | **Explanation** |
|---|---|---|
| Interarrival-time distribution ($A$) Service-time distribution ($B$) | $M$ | Exponential |
| | $D$ | Deterministic |
| | $E_k$ | Erlang type $k$ ($k = 1, 2, \ldots$) |
| | $H_k$ | Mixture of $k$ exponentials |
| | $PH$ | Phase type |
| | $G$ | General |
| **Parallel servers** ($X$) | $1, 2, \ldots, \infty$ | |
| **System capacity** ($Y$) | $1, 2, \ldots, \infty$ | |
| **Queue discipline** ($Z$) | FCFS | First come, first served |
| | LCFS | Last come, first served |
| | RSS | Random selection for service |
| | PR | Priority |
| | GD | General discipline |

**Example:** $M/D/2/\infty/FCFS$ represents a system with exponential inter-arrival times, deterministic service times, two servers, infinite capacity, and a first-come-first-served discipline.

### 4.4.3 Little's Law

Little's Law is a foundational principle in queuing theory and is widely applicable in the analysis of dynamic systems. [36] It defines a fundamental relationship among three key quantities:

- $\lambda$: the average arrival rate of customers (or items) into the system,
- $W$: the average time a customer (or item) spends within the system,
- $L$: the average number of customers (or items) present in the system.

The law states:

$$L = \lambda \cdot W \tag{4.14}$$

Given any two of these three variables, the third can be directly inferred. This relationship holds under minimal assumptions: only that entities depart after arriving ($W \geq 0$) and the system is stable (i.e., the arrival rate equals the departure rate over time). It does not require assumptions like Poisson arrivals, exponential service times, or specific queue disciplines.

**Examples of System Definitions:**

1. **System = Queue + Server:**

$$L = \lambda \cdot W \tag{4.15}$$

   with $W$ including both waiting and service times.

2. **System = Queue Only:**

$$L_q = \lambda \cdot W_q \tag{4.16}$$

   where $L_q$ is average number waiting and $W_q$ is average waiting time.

3. **System = Single Server:**

$$L_s = \lambda \cdot \mathbb{E}[S] = 1 - p_0 \tag{4.17}$$

   where $\mathbb{E}[S]$ is the expected service time and $p_0$ is the fraction of idle time.

4. **System with Blocking (Finite Capacity):**

$$L = (1 - p_b) \cdot \lambda \cdot W \tag{4.18}$$

   where $p_b$ is the blocking probability and $W$ is measured only for admitted entities.

### 4.4.4 Geometric Interpretation and Proof of Little's Law

While formal proofs of Little's Law rely on rigorous stochastic process theory, a compelling geometric argument can be given to illustrate its essence. This section develops the intuition behind the result, progressively relaxing assumptions on the system's state at the beginning and end of the observation period.

**Geometric Setting**

Consider a queuing system observed over a finite time interval $[0, T]$. Let:

- $A(t)$: cumulative number of *Arrivals* by time $t$,
- $D(t)$: cumulative number of *Departures* by time $t$,
- $N(t) = A(t) - D(t)$: number of customers *in the system* at time $t$,
- $N$: total number of arrivals during $[0, T]$,
- $W_k$: time customer $k$ spends in the system,
- $W = \frac{1}{N} \sum_{k=1}^{N} W_k$: average time in system,
- $\lambda = \frac{N}{T}$: average arrival rate,
- $L = \frac{1}{T} \int_0^T N(t)\, dt$: average number of customers in the system over time.

Assume that each customer is represented by a horizontal rectangle from arrival time $A_k$ to departure time $D_k = A_k + W_k$, forming a stack of shaded regions. The total shaded area then equals the total time all customers spend in the system:

$$\int_0^T N(t)\, dt = \sum_{k=1}^{N} W_k \tag{4.19}$$

Dividing the left and right term by $T$ gives:

$$\frac{1}{T} \int_0^T N(t)\, dt = \frac{1}{T} \sum_{k=1}^{N} W_k = \frac{N}{T} (\frac{1}{N} \sum_{k=1}^{N} W_k) \tag{4.20}$$

Which resolves in:

$$L = \lambda \cdot W \tag{4.21}$$

**Case 1: System Starts and Ends Empty**

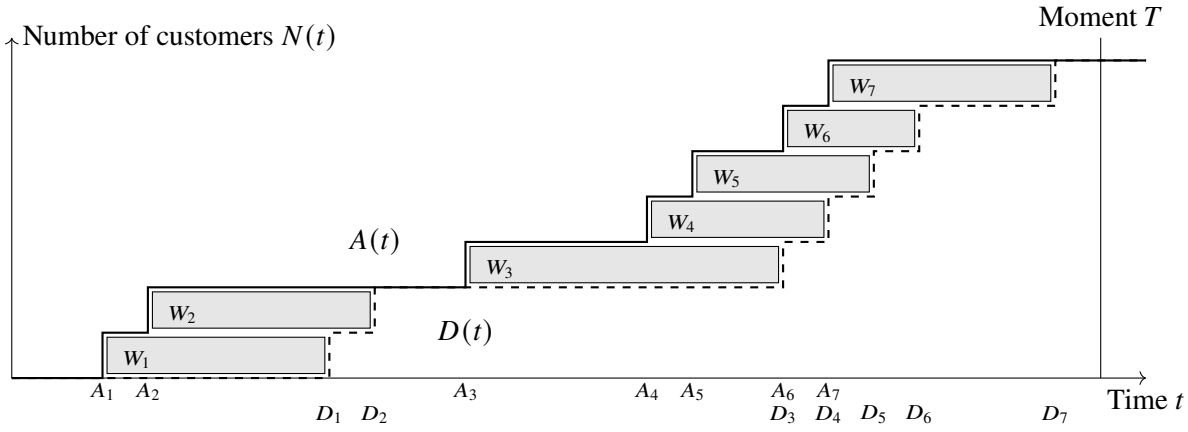



Figure 4.2: Geometric representation of Little's Law (in-order departures)

If the system is empty at both $t = 0$ and $t = T$, then $A(0) = D(0)$ and $A(T) = D(T)$. In this case, the total shaded area (sum of waiting times) coincides exactly with the integral

of the number of customers in the system and Little's Law gives an accurate result.

**Case 2: Out-of-Order Departures**

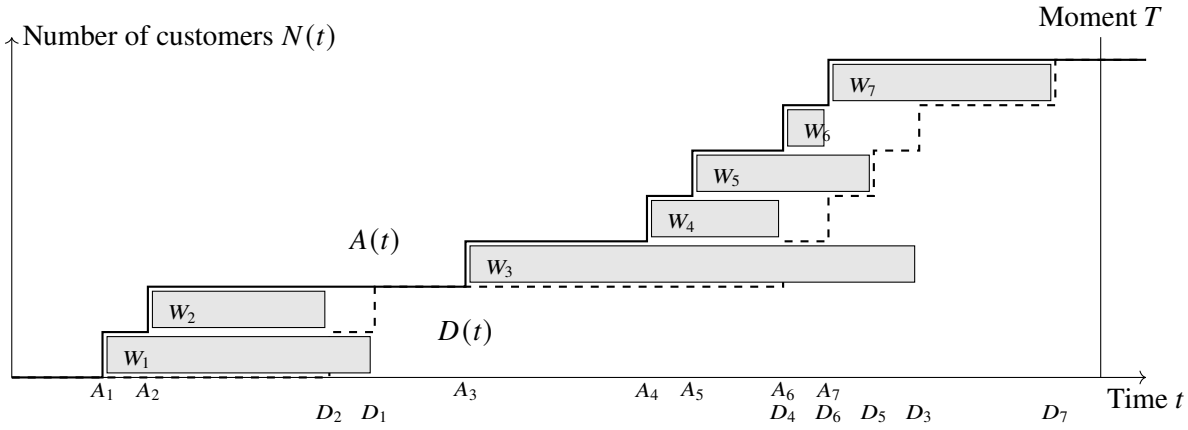



Figure 4.3: Geometric representation of Little's Law (out of order departures)

If customers depart in a different order than their arrivals, the rectangles representing their time in the system no longer align neatly with the departure curve $D(t)$. However, for every portion of a rectangle that extends beyond $D(t)$, there exists a corresponding empty space between $A(t)$ and $D(t)$. These regions can be rearranged to preserve total area, and thus the equality still holds, leading again to:

$$L = \lambda \cdot W \tag{4.22}$$

**Case 3: System Does Not End Empty**

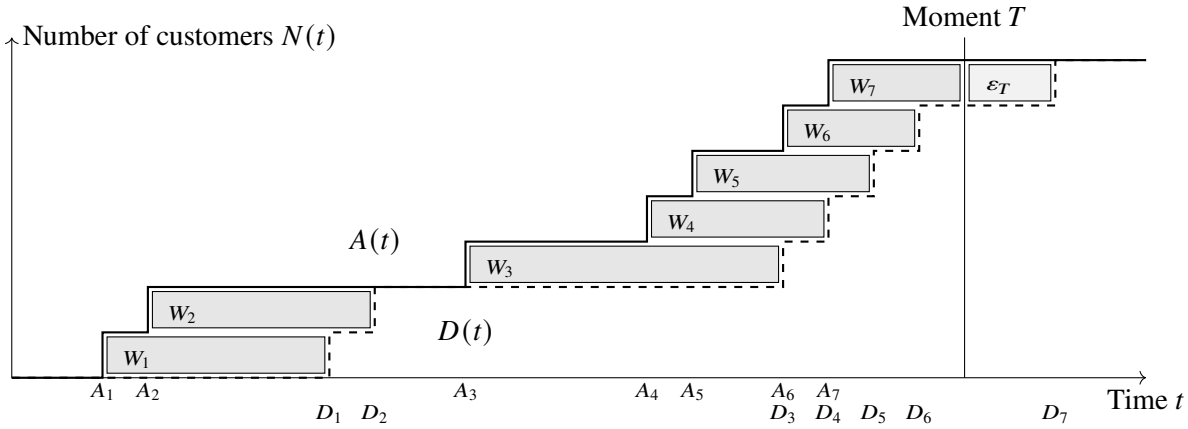



Figure 4.4: Geometric representation of Little's Law (in-order departures, no empty ending)

If the system does not end in an empty state, then some rectangles $W_k$ extend beyond $T$. The integral

$$\int_0^T N(t)\,dt \tag{4.23}$$

only accounts for time spent in the system up to time $T$. Therefore, the equality becomes:

$$\sum_{k=1}^{N} W_k \approx \int_0^T N(t)\,dt + \varepsilon_T \tag{4.24}$$

where $\varepsilon_T$ represents the excess time beyond the horizon. Over a sufficient long time span, this mismatch becomes negligible:

$$\lim_{T\to\infty} \frac{\varepsilon_T}{T} = 0 \tag{4.25}$$

Hence, in the limit:

$$L = \lambda \cdot W \tag{4.26}$$

still holds.

**Case 4: System Does Not Start Empty**

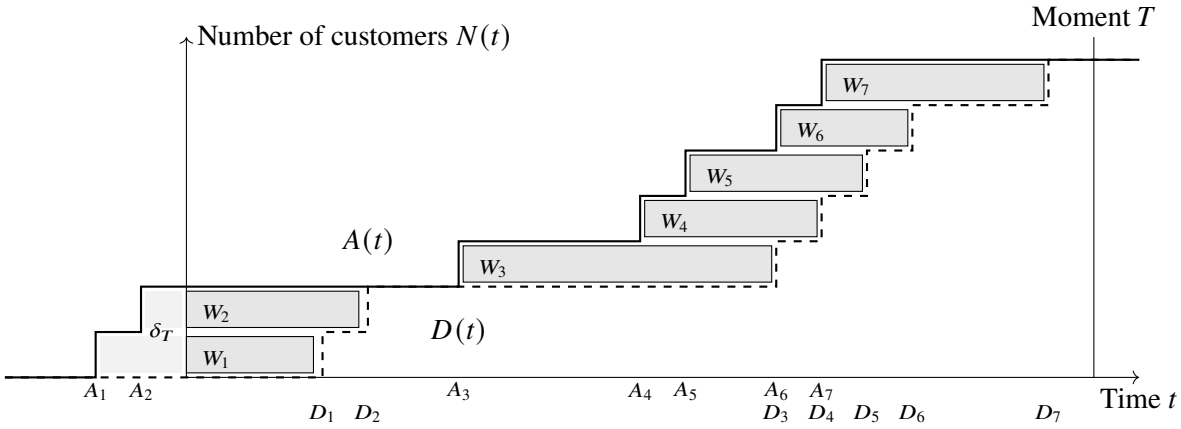



Figure 4.5: Geometric representation of Little's Law (in-order departures, occupied start)

If the system does not start empty, there are already customers present at time $t = 0$, who contribute area under the $N(t)$ curve but are not included in $A(t)$. Let the number of customers present initially be $N(0)$. These individuals are not part of the $N$ arrivals

counted in $A(t)$, so their time in system is not included in $\sum_{k=1}^{n} W_k$, but they do contribute to the integral $\int_0^T N(t)\,dt$. Thus, the shaded area understates the actual time integral:

$$\int_0^T N(t)\,dt \approx \sum_{k=1}^{n} W_k + \delta_T \tag{4.27}$$

with $\delta_T$ accounting for initial residents. Again, as $T \to \infty$, this discrepancy becomes negligible:

$$\lim_{T\to\infty} \frac{\delta_T}{T} = 0 \tag{4.28}$$

so the average number in the system still converges to:

$$L = \lambda \cdot W \tag{4.29}$$

## Summary

Regardless of whether the system begins or ends in an empty state, or whether customers depart in or out of arrival order, the geometric intuition behind Little's Law holds in the limit as $T \to \infty$, provided that:

- the long-run arrival rate $\lambda = \lim_{t\to\infty} \frac{A(t)}{t}$ exists and is finite,
- the long-run average time in system per customer $W = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} W_k$ exists and is finite.

Then, the average number of customers in the system satisfies:

$$L = \lambda \cdot W \tag{4.30}$$

This relationship is both robust and universal, making it one of the most powerful tools in queuing theory and performance analysis.

Examples at different system levels include:

- Machine level: 5 parts/minute, 2 minutes average duration $\to L = 10$ parts.
- Work sequence: 20 parts/hour, 3 hours average $\to L = 60$ parts.
- Line segment: 40 parts/hour, 2 hours average $\to L = 80$ parts.
- Entire line: 200 parts/hour, 5 hours average $\to L = 1000$ parts.

Properly observing $\lambda$ and $W$ over time ensures stable estimates. These insights underpin the design of buffers, performance targets, and reliability measures, as discussed in [96] and [97].

## 4.5 From Classical OEE to a Data-Driven Estimator

This section builds directly on the theoretical foundations established in the previous parts of this chapter, particularly Little's Law, which demonstrates that throughput, work-in-process, and cycle time are invariant relationships across systems and subsystems. By analogy, OEE should also be invariant at different levels of analysis: the overall effectiveness of a line must equal the effectiveness of its parts, even if the categorisation of losses shifts between availability, performance, or quality. The approach presented here applies this reasoning to derive a timestamp-based estimator of OEE that is both data-driven and consistent with queuing-theoretic principles.

### 4.5.1 Motivation and Theoretical Foundation

- **Availability** retains its conventional interpretation:

$$A = \frac{\text{Operation Time}}{\text{Loading Time}} = \frac{\text{Loading Time} - \text{Downtime}}{\text{Loading Time}} \tag{4.31}$$

  This captures time-based efficiency and is consistent with TPM-aligned definitions when system states (operating, down) are well-defined.

- **Quality** also follows standard practice:

$$Q = \frac{\text{Processed Quantity} - \text{Defects}}{\text{Processed Quantity}} \tag{4.32}$$

  assuming reliable and complete detection of nonconforming items.

### 4.5.2 Different levels of perception

In practice, OEE raises a methodological difficulty: the same formula may yield different results depending on the **level of aggregation**.

- At machine level, performance reflects whether a unit runs at its design speed. Blocking and starvation appear as local availability losses.
- At line level, performance reflects whether the flow between machines is synchronised. Time loss in one unit may reappear as a speed effect at another.
- At system level, OEE should remain invariant: the overall number must represent the ratio of good output to the maximum achievable output under ideal conditions, regardless of decomposition. Only the categorisation of losses shifts between levels.

This mirrors the logic of Little's Law, which holds for a system and all its subsystems:

$$L = \lambda \cdot W \tag{4.33}$$

remains invariant, even if the interpretation of $L$, $\lambda$, and $W$ differs across levels.

### 4.5.3 Reinterpreting Performance as flow synchronisation

While Availability and Quality remain aligned with classical TPM definitions, **Performance** requires reinterpretation in flow-oriented systems. Instead of viewing performance as a simple cycle-time ratio, I define it as a factor describing **flow stability**:

$$P = (1 - P_0) \times (1 - P_b) \tag{4.34}$$

where

$$1 - P_0 = \text{probability of not being starved (adequate upstream supply)}, \tag{4.35}$$

$$1 - P_b = \text{probability of not being blocked (adequate downstream buffering)}. \tag{4.36}$$

Thus, performance measures how well the buffers before and after a machine are designed to keep the flow stable. [37]

- **Speed Rate** $(1 - P_0)$**:** indicates how often a machine avoids running empty.
- **Net Operating Rate** $(1 - P_b)$**:** indicates how often a machine avoids being blocked and how well downstream disruptions are resolved.

### 4.5.4 Operating Speed and Net Operating Rate

The performance degradation due to downstream congestion (blocking) is captured by the **Operating Speed**, defined as:

$$\text{Operating Speed} = \frac{CT_{\text{theo}}}{CT_{\text{act}}} = \frac{\lambda_{\text{act}}}{\lambda_{\text{theo}}} = 1 - p_b \tag{4.37}$$

where:

- $\lambda_{\text{act}}$: observed throughput rate
- $\lambda_{\text{theo}}$: ideal or design throughput rate
- $p_b$: probability the machine is blocked (output prevented)

Likewise, upstream starvation is reflected in the **Net Operating Rate**, given by:

$$\begin{aligned} \text{Net Operating Rate} &= \frac{\text{Processed Amount} \cdot CT_{\text{act}}}{\text{Operation Time}} \\ &= (1 - p_0) \cdot \lambda_{\text{act}} \cdot CT_{\text{act}} \\ &= (1 - p_0) \end{aligned} \tag{4.38}$$

Here, $p_0$ denotes the probability the machine is starved—i.e., waiting for an incoming part.

#### Deriving the Performance Factor

By combining the above, I arrive at a more complete and behaviorally grounded expression for performance:

$$P = (1 - p_b) \cdot (1 - p_0) \cdot \lambda_{\text{act}} \cdot CT_{\text{act}} \tag{4.39}$$

In practice, $\lambda_{\text{act}} \cdot CT_{\text{act}} \approx 1$ under stable conditions, so the essential drivers of performance loss become the probabilities $p_b$ and $p_0$.

### 4.5.5 Toward an Empirical and Distribution-Free Definition

To avoid dependence on theoretical queuing models or ideal cycle assumptions, I reinterpret $p_0$ and $p_b$ empirically using the quantiles of the observed cycle time distribution. Instead of normalizing by the interquartile range (IQR), I normalize by the full outlier limit range (OLR), which reflects the data range excluding extreme outliers:

$$\text{OLR} = (Q_3 + 1.5 \cdot \text{IQR}) - \max(0, Q_1 - 1.5 \cdot \text{IQR}) \tag{4.40}$$

Using this, it can be defined:

- $p_0 \approx \dfrac{Q_3 - \text{Median}}{\text{OLR}}$ (starvation tail)
- $p_b \approx \dfrac{\text{Median} - Q_1}{\text{OLR}}$ (blocking front)

This leads to a purely timestamp-driven calculation of performance:

$$P = (1 - p_0) \cdot (1 - p_b) \tag{4.41}$$

### 4.5.6 Toward a data-driven estimator

Cycle-time distributions derived from event logs offer a practical way to operationalise these concepts. From timestamps of part completions, cycle times are computed and

classified using robust statistical thresholds:

- Median (Q2): central tendency of stable operation,
- Q3 (third quartile): upper bound of expected variation.

Classification rules:

1. Cycles $> Q3 \quad \Rightarrow$ Failures (Availability losses).
   Availability is reduced by the excess time beyond Q3:

$$A = 1 - \frac{\sum_{CT_i > Q3}(CT_i - Q3)}{T_{\text{Load}}} \tag{4.42}$$

2. Cycles $\leq$ Median $\quad \Rightarrow$ Blocking recoveries $(1 - P_b)$.

3. Median $<$ Cycles $\leq Q3 \quad \Rightarrow$ Starvation $(1 - P_0)$.

From the relative frequencies:

$$1 - P_b = \frac{\#\{CT \leq \text{Median}\}}{N}, \quad 1 - P_0 = \frac{\#\{\text{Median} < CT \leq Q3\}}{N} \tag{4.43}$$

Performance then follows as:

$$P = (1 - P_b)(1 - P_0). \tag{4.44}$$

Quality is calculated as:

$$Q = \frac{\text{Good Parts}}{\text{All Parts}}. \tag{4.45}$$

Finally:

$$OEE = A \cdot P \cdot Q \tag{4.46}$$

### 4.5.7 Advantages of the data-driven approach

1. **Consistent across levels** – OEE invariance is preserved.
2. **Flow-oriented** – Performance reflects synchronisation, not just cycle-time ratios.
3. **Data-driven thresholds** – Median and Q3 adapt to system behaviour.
4. **Actionable decomposition** – Direct mapping to interventions (maintenance, buffer design, process stabilisation).

### 4.5.8 Solving the CT Controversy

A recurring problem in the practical application of OEE is the definition of the theoretical cycle time $CT_{\text{theo}}$. In many industrial environments this value is taken directly from machine documentation or design specifications. However, such values are often negotiated, adapted to local conventions, or even strategically chosen to present performance in a favorable light. In simulation models the problem is amplified: abstraction levels and simplifications frequently alter the meaning of $CT_{\text{theo}}$, which leads to endless debates on whether the "right" cycle time was applied. [38, 39] As a result, much of the discussion around OEE drifts away from effectiveness and instead focuses on the politics of parameter selection.

To resolve this controversy, I propose a purely statistical definition of cycle times that is grounded in the data itself. The approach distinguishes between the *actual cycle time* $CT_{\text{act}}$ and the *theoretical cycle time* $CT_{\text{theo}}$:

$$CT_{\text{act}} = \text{median of the empirical cycle time distribution,} \tag{4.47}$$

$$CT_{\text{theo}} = \begin{cases} \text{first non-zero mode of the cycle time distribution,} & \text{if such a mode exists,} \\ \text{median of the distribution,} & \text{otherwise.} \end{cases} \tag{4.48}$$

This definition is motivated by the repetitive nature of industrial machines. If a machine is engineered to perform a task in a repetitive manner, the resulting data will often show a recurring cycle time that manifests as a statistical mode. Such a mode is interpreted as the "design rhythm" of the machine. If several distinct modes occur, this typically indicates that multiple product types with different cycle times have been processed on the same resource. In cases where no significant non-zero mode exists, or where the first mode is at zero, the machine has not revealed a stable design rhythm in the data, and the median provides the most robust fallback.

In this way, the definitions of $CT_{\text{act}}$ and $CT_{\text{theo}}$ become transparent, reproducible, and free from political or model-dependent bias. They are entirely derived from empirical observations and therefore suitable both for real-world measurements and for simulation studies. By anchoring cycle times in statistical properties of the data, the persistent controversy around "which cycle time should be used" in OEE calculations is resolved, and the discussion can return to its true focus: identifying and eliminating losses.

**Relation to Little's Law.** The proposed statistical definition of cycle times also establishes a direct parallel to Little's Law. Little's Law guarantees that the relation $L = \lambda \cdot W$

holds for every system and subsystem, regardless of structure or aggregation. Similarly, the choice of $CT_{\text{theo}}$ must not depend on managerial conventions or documentation practices but should be derived from the same empirical distribution that governs the observed throughput. By grounding $CT_{\text{theo}}$ in the mode (or, if absent, the median) of the cycle time distribution, it is ensured that the measure is system-invariant: the overall OEE value remains stable across levels of analysis, while only the categorisation of losses (availability, performance, quality) shifts. This parallel to Little's Law provides a theoretical justification for abandoning documented cycle times in favour of statistical ones, as it restores consistency and invariance to OEE calculations.

### Distinguishing Efficiency from Productivity: A Maintenance-Centered Perspective

The median-based approach to OEE calculation represents a fundamental shift from traditional interpretations that often conflate *efficiency* with *productivity*. This distinction is critical for maintenance decision-making and requires careful positioning relative to established practice.

**The Commercialization Problem in OEE** Traditional OEE implementations frequently promote the misconception that "higher OEE equals more output." This stems from using theoretical cycle times that may be outdated, overly optimistic, or strategically chosen to present favorable performance metrics. The result is a focus on volume maximization rather than waste elimination—a misalignment with lean manufacturing principles and maintenance objectives.

In maintenance contexts, this volume-centric interpretation creates perverse incentives:

- Deferring maintenance to maintain high availability numbers
- Running equipment beyond optimal speeds to improve performance metrics
- Accepting quality degradation to maintain throughput targets

**Reframing OEE for Maintenance Excellence** The median-based calculation fundamentally changes what "improvement" means:

- **Traditional interpretation:** OEE improvement = increased output relative to theoretical maximum
- **Maintenance-centered interpretation:** OEE improvement = reduced waste relative to demonstrated capability

This reframing aligns with core maintenance objectives:

- **Consistency over speed:** Reducing cycle time variability indicates better equipment condition
- **Sustainability over peaks:** Median-based baselines reflect sustainable operating conditions
- **Actual vs. aspirational:** Performance measured against real capability, not design specifications

**Communicating the Paradigm Shift** For practitioners accustomed to traditional OEE, the median-based approach requires explicit positioning:

- **Efficiency vs. Capacity:** Emphasize that OEE measures how efficiently available capacity is used, not how much capacity exists
- **Waste identification:** Frame improvements as waste reduction rather than output increases
- **Predictive value:** Highlight how consistency metrics better predict maintenance needs than volume metrics

**Implementation Considerations** When introducing this approach in industrial settings:

- Present median-based OEE alongside traditional calculations during transition periods
- Focus on trend analysis rather than absolute values
- Use case studies showing how variability reduction correlates with equipment reliability

This conceptual foundation supports the maintenance decision framework by ensuring that OEE signals reflect equipment condition rather than production pressure, making them more reliable indicators for maintenance timing decisions.

On reversing OEE to infer cycle times

It is a common but misleading practice to "reverse engineer" cycle times from the OEE formula. This approach assumes that OEE can be treated as a deterministic design specification, which is incorrect.
The median cycle time $CT_{\text{act}}$ defined here is a statistical description of observed operation, not a fixed design parameter. Likewise, the theoretical cycle time $CT_{\text{theo}}$,

whether defined by the mode or median of the distribution, reflects empirical behaviour rather than machine specifications.
Machine designers work with fixed design values (e.g., maximum spindle speed or nominal takt time), while OEE is an effectiveness ratio based on actual performance. Reversing the OEE equation to produce a "design" cycle time conflates performance measurement with engineering specification and should therefore be avoided.

### 4.5.9 Illustrative Interpretation of $CT_{theo}$ and OEE Decomposition

Figure 4.6 provides an illustrative, discretised view of how the theoretical cycle time $CT_{theo}$ acts as a structural lower bound and how the classical OEE factors $A$, $P$, and $Q$ emerge from deviations in realised behaviour. The figure is intentionally schematic: each numbered block represents one produced unit, and the horizontal extent corresponds to elapsed production time.

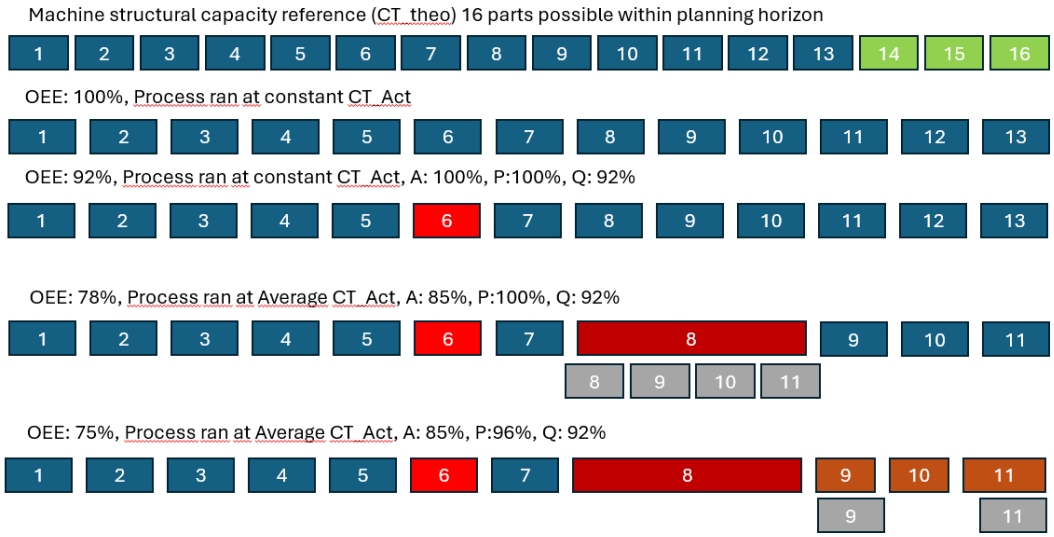



Figure 4.6: Conceptual illustration of $CT_{theo}$ as a structural capacity reference and its relationship to the OEE decomposition. The top line represents the theoretical maximum throughput (16 parts within the planning horizon) implied by $CT_{theo}$; the remaining lines show realised production with progressively introduced losses in $Q$, $A$, and $P$.

**Structural meaning of $CT_{theo}$ (top line).** The first line does *not* represent an OEE scenario. Instead, it defines the *structural capacity reference*: if a process could operate continuously at its minimal stable cycle time, then within a fixed planning horizon a maximum of 16 units would be feasible. This reference expresses a physical lower bound on cycle time, hence

$$CT_{act} \geq CT_{theo} > 0. \tag{4.49}$$

Importantly, demand limitations (e.g., only 13 units are required) affect utilisation of available capacity but do not reduce OEE when the planning horizon is defined over the *planned* production time. Therefore, the top line is used solely to anchor the interpretation of $\mathrm{CT}_{\mathrm{theo}}$ and to separate capacity utilisation from efficiency losses.

**Ideal realised operation (second line).** The second line illustrates ideal operation in which the process runs at constant $\mathrm{CT}_{\mathrm{act}} = \mathrm{CT}_{\mathrm{theo}}$ and produces the required number of units without scrap or interruptions. Under these conditions $A = 1$, $P = 1$, and $Q = 1$, yielding OEE = 1.

**Quality loss only (third line).** The third line introduces a single rejected unit (highlighted), while the cycle time remains constant. The availability and performance components remain ideal ($A = 1$, $P = 1$), and OEE is reduced purely by quality:

$$\mathrm{OEE} = A \cdot P \cdot Q = 1 \cdot 1 \cdot Q. \tag{4.50}$$

**Availability loss with stable cycle time (fourth line).** The fourth line illustrates unplanned idle time (extended red segment), which reduces availability. The cycle time during productive periods remains stable, hence $P \approx 1$, while $Q$ is unchanged relative to the quality-loss case. The example demonstrates that a reduction in throughput can occur without changing the productive cycle time itself, because lost operating time reduces $A$:

$$\mathrm{OEE} = A \cdot P \cdot Q \approx A \cdot 1 \cdot Q. \tag{4.51}$$

**Combined availability and performance loss (fifth line).** Finally, the fifth line combines availability loss with a reduction in performance, shown as a systematic increase in effective cycle time during productive periods (brown blocks). Here, availability is reduced by downtime, quality remains below unity due to scrap, and performance is reduced because $\mathrm{CT}_{\mathrm{act}}$ exceeds $\mathrm{CT}_{\mathrm{theo}}$:

$$P = \frac{\mathrm{CT}_{\mathrm{theo}}}{\mathrm{CT}_{\mathrm{act}}} < 1. \tag{4.52}$$

This case is central for the present work, because it corresponds to gradual performance deterioration that is not directly observable as a binary failure event, yet accumulates into measurable throughput loss and increased uncertainty.

**Implication for estimating $\mathrm{CT}_{\mathrm{theo}}$ from data.** In high-frequency timestamp data, naïve estimators such as $\min(\Delta t)$ are unstable because measurement noise and stochastic effects may yield arbitrarily small observed intervals. For this reason, $\mathrm{CT}_{\mathrm{theo}}$ is interpreted as

the *most frequently reached minimal stable cycle time*, inferred from the dominant inlier cluster of the cycle-time distribution after robust filtering (e.g., IQR-based). This ensures $CT_{theo} > 0$ and preserves the interpretability of the OEE decomposition.

### 4.5.10 Routines Used (Python)

The Python routines responsible for cycle-time reconstruction and OEE estimation are part of the structured data enrichment layer described in Appendix B. The appendix provides a formal description of the transformation pipeline, derived metrics, and implementation logic.

# 5 Transforming Maintenance Tasks into Failure Profiles

## 5.1 From Maintenance Actions to Hazard Functions

The failure behaviour of technical systems is often described by the *bathtub curve*, which can be represented by the Weibull distribution. The shape parameter $\beta$ distinguishes between three regimes:

- $\beta < 1$: decreasing hazard (infant mortality, dominated by early defects),
- $\beta = 1$: constant hazard (stable phase, time-independent failures),
- $\beta > 1$: increasing hazard (wear-out and ageing effects).

In this thesis I restrict the analysis to the stable operating phase, where $\beta = 1$ and the hazard function reduces to the Negative Exponential form:

$$h(t) = \lambda, \tag{5.1}$$

with $\lambda$ constant. Failures are then random, time-independent events — an assumption consistent with serial production environments where machines are operated within their designed life span.

#### TPM as Hazard Control

Total Productive Maintenance (TPM) in this context consists of two complementary types of prescribed interventions [5]:

- **Autonomous maintenance:** Routine tasks performed by operators, usually linked to shift changes or setup activities (e.g. lubrication, cleaning, inspection). These aim to prevent small disturbances from accumulating and thereby keep the hazard baseline at its expected level. [40]

- **Planned maintenance:** Scheduled activities carried out by the maintenance team, often based on OEM directives or expert knowledge. These include recalibration, replacement of wear parts, or more elaborate overhauls, typically at weekly, monthly, or annual intervals. Planned maintenance resets the effective hazard rate by restoring the equipment closer to its reference condition.

Together, autonomous and planned actions define TPM. Their effect can be interpreted as controlling the *timing and magnitude of resets* in the hazard trajectory, while operating under the assumption of constant baseline hazard between interventions. [41]

#### Predictive Maintenance as Opposite Approach

In contrast to TPM, predictive maintenance does not follow fixed schedules. Instead it relies on intensive data collection and analytics to forecast the future trajectory of $h(t)$. By estimating parameters such as $\beta$ and $\eta$ dynamically, predictive methods aim to time interventions exactly when degradation signals indicate an elevated risk. This approach, while potentially more efficient, is costly, requires high data quality, and is less plannable in practice.

#### Research Objective

The objective pursued here is to shift TPM in the direction of predictive maintenance:

- Reducing the over-aggressiveness of fixed-interval preventive activities,
- While avoiding the full data-intensity and unpredictability of predictive approaches.

This results in a hybrid concept: TPM remains structured and plannable, but is guided by stochastic indicators (e.g. OEE decay, hazard resets) that make it more adaptive to real operating conditions.

## 5.2 Sawtooth Hazard Rate Model

While the exponential and Weibull models capture fundamental aspects of failure behaviour, they are insufficient to reflect the influence of repeated maintenance. Classical preventive interventions reset the condition of equipment to a state closer to “as good as new.” The result is a hazard trajectory that grows during operation but is periodically reduced. This gives rise to a *sawtooth hazard rate model*, which alternates between degradation growth and reset events.

### 5.2.1 Mathematical Formulation

The hazard function of a Weibull distribution is given by

$$h(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}, \tag{5.2}$$

with $\beta$ the shape parameter and $\eta$ the scale parameter. [42]
To simulate periodic resets, consider a fixed maintenance interval $\tau$. For $t \in [k\tau, (k+1)\tau)$, where $k \in \mathbb{N}$ denotes the maintenance cycle, the hazard rate is defined as:

$$h_{\text{saw}}(t) = \frac{\beta}{\eta}\left(\frac{t - k\tau}{\eta}\right)^{\beta-1}. \tag{5.3}$$

At each reset point $t = k\tau$, the effective "age" of the component is reduced and the hazard rate returns to baseline. The resulting profile resembles a sawtooth: increasing with time until maintenance restores the condition.

### 5.2.2 Balancing Weibull and Exponential Areas

In the stable operating phase ($\beta = 1$), the hazard rate is constant:

$$h_{\text{exp}}(t) = \lambda, \tag{5.4}$$

representing time-independent, memoryless failures.
The sawtooth model ($\beta > 1$) produces an alternating sequence of growth and reset. The key idea is that the *area under the hazard curve* can be used to benchmark maintenance aggressiveness. [43]

- If the cumulative area under $h_{\text{saw}}(t)$ over one maintenance cycle is approximately equal to the area under $h_{\text{exp}}(t)$ for the same interval, maintenance is *balanced*: neither too frequent nor too sparse.

- If the sawtooth area is significantly larger, interventions are too frequent: the system is over-maintained, incurring unnecessary downtime and cost.

- If the sawtooth area is significantly smaller, interventions are too sparse: the system experiences excess degradation, increasing the probability of failure before scheduled maintenance.

Formally, the balance condition can be expressed as:

$$\int_0^{\tau} h_{\text{saw}}(t)\,dt \;\approx\; \int_0^{\tau} h_{\text{exp}}(t)\,dt. \tag{5.5}$$

### 5.2.3 Interpretation

This perspective links preventive maintenance directly to reliability metrics. By comparing the area under the sawtooth hazard with the constant exponential baseline, one obtains a measure of whether maintenance actions are too aggressive or too relaxed.

- A **ratio near unity** implies optimal balance: the planned interventions compensate degradation without overspending resources.
- A **ratio above unity** signals excessive maintenance effort.
- A **ratio below unity** indicates insufficient intervention, risking unexpected failures.

The sawtooth model therefore provides a mathematically tractable framework that captures both the stochastic nature of failures and the deterministic impact of scheduled maintenance. By introducing the area-balance criterion, it also offers a practical diagnostic tool to evaluate whether the chosen maintenance regime is appropriately tuned.

## 5.3 From Maintenance Actions to Maintenance–Failure Modelling

### 5.3.1 Theoretical Framework and Standing Assumptions

Attention is restricted to the stable operating phase of the bathtub curve ($\beta = 1$), so failures are time–independent with constant hazard $\lambda = 1/\text{MTBF}$. [44] Let MTBF and MTTR denote mean time between failures and mean time to repair, respectively. Availability is

$$A = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} \tag{5.6}$$

Rather than modelling failures explicitly, I replace them by their maintenance equivalents:

$$\tau^{\star} \approx \frac{\text{MTBF}}{2}, \quad d \approx \frac{\text{MTTR}}{2} \tag{5.7}$$

Rule (5.7) balances the expected "risk accumulation" between resets against planned downtime.

### 5.3.2 Implementation Framework in Discrete Event Simulation

The sawtooth hazard rate model requires specific implementation logic to accurately represent maintenance timing and effects. [45] Based on validation studies using Plant Simulation 2504, the following framework ensures proper integration of maintenance activities within production systems.

#### Maintenance Control Logic

Maintenance implementation is managed through a Generator object that controls both timing and duration of maintenance activities:
**Interval Parameter:** Defines the time between maintenance activations, effectively setting the period from the end of one maintenance event to the start of the next:

$$\text{Maintenance Interval} = \tau^\star = \frac{\text{MTBF}}{2} \tag{5.8}$$

**Duration Parameter:** Defines the maintenance execution time span:

$$\text{Maintenance Duration} = d \approx \frac{\text{MTTR}}{2} \tag{5.9}$$

#### Failure Profile Management

The sawtooth model requires dynamic failure profile management synchronized with maintenance events:
**Start of Maintenance Event:**

- If a failure is active, deactivate it immediately
- Deactivate the failure profile to prevent new failures during maintenance
- Reset the hazard accumulation to baseline

**End of Maintenance Event:**

- Set failure profile start time to current simulation time
- Set failure profile stop time to the start of next scheduled maintenance
- Reactivate the failure profile with reset hazard rate

**Initialization Protocol:** Both failure profile and maintenance generator are initialized at simulation start to ensure consistent timing throughout the simulation run.

### 5.3.3 Parameter Optimization and Calibration

The effectiveness of the sawtooth model depends critically on proper parameter calibration. The scale parameter $\eta$ in the Weibull distribution must be optimized to align with the baseline NegExp hazard rate.

#### Mathematical Optimization

Setting the Weibull hazard rate equal to the NegExp rate at the optimal maintenance point $t = \mu$:

$$\frac{2\mu}{\eta^2} = \frac{1}{\mu} \tag{5.10}$$

Solving for the optimal scale parameter:

$$\eta = \sqrt{2\mu} \tag{5.11}$$

This calibration ensures that maintenance is triggered precisely when the accumulating hazard matches the constant baseline risk, optimizing the balance between preventive action and resource utilization.

### 5.3.4 Buffering and System-Level Integration

**Buffering and system-level stabilisation.** Inter-operation buffers dampen the realisation of upstream/downstream disturbances at the workstation level. I encode buffer quality by a single mitigation factor $B \in [0, 1]$:

$$B \approx 0.8 \quad \text{(well designed buffer)}, \qquad B \approx 0.6 \quad \text{(poorly designed buffer)}.$$

A convenient way to express the effect is a *buffered availability* that accounts for the fraction of unavailability the buffer can absorb:

$$A_b \;=\; 1 - (1 - B)\,(1 - A) \;=\; A + B\,(1 - A). \tag{5.12}$$

When quality is stable ($Q \approx 1$), the *Performance* term of OEE primarily reflects the net operating rate (stability) and operating speed; (5.12) provides a lower bound on the stability component that the buffer can sustain against short failures and MTTR asymmetries.

**Postponement test from Performance and time since maintenance.** Let $t_s$ denote time elapsed since the last maintenance job and $\tau^\star$ the planning anchor. Fix an operational

OEE target $OEE_{\min}$ (or equivalently a Performance target). Given the current observations $P_{\text{obs}}$ and $Q_{\text{obs}}$ (often $Q_{\text{obs}} \approx 1$), predict the minimum Performance that still meets the target under buffered reliability [46]:

$$P_{\min} \;=\; \frac{OEE_{\min}}{A_b\, Q_{\text{obs}}} \quad (\text{clip to } [0,1]). \tag{5.13}$$

Define the *age ratio* $r := t_s/\tau^\star$ and a simple *postponement score* that trades off current headroom in Performance against elapsed time [47]:

$$\text{MPS} \;=\; \underbrace{\frac{P_{\text{obs}} - P_{\min}}{1 - P_{\min}}}_{\text{performance headroom}} \times \underbrace{\max\{0,\, 1 - r\}}_{\text{time headroom}}. \tag{5.14}$$

A practical decision rule is:

$$\text{If } \text{MPS} \geq \theta \text{ (e.g. } \theta = 0.3),\ \text{postpone;} \qquad \text{else, execute maintenance.} \tag{5.15}$$

Interpretation: (i) $A_b$ raises the floor in (5.13) when buffering is effective (large $B$), meaning the line can tolerate more local disturbances without harming OEE; (ii) as $t_s$ approaches $\tau^\star$ ($r \to 1$), the time headroom vanishes, making postponement increasingly unlikely; (iii) when $Q_{\text{obs}} < 1$, (5.13) tightens and MPS drops, so quality drift automatically curtails postponement.

**Worked algebra and implementation hints.**

(a) Compute $A$ from (5.6) using current MTBF and MTTR; set $\tau^\star = \text{MTBF}/2$, $d = \text{MTTR}$.

(b) Choose $B \in \{0.8,\, 0.6\}$ from the buffer audit; compute $A_b$ via (5.12).

(c) Fix $OEE_{\min}$ (plant policy) and read $P_{\text{obs}}, Q_{\text{obs}}$ from the OEE calculator.

(d) Form $P_{\min}$ using (5.13), $r = t_s/\tau^\star$, and MPS from (5.14).

(e) Decide with threshold $\theta$; optionally tune $\theta$ by backtesting to align with risk appetite.

**Why this remains low data intensity.** All inputs are already present in standard shopfloor reporting [48]: MTBF, MTTR (from downtime logs), buffer grade $B$ (once per line from a design audit), and $P_{\text{obs}}, Q_{\text{obs}}$ (from the OEE calculator). The method therefore shifts TPM toward a predictive posture—using Performance and elapsed time to *time* interventions—without the burden of continuous high-frequency condition monitoring. [49, 50]

**Integration with Production Scheduling**

The maintenance framework must integrate seamlessly with production planning:

- **Maintenance Windows:** Align maintenance intervals with natural production breaks where possible
- **Resource Coordination:** Ensure maintenance duration accounts for technician availability and spare parts logistics
- **Performance Monitoring:** Use OEE degradation signals to trigger early maintenance when $P_{\text{obs}} < P_{\text{min}}$

### 5.3.5 Validation and Performance Metrics

The framework's effectiveness is measured through key performance indicators that capture both reliability and efficiency improvements [51]:

- **Time Between Failures:** Extended intervals demonstrate hazard reset effectiveness
- **Maintenance Efficiency:** Ratio of preventive to corrective maintenance actions
- **System Availability:** Overall uptime considering both failures and planned maintenance
- **Throughput Stability:** Reduced variability in production output

Statistical validation through simulation experiments confirms that the sawtooth model with optimized parameters consistently outperforms fixed-interval maintenance strategies, particularly in systems with moderate to high availability requirements. [52]

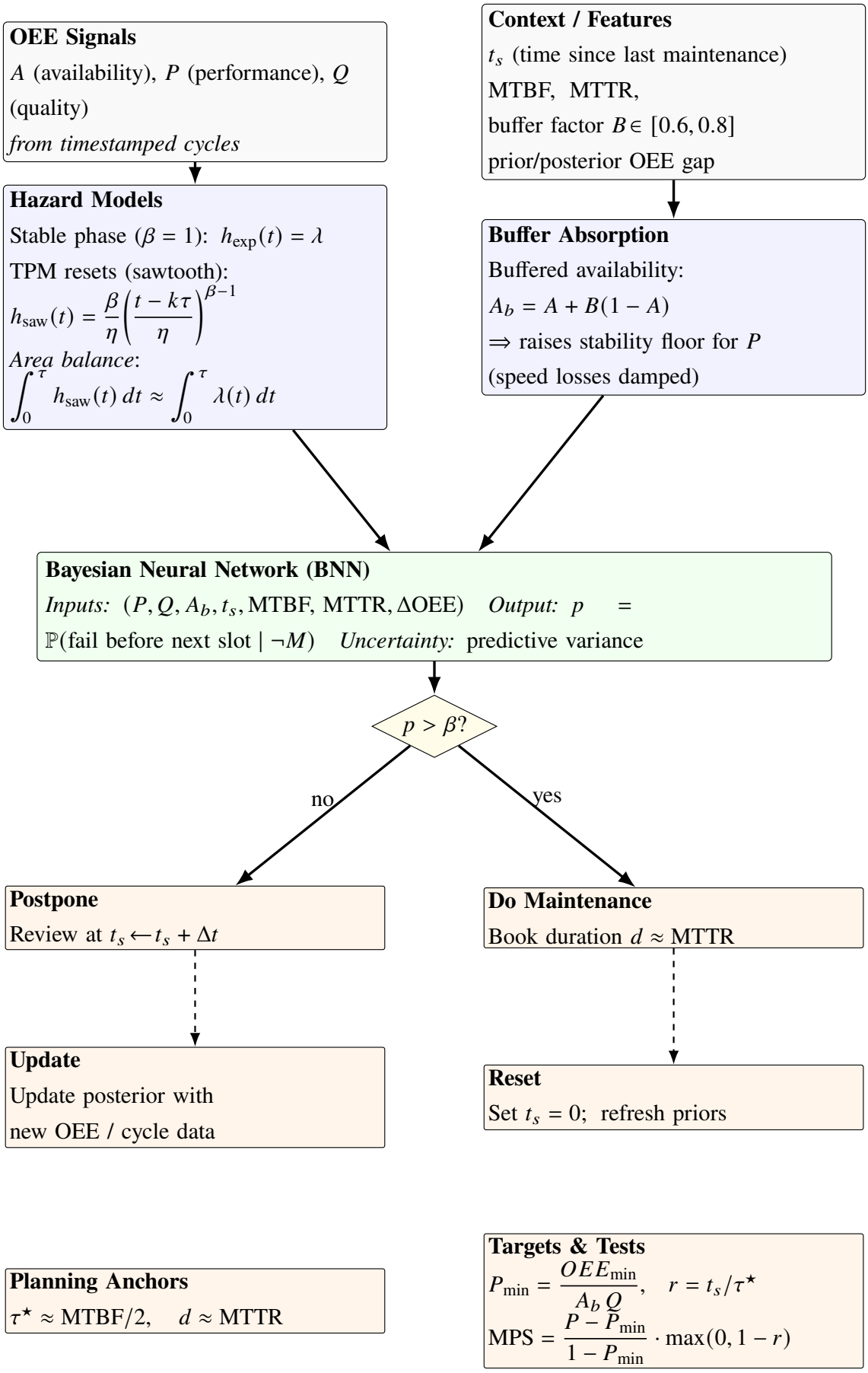



Figure 5.1: Vertical layout: (1) OEE & context; (2) hazard & buffer; (3) BNN (two-column span); (4) decision; (5) actions; (6) update/reset; (7) spacer; (8) anchors & tests.

# 6 Bayesian Neural Networks for Confidence-Based Maintenance Decision-Making

## 6.1 Introduction

The previous chapters established the probabilistic foundations for interpreting equipment performance in high-frequency production environments. Chapters 3 and 4 modelled degradation behaviour using Weibull-based sawtooth hazard profiles and Negative Exponential decay, while Chapter 5 reframed the Overall Equipment Effectiveness (OEE) as a probability measure defined over event-spaces for availability, performance, and quality. These developments motivate a transition away from deterministic maintenance strategies toward a fully probabilistic decision framework capable of quantifying uncertainty and supporting risk-aware intervention timing.

In this chapter, the conceptual basis for using Bayesian Neural Networks (BNNs) is introduced as the core mechanism for predicting the probability of unacceptable performance loss. [53, 54, 55] I do not yet describe the implementation or architecture of the BNN—that is deferred to Chapter 7. Instead, I establish the reasoning behind selecting Bayesian methods, explain the probabilistic interpretation of the OEE signal, and formalise the decision rule that governs maintenance execution or postponement. [56, 57]

## 6.2 Limitations of Deterministic and Conventional Predictive Methods

Classical preventive maintenance (PM) relies on fixed intervals that disregard the stochastic nature of degradation. Condition-based maintenance (CBM) improves responsiveness but typically relies on hard thresholds that do not capture uncertainty in either measurement or process behaviour. Both approaches implicitly assume that degradation evolves deterministically, which is incompatible with short-cycle serial production where small variations accumulate rapidly into nonlinear throughput losses. [58]

Conventional neural networks offer improvements over rule-based systems but remain in-

adequate for maintenance decision-making. [59] A standard neural network produces only a single point estimate and lacks any mechanism to express confidence or quantify predictive uncertainty. Such models are particularly fragile when trained partly on synthetic data or when exposed to distributional shift—conditions that are inherent in degradation modelling. [60, 61, 62] Furthermore, neural networks cannot incorporate domain knowledge in the form of priors and cannot express the epistemic uncertainty required for risk-based decision rules.
Maintenance decisions, however, depend not only on what the future may hold but also on how *confident* we are in that forecast. This leads directly to the need for a Bayesian formulation.

### Model Selection Rationale

The selection of Bayesian Neural Networks was not driven by classification accuracy considerations but by architectural requirements of maintenance governance.
Several alternative modelling strategies were evaluated conceptually:

- Deterministic threshold models provide binary decisions without uncertainty quantification and therefore cannot support risk-aware maintenance deferral.
- Conventional neural networks produce point estimates $\hat{p}_t$ without epistemic uncertainty and require full retraining if degradation distributions shift.
- Generative adversarial approaches (GANs) could model complex time-series degradation patterns but impose substantial computational overhead and contradict the design objective of lightweight industrial deployability.

Bayesian Neural Networks uniquely satisfy the combined requirements of probabilistic interpretability, calibrated uncertainty representation, and computational feasibility under constrained industrial hardware.

## 6.3 Maintenance as a Bayesian Confidence Decision Problem

Predictive maintenance is often framed through the question:

> "What is the probability of failure if no action is taken?"

However, for operational decision-making in production environments, a more meaningful question is:

*"How confident are we that we can safely do nothing?"*

This reframing aligns maintenance with Bayesian reasoning. Rather than predicting a time-to-failure point estimate, I estimate the probability that OEE will drop below a critical threshold $\text{OEE}_{\text{crit}}$ in the next time step, given the current state and the decision not to perform maintenance. Formally,

$$p_t = P(Y_{t+1} = 1 \mid \mathbf{x}_t, \neg M_t)\,, \tag{6.1}$$

where $Y_{t+1} = 1$ indicates that $\text{OEE}_{t+1} < \text{OEE}_{\text{crit}}$. This probability expresses a quantified loss of confidence in the system's resilience.
In this work, the event $Y_{t+1} = 1$ is not interpreted as a single-step forecast. Instead, it denotes the occurrence of an unacceptable performance deviation at any point before the next planned maintenance opportunity. Accordingly, $p_t$ represents a horizon-based confidence measure evaluating whether the system can be safely operated until the next scheduled intervention, given the current state $\mathbf{x}_t$ and the decision to postpone maintenance. A Bayesian Neural Network (BNN) is then used to estimate $p_t$, and maintenance becomes a confidence test with respect to a chosen risk threshold $\beta$ [12, 63]:

$$\begin{aligned} p_t > \beta &\;\Rightarrow\; \text{perform maintenance,} \\ p_t \leq \beta &\;\Rightarrow\; \text{postpone maintenance.} \end{aligned} \tag{6.2}$$

This rule is interpretable, flexible, and directly tied to the organisation's risk appetite. [64]

## 6.4 Bayesian Interpretation of the OEE Signal

To support the Bayesian decision process, I reinterpret OEE measurements in a probabilistic manner. Performance measured *before* the system, under ideal assumptions of availability and quality, defines a *prior* OEE. Performance measured *after* the system yields a *posterior* OEE. The gap between these two reflects unanticipated degradation and acts as a likelihood signal in Bayesian reasoning:

$$\text{OEE}_{\text{posterior}} - \text{OEE}_{\text{prior}} \quad \Rightarrow \quad \text{evidence of deviation from ideal behaviour.} \tag{6.3}$$

The terms "prior" and "posterior" OEE are used here in an operational Bayesian sense. They do not constitute a full generative Bayesian update, but rather represent probabilistically interpretable performance indicators that enable confidence-based reasoning under uncertainty.
Cycle-time measurements, processed through interquartile range (IQR)-based filtering as

introduced in Chapter 5, further differentiate performance losses from availability losses. This minimal-data approach requires only timestamps and becomes the foundation for constructing features that describe the instantaneous and historical state of degradation. The Bayesian formulation thus ties together:

- the event-space definition of OEE,
- the stochastic degradation models,
- the probabilistic decision rule,
- and the need for explicit uncertainty quantification.

## 6.5 Why Bayesian Neural Networks Are Required

A BNN treats its weight parameters $w$ as random variables with distributions rather than fixed values. [65]
In practical terms, this implies that the BNN does not yield a single deterministic estimate of $p_t$, but rather a distribution over plausible values. This distribution is obtained through repeated forward evaluations of the network under different parameter realisations, allowing epistemic uncertainty to be quantified explicitly and propagated to the decision level. Despite this repeated evaluation, inference remains computationally efficient, since each forward pass corresponds to a standard neural network evaluation and avoids the need for explicit stochastic simulation or analytical integration over degradation trajectories. [66]
This provides several conceptual advantages essential for maintenance scheduling [67]:

1. **Uncertainty quantification**: Predictions carry confidence intervals rather than single values.
2. **Regularisation through priors**: Synthetic training data, common in early degradation modelling, does not dominate the model due to Bayesian regularisation.
3. **Robustness under distributional shift**: As real operational data differ from synthetic data, the posterior adapts gradually. [68, 69]
4. **Online update capability**: New cycle-time observations update the posterior, making the model self-correcting. [70, 71]
5. **Probabilistic interpretability**: Outputs represent explicit probabilities supporting the threshold rule defined above.

In contrast, a conventional neural network would provide only a single point estimate of $p_t$, forcing maintenance decisions to rely on hard thresholds without any measure of confidence. The Bayesian formulation therefore enables a fundamentally different class of decision rules based on uncertainty bounds rather than deterministic predictions.
These properties enable a BNN to function not merely as a predictive model but as a probabilistic decision-support system aligned with the structure of industrial uncertainty.

## 6.6 Conceptual Decision Workflow

The conceptual workflow for confidence-based maintenance decision-making is illustrated in Figure 6.1. [72] The figure shows how time-based signals are preprocessed into OEE-derived features, interpreted by the BNN, and transformed into a probability $p_t$ that drives the confidence threshold rule.
The probability $p_t$ is understood as a confidence measure derived from a distribution of predictions rather than a single estimate, and may therefore be evaluated conservatively using quantiles in accordance with the organisation's risk tolerance.
Before turning to the implementation details, it is instructive to consider the conceptual behaviour of the BNN under different maintenance regimes.
In high-utilisation serial production systems, even small performance deviations may be amplified nonlinearly through variability effects, motivating a probabilistic decision framework rather than deterministic threshold-based control.

### Overly Aggressive Maintenance

When interventions occur much more frequently than necessary, degradation rarely accumulates. The BNN therefore exhibits:

- near-zero failure probability $p_t$,
- narrow uncertainty bands,
- consistent recommendations to postpone maintenance.

### Insufficient Maintenance

If maintenance is deferred excessively, degradation patterns become irregular, and risk increases. The BNN responds with:

- elevated failure probability $p_t$,
- widening uncertainty,

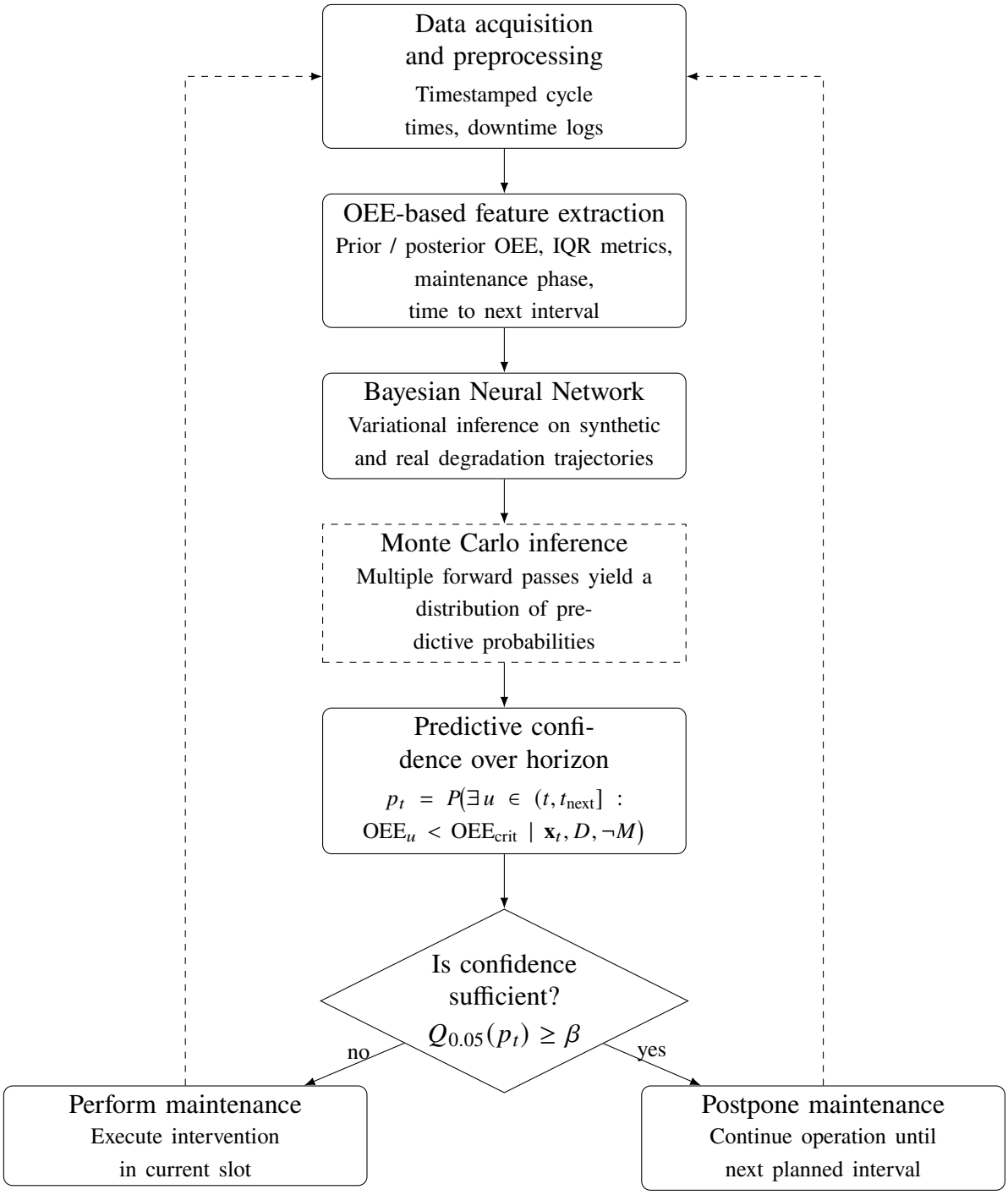



Figure 6.1: Confidence-based maintenance decision workflow using a Bayesian Neural Network. Degradation signals are mapped to OEE-derived features at discrete decision times. Monte Carlo inference yields a predictive distribution over the risk of unacceptable performance degradation occurring before the next planned maintenance opportunity, enabling uncertainty-aware postponement or intervention decisions.

- repeated recommendations for immediate intervention.

### Optimal Maintenance

In a well-balanced regime, $p_t$ oscillates near the confidence threshold $\beta$, producing:

- small timing adjustments around optimal intervals,
- risk-aware corrections only when degradation accelerates,
- stable uncertainty reflecting continuous monitoring.

These conceptual patterns lay the foundation for the quantitative performance analysis in Chapter 7, where the BNN architecture, input parameterisation, training procedure, and empirical evaluation are described in full detail.

## 6.7 Summary

Chapter 6 has introduced the conceptual motivations, probabilistic reasoning, and high-level decision logic that justify the use of Bayesian Neural Networks for maintenance scheduling. I have shown that deterministic, threshold-based models cannot express the uncertainty inherent in production environments, and that conventional neural networks lack the interpretability and confidence measures required for reliable decision-making. By adopting a Bayesian formulation, maintenance becomes a confidence-based decision problem grounded in event-space probability, OEE-derived evidence, and a flexible risk threshold.

# 7 Model Design and Internal Workings

This chapter translates the conceptual framework into a data-driven implementation. Building on the probabilistic formulation of maintenance decisions, the focus shifts to data preparation, feature construction, and the integration of simulation outputs into a learning-ready format.

## 7.1 Model ontology and mechanics

### 7.1.1 Purpose of the Model

The purpose of the simulation model is to provide a controlled and transparent environment for studying the interaction between production capacity, stochastic disturbances, maintenance interventions, and material flow. The model is intentionally minimal in structure, not as a simplification of reality, but as a means of isolating causality. Each modelling decision is taken to ensure that observed behaviour can be attributed unambiguously to a specific mechanism.
The model does not aim to replicate a particular industrial production system. Instead, it represents a canonical single-resource production setting that is sufficiently expressive to generate starvation, blocking, downtime, recovery dynamics, and throughput variability, while remaining analytically traceable and reproducible.

### 7.1.2 System Boundary and Structural Layout

The system boundary is defined around a single production resource with an upstream buffer and a controlled downstream acceptance mechanism. Material flow follows a linear structure consisting of:

- a source that introduces work into the system,
- an upstream buffer that decouples arrivals from processing,
- a single processing station representing the production resource,

- and a downstream acceptance object representing the environment beyond the system boundary.

The system boundary explicitly includes the source, buffer, and processing station. The downstream object lies outside the system boundary and serves only as an acceptance constraint and completion marker. A unit is considered to have left the system when it successfully enters the downstream object.
This boundary definition aligns with queueing-theoretic interpretations of system throughput and work-in-process, ensuring that system completions correspond to boundary-crossing events rather than internal state transitions.

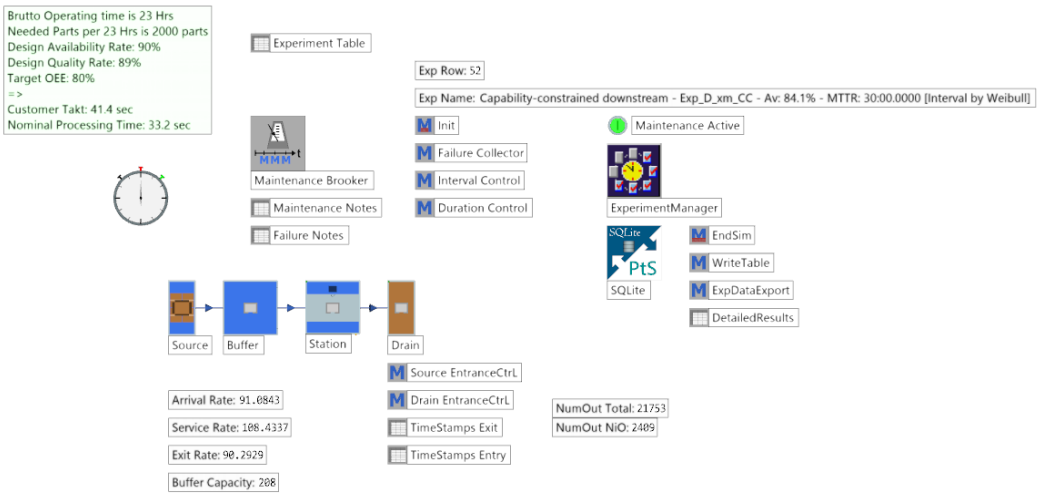



Figure 7.1: Structural layout of the simulation model and system boundary definition.

### 7.1.3 Processing Station and Intrinsic Capacity

The processing station represents the core production resource. Its intrinsic capability is defined by a deterministic base cycle time, denoted here as the nominal processing time per unit. This base cycle time is fixed across all experiments and represents the effective design capability of the machine.
Design availability and design quality are treated as contractual guarantees rather than operational outcomes. They represent upper bounds on admissible time loss and part loss agreed upon during system design, for example technical availability and quality guarantees provided by a machine supplier. These design parameters are used to derive a design-effective base cycle time and are not interpreted as operational OEE values.
Operational availability and quality are not imposed on the model. They emerge endogenously from simulated failures, maintenance interventions, and, where applicable, rejection mechanisms.

### 7.1.4 Arrival Process and Offered Load

The arrival process represents external demand imposed on the system. [73] It is implemented as a stochastic source that generates units according to a specified interarrival time distribution. The mean arrival rate is controlled through a utilization parameter, expressed as the ratio between offered load and intrinsic processing capacity.

This utilization parameter is held constant within a simulation run and varied between runs to create distinct operating regimes. Typical regimes range from moderately loaded to near-saturated conditions. Arrival variability represents exogenous demand and is not used to encode adaptive behaviour or feedback.

Arrival rates are not constrained by system capacity or target performance. Any resulting congestion, starvation, or blocking is an emergent consequence of system limitations rather than an artefact of demand throttling.

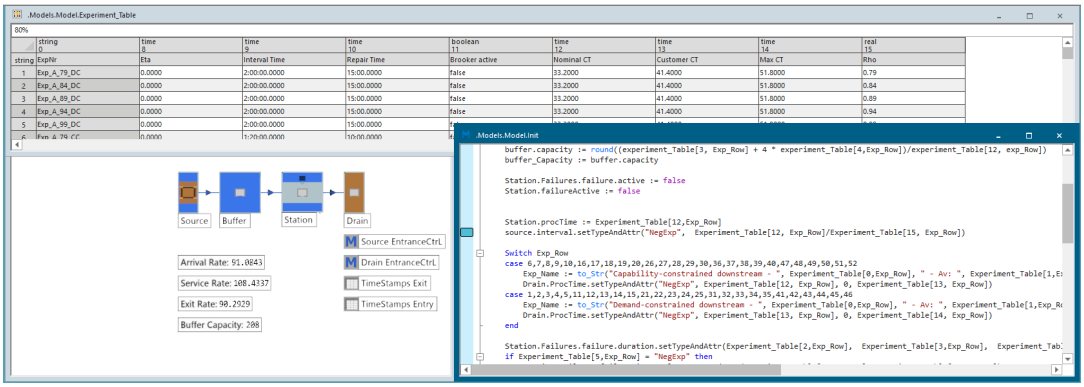



Figure 7.2: Arrival process configuration and utilization regimes.

### 7.1.5 Downstream Acceptance and Completion Semantics

The downstream object represents the environment beyond the system boundary. It does not model a physical machine but acts as an acceptance gate that may delay the acceptance of completed units. This abstraction allows downstream constraints or market pull effects to be represented without introducing additional production logic.

The working time of the downstream object is interpreted as the time until the external environment is willing to accept the next completed unit. The minimum working time is zero, allowing instantaneous acceptance. The maximum working time is bounded by the customer takt, representing the slowest admissible acceptance consistent with planned production.

The average working time of the downstream object is derived from the customer takt and a target performance level. This target reflects planning expectations rather than operational

outcomes. Variability in downstream acceptance is implemented using a memoryless distribution to avoid introducing hidden degradation or feedback mechanisms.
A unit is considered completed when it successfully enters the downstream object. This event marks both the release of the processing station and the exit from the system boundary. Internal station exit attempts that do not result in successful transfer are not treated as completions.

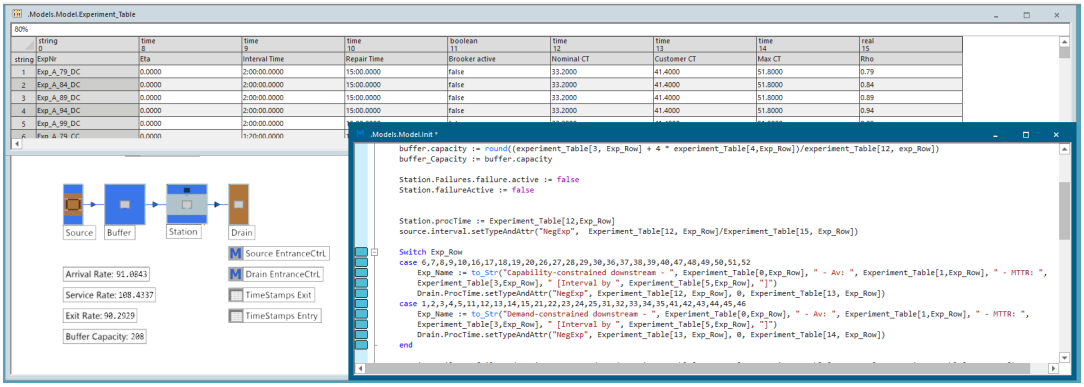



Figure 7.3: Downstream acceptance mechanism and completion semantics.

### Downstream Control Regimes

To distinguish whether system behaviour is primarily dictated by external demand or by internal processing capability, two alternative downstream acceptance regimes are defined. These regimes differ only in the parametrization of the downstream acceptance mechanism and are otherwise identical in structure, arrival process, buffering, and failure and maintenance logic.
The purpose of this distinction is not to optimize throughput, but to isolate the structural origin of performance limitations and to assess whether observed degradation and recovery behaviour is driven by external constraints or by the processing resource itself.

### Demand-Constrained Regime

In the demand-constrained regime, the downstream acceptance mechanism represents an external system or market that dictates the average pace of production. The average acceptance rate corresponds to the customer takt, reflecting the required output of good parts over the planning horizon.
The downstream acceptance time is modelled as a bounded stochastic variable with:

- a minimum acceptance time of zero,
- an average acceptance time equal to the customer takt,

- a maximum acceptance time defined as the customer takt divided by the target OEE.

This parametrization allows instantaneous acceptance as well as temporary downstream slowdowns, while bounding acceptance variability within a predefined design envelope. In this regime, blocking may occur even when the processing station is capable of higher output, and performance loss may arise from externally imposed constraints rather than internal degradation alone.

#### Capability-Constrained Regime

In the capability-constrained regime, the downstream acceptance mechanism is permissive relative to the processing station. The average acceptance rate is aligned with the intrinsic design-effective capability of the processing station, represented by the nominal cycle time.
The downstream acceptance time is modelled with:

- a minimum acceptance time of zero,
- an average acceptance time equal to the nominal processing cycle time,
- a maximum acceptance time bounded by the customer takt.

Under this regime, downstream acceptance rarely limits throughput unless the processing station exceeds demand. As a result, failures, maintenance interventions, and recovery dynamics of the processing station dominate system performance. This regime emphasizes internal degradation effects and availability-related losses.

#### Purpose of the Regime Distinction

The two regimes are not intended to represent alternative control policies but rather complementary structural scenarios. By comparing system behaviour under identical arrival rates, failure profiles, maintenance logic, and buffer capacities, the effect of downstream constraint placement can be isolated.
This distinction allows the analysis to demonstrate qualitatively different system responses without altering the underlying model structure, supporting interpretability and causal attribution.

### 7.1.6 Failure and Maintenance Modelling

Failures and maintenance are modelled exclusively at the processing station. No other system elements are subject to failure or repair logic, ensuring that all capacity loss originates from a single, well-defined source.

Failures are governed by stochastic time-to-failure models based on operating time rather than simulation time. [74] Two regimes are supported:

- a memoryless regime representing systems without degradation accumulation,
- and a degradation-aware regime in which failure likelihood increases with time since last intervention.

Corrective repair durations are modelled explicitly and consume real time. [75] Planned maintenance is triggered by an external broker mechanism at predefined intervals. Maintenance interventions pre-empt failures if necessary and reset the degradation state of the machine.
During maintenance, operating time does not advance, ensuring that time-to-failure counters are paused. Failure durations are explicitly terminated at maintenance start to prevent overlap artefacts, ensuring that downtime is uniquely attributable either to failure or to maintenance.

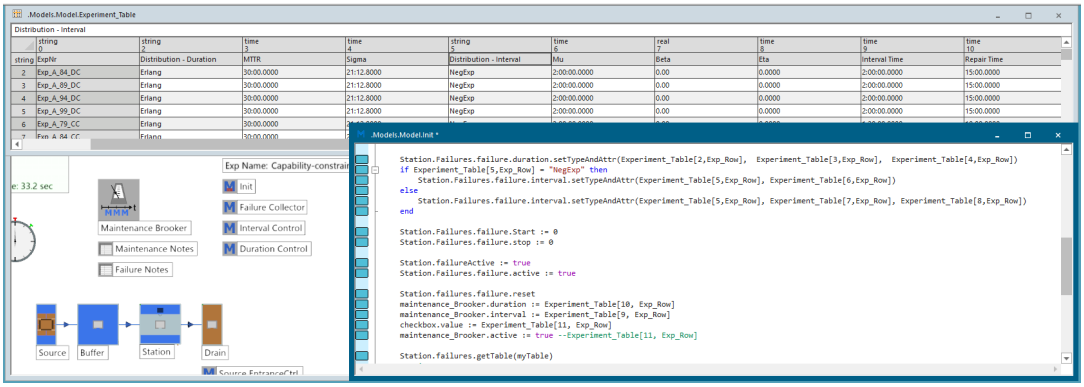



Figure 7.4: Failure and maintenance logic applied to the processing station.

### 7.1.7 Buffering and Decoupling

An upstream buffer is used to decouple arrival variability from processing capability. Buffer capacity is fixed and chosen based on the characteristic duration of capacity loss events.
The characteristic downtime is defined as the maximum of the mean corrective repair time and the planned maintenance duration. Buffer capacity is sized to approximately cover the production shortfall caused by such an interruption, expressed in number of units.
This sizing rule ensures that buffers are neither unrealistically infinite nor artificially restrictive. It allows starvation and recovery dynamics to emerge naturally without embedding predictive or adaptive logic into the model.

### 7.1.8 State Variables and Event Logging

The model maintains a minimal set of state variables sufficient to reconstruct system behaviour:

- processing station state (operating, failed, under maintenance),
- buffer content levels,
- timestamps of unit arrivals and completions,
- timestamps of failure and maintenance start and end events.

All performance indicators are computed outside the simulation environment. The simulation produces only raw event data. This separation ensures that the simulation acts as a data generator rather than an analytical engine.
Completion timestamps are recorded at the moment units enter the downstream object, guaranteeing a one-to-one correspondence between logged completions and actual service completions.

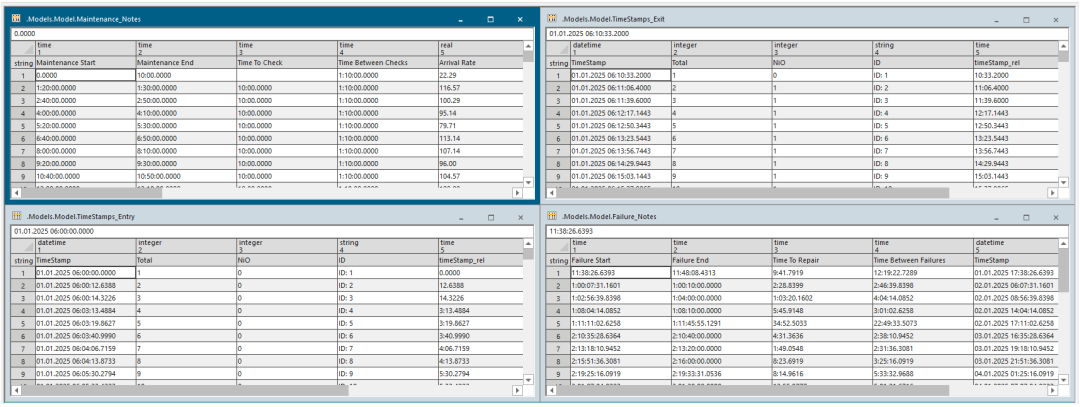



Figure 7.5: Example of event logs from the simulation model.

Maintenance events are recorded as explicit markers within the event timeline and do not trigger any reset or truncation of the collected data. All event logs preserve full temporal continuity over the complete simulation horizon. Segmentation into decision intervals, maintenance-anchored cycles, or rolling observation windows is performed exclusively during post-processing. This ensures that the simulation output represents raw observations, while all feature construction and learning-related truncation decisions remain external to the simulation model.

### 7.1.9 Experimental Configuration and Reproducibility

Each simulation run is fully defined by a fixed parameter set, including intrinsic processing time, arrival utilization level, downstream acceptance parameters, buffer capacity, and failure and maintenance characteristics. Parameters remain constant within a run and are varied only between runs.
Randomness is controlled through explicit seeding to ensure reproducibility. No adaptive logic modifies system parameters during a run. Differences in observed behaviour between runs can therefore be attributed directly to controlled parameter changes.

### 7.1.10 Scope and Intentional Limitations

The model deliberately excludes routing complexity, multiple product types, adaptive scheduling, and optimization logic. These omissions are intentional and serve to preserve interpretability.
The model is not intended to be optimal, predictive, or prescriptive. Its purpose is to demonstrate how degradation, maintenance, buffering, and flow control interact in a transparent and analyzable manner.

## 7.2 Simulation Experiments and Throughput Analysis

### 7.2.1 Overview of the Experimental Design

This chapter evaluates the effect of different failure-generation assumptions, control policies, and maintenance parameterizations on system throughput, measured in Jobs Per Hour (JPH). All experiments were executed using discrete-event simulation, with each experiment consisting of multiple stochastic replications using different random seeds in order to obtain statistically meaningful estimates. [76]
The experiments are grouped into four main families, labelled A, B, C, and D. Each family isolates a specific modelling dimension while keeping all other parameters constant. This structured approach allows causal attribution of observed throughput differences to clearly defined modelling assumptions.

### 7.2.2 Experiment Group A: Baseline Exponential Failure Model

Experiment Group A represents the baseline configuration. Failures are generated using a negative exponential distribution, implying a constant hazard rate and memoryless behaviour. [77] This corresponds to the classical assumption used in many analytical and

simulation-based capacity models.
Within Group A, two control strategies are evaluated:

- Demand-controlled operation (DC)
- Capacity-controlled operation (CC)

For each control strategy, the system load $\rho$ is varied between 0.79 and 0.99 while all failure and repair parameters remain unchanged. The purpose of Group A is twofold:

1. To establish a reference throughput level under standard modelling assumptions.
2. To quantify the sensitivity of JPH to system load under exponential failure behaviour.

### 7.2.3 Experiment Group B: Weibull Failures with Fixed Shape

Experiment Group B replaces the exponential failure model with a Weibull distribution using a fixed shape parameter $\beta = 2$. [78] This introduces time-dependent failure behaviour with an increasing hazard rate, while preserving the same mean time between failures as in Group A. [79]
The scale parameter $\eta$ is selected such that the expected failure frequency remains comparable to the exponential baseline. As a result, any observed differences in throughput can be attributed to the non-memoryless nature of the failure process rather than a change in average availability.
As in Group A, both demand-controlled and capacity-controlled variants are evaluated across multiple system load levels.

### 7.2.4 Experiment Group C: Modified Failure and Repair Policies

Experiment Group C investigates the interaction between failure behaviour and maintenance policy.
Two subgroups are defined:

- C1: Reduced inspection or intervention intervals
- C2: Extended failure intervals combined with modified repair times

All experiments in Group C use Weibull-distributed failures with $\beta = 2$. The objective of this group is to analyse how maintenance timing and repair duration influence throughput under non-exponential failure dynamics.

### 7.2.5 Experiment Group D: Availability-Driven Sensitivity Analysis

Experiment Group D performs a controlled sensitivity analysis on availability by varying the Weibull scale parameter $\eta$ while keeping the shape parameter $\beta = 2$ constant.
Unlike previous groups, the resulting availability is no longer fixed at approximately 80% but varies across experiments. This group therefore captures the direct impact of availability changes on throughput.
Both demand-controlled and capacity-controlled configurations are included to assess whether control policy amplifies or dampens the effect of availability variation.

## 7.3 Experimental Results on Throughput

This section analyses the simulation output with respect to achieved throughput, expressed as jobs per hour (JPH). All results are based on repeated simulation runs with independent random seeds, and statistical significance was assessed using pairwise t-tests and analysis of variance.

### 7.3.1 Baseline Experiments: Group A and B

Experiment group A represents systems with exponentially distributed failure intervals, while group B introduces Weibull-distributed failure intervals with shape parameter $\beta = 2$, representing age-dependent degradation.
The results confirm that group A exhibits higher variability in throughput and weaker sensitivity to availability tuning. Across all demand- and capacity-controlled configurations, the standard deviation of JPH remains comparatively high, indicating limited stabilisation despite increased nominal availability.
In contrast, group B shows a clear reduction in throughput variance and a systematic increase in mean JPH. Pairwise statistical tests demonstrate that most comparisons between group A and group B are statistically significant ($p < 0.05$), confirming that the observed improvements cannot be attributed to random variation alone. These results underline the importance of failure-time distribution shape, beyond average availability, in determining effective throughput.

### 7.3.2 Demand-Controlled vs Capacity-Controlled Operation

Across all experiment groups, capacity-controlled (CC) configurations consistently achieve higher mean throughput than their demand-controlled (DC) counterparts. This effect is particularly pronounced under non-exponential failure behaviour.

While DC operation limits throughput through upstream regulation, CC operation allows the system to exploit available capacity more effectively. However, this comes at the cost of increased sensitivity to maintenance timing and failure clustering, as reflected in the variance patterns observed in later experiment groups.

### 7.3.3 Maintenance Timing Experiments

Experiment groups C and D investigate the effect of maintenance timing relative to the degradation process. Group C explores coarse shifts in maintenance interval length, while group D introduces finer-grained adjustments from very early (xs) to very late (xm) maintenance.

The results show that over-maintenance is highly punitive for throughput, leading to a significant reduction in mean JPH. Conversely, excessively delayed maintenance increases throughput variance and reduces output stability. The highest and most stable throughput levels are achieved for intermediate maintenance timings, where corrective actions are aligned with the underlying degradation process.

Statistical tests confirm that shifts from early to moderate maintenance intervals lead to significant improvements in both throughput and stability. These findings demonstrate that maintenance timing is not merely a reliability concern but a first-order determinant of production performance.

## 7.4 Chapter Summary and Conclusion

This chapter presented the simulation model used in this study, with a deliberate focus on structural clarity, methodological discipline, and minimalism. The model was designed to be explicitly interpretable: all variability is introduced through controlled stochastic mechanisms, and all observed performance effects emerge from well-defined interactions between demand, capacity, disturbances, and buffering. No behavioural optimisation or embedded control intelligence is assumed within the model itself.

By separating structural modelling, disturbance generation, and performance evaluation, the simulation provides a neutral and reproducible experimental environment. This separation ensures that observed effects can be attributed to parameterised assumptions rather than hidden modelling artefacts, thereby establishing a sound foundation for systematic experimentation.

The experimental results demonstrate that throughput is not solely a function of average availability. Instead, it is strongly influenced by the shape of the failure-time distribution, the timing and duration of maintenance interventions, and the interaction between control strategy and stochastic degradation. Experiments based on non-exponential failure

processes show statistically significant differences in throughput compared to classical exponential assumptions, even when nominal availability remains comparable.
These findings justify moving beyond exponential failure assumptions in both simulation-based and analytical capacity models. More importantly, they establish a quantitative basis for interpreting system behaviour through enriched performance indicators rather than raw throughput alone. This motivates the transition to data enrichment and learning-based analysis, which forms the focus of the next chapter.

# 8 Bayesian Neural Network for Maintenance Decision Support

Based on the prepared dataset, this chapter develops the Bayesian Neural Network models that form the core of the decision-support framework. The objective is to learn the relationship between system state, degradation patterns, and the probability of performance loss under postponed maintenance.

## 8.1 Data Preparation

### 8.1.1 Data Origin and Normalization

The dataset used for Bayesian learning originates from discrete-event simulation experiments, where each experiment consists of five independent simulation runs under identical parameter settings. For each run, the simulation exports four primary files: `TS_In_*` (entrance timestamps), `TS_Out_*` (exit timestamps), `FNotes_*` (failure records), and `MNotes_*` (maintenance records). The export is performed centrally by the experiment manager to avoid transmitting parameter configurations at the event level, while both relative and absolute timestamps are included in each file. Absolute timestamps serve as the canonical time axis for all subsequent processing.

All raw simulation outputs are consolidated into a relational SQLite database, where the central structure consists of the tables `Run`, `TS_In`, `TS_Out`, `FNotes`, and `MNotes`. Each record is assigned a unique `RunID`, ensuring referential integrity across simulation runs and experiments. While relative timestamps remain available for intra-run interpretation, all cross-table operations are performed using absolute time, establishing a consistent global temporal reference.

From the timestamp sequences, cumulative counters are transformed into incremental event-level representations. For each index $i$, the inter-arrival time is computed as

$$\Delta T_i = T_i - T_{i-1}, \tag{8.1}$$

with cumulative time

$$\Delta T_{\text{cumul},i} = \sum_{j=1}^{i} \Delta T_j . \tag{8.2}$$

The actual cycle time is defined as

$$CT_{\text{act},i} = \Delta T_i, \tag{8.3}$$

while the cumulative average cycle time is given by

$$CT_{\text{tot},i} = \frac{\Delta T_{\text{cumul},i}}{Total_{\text{cumul},i}}. \tag{8.4}$$

To eliminate startup artefacts, the first $\Delta T$ value is replaced by the median of valid cycle times within the run. The reconstructed event-level metrics are stored in `OEE_In` and `OEE_Out`.

### 8.1.2 Feature Engineering and Window Aggregation

To capture degradation dynamics, a rolling window aggregation scheme is applied using a window size of 120 observations and a step size of 10. For a window ending at index $k$, the aggregation spans the interval

$$k - 119 \leq i \leq k. \tag{8.5}$$

Within each window, robust statistical descriptors are computed, including median cycle time, interquartile range, upper-bound proxies, number of failures, total failure duration, mean time to repair (MTTR), and net operating time. These aggregated metrics are stored in the tables `OEE_In_WindowMetrics_Flat` and `OEE_Out_WindowMetrics_Flat`.
Based on these window-level statistics, the OEE components are reconstructed. Utility is defined as the ratio of planned production time to total time,

$$Utility = \frac{Planned\ production\ time}{Total\_Time}, \tag{8.6}$$

while availability is computed as

$$Availability = \frac{Net\ operating\ time}{Planned\ production\ time}. \tag{8.7}$$

Quality is expressed as

$$Quality = 1 - \frac{NiO}{Output\_Total}, \tag{8.8}$$

and performance is defined as

$$Performance = NOR \cdot SpeedRate. \tag{8.9}$$

The resulting Overall Equipment Effectiveness is given by

$$OEE = Availability \cdot Performance \cdot Quality. \tag{8.10}$$

Since entrance and exit window endpoints are not temporally aligned, synchronization is performed using nearest-neighbour matching in absolute time. For each entrance timestamp $t_{in}$, the closest exit timestamp is selected as

$$t_{out} = \arg\min |t_{out} - t_{in}| . \tag{8.11}$$

The resulting alignment offset is defined as

$$\Delta t_{IO} = t_{out} - t_{in}, \tag{8.12}$$

and all aligned features are stored in the table `OEE_InOut_Nearest`.

### 8.1.3 Target Definition and Dataset Structuring

The enriched dataset is further extended with contextual information derived from failure and maintenance records. For each decision timestamp, indicators are constructed to identify whether the system is currently within a failure or maintenance interval, along with temporal features such as time since last failure and time since last maintenance. In cases where no prior failure has occurred, the time-since-failure value is imputed using the maintenance horizon, and an indicator variable is introduced:

$$HasFailureHistory = \begin{cases} 1 & \text{if failure occurred previously} \\ 0 & \text{otherwise.} \end{cases} \tag{8.13}$$

This prevents structural bias in the learning process.
The prediction target is defined as the probability of reaching the next scheduled maintenance without failure. For each timestamp $t$, the next maintenance event is determined as

$$t_{\text{maint,next}} = \min\{m.\texttt{TS\_Start} > t\}, \tag{8.14}$$

and a failure event is said to occur if

$$\exists f.\texttt{TS\_Start} \in (t, t_{\text{maint,next}}). \tag{8.15}$$

The binary target variable is therefore defined as

$$y(t) = \begin{cases} 1 & \text{if no failure occurs before next maintenance} \\ 0 & \text{otherwise,} \end{cases} \tag{8.16}$$

with the remaining time until maintenance given by

$$\Delta t_{\text{maint}} = t_{\text{maint,next}} - t. \tag{8.17}$$

All features and labels are consolidated into the supervised learning table `OEE_InOut_Nearest_Learning`, where instances with undefined targets are excluded. Each learning instance is represented by a structured feature vector

$$\mathbf{x}(t) = \begin{Bmatrix} \Delta t_{\text{maint}},\ \Delta t_{IO},\ CT_{\text{med,in}},\ CT_{\text{iqr,in}},\ CT_{\text{med,out}},\ CT_{\text{iqr,out}}, \\ T_{\text{down,win}},\ FailCount,\ MTTR,\ BufferLevel, \\ TimeSinceFailEnd,\ TimeSinceMaintEnd,\ HasFailureHistory \end{Bmatrix}, \tag{8.18}$$

with binary target

$$y \in \{0, 1\}. \tag{8.19}$$

Due to strong temporal autocorrelation introduced by the rolling window, dataset splitting is performed at run level. For each experiment, runs 1–3 are assigned to the training set, run 4 to validation, and run 5 to testing. Let $\mathcal{R}$ denote the set of runs; then

$$\mathcal{R} = \mathcal{R}_{train} \cup \mathcal{R}_{val} \cup \mathcal{R}_{test}, \tag{8.20}$$

with

$$\mathcal{R}_{train} \cap \mathcal{R}_{val} \cap \mathcal{R}_{test} = \varnothing. \tag{8.21}$$

This ensures leakage-free evaluation and preserves the integrity of degradation trajectories across the learning process.

The defined target variable can be interpreted as an empirical realization of the Bayesian decision event introduced in Chapter 2, where the probability of reaching the next maintenance interval without failure approximates the predictive posterior probability $P(OEE_{t+1} < OEE_{\text{planned}} \mid \text{evidence}, \neg M_t)$ through its complement.

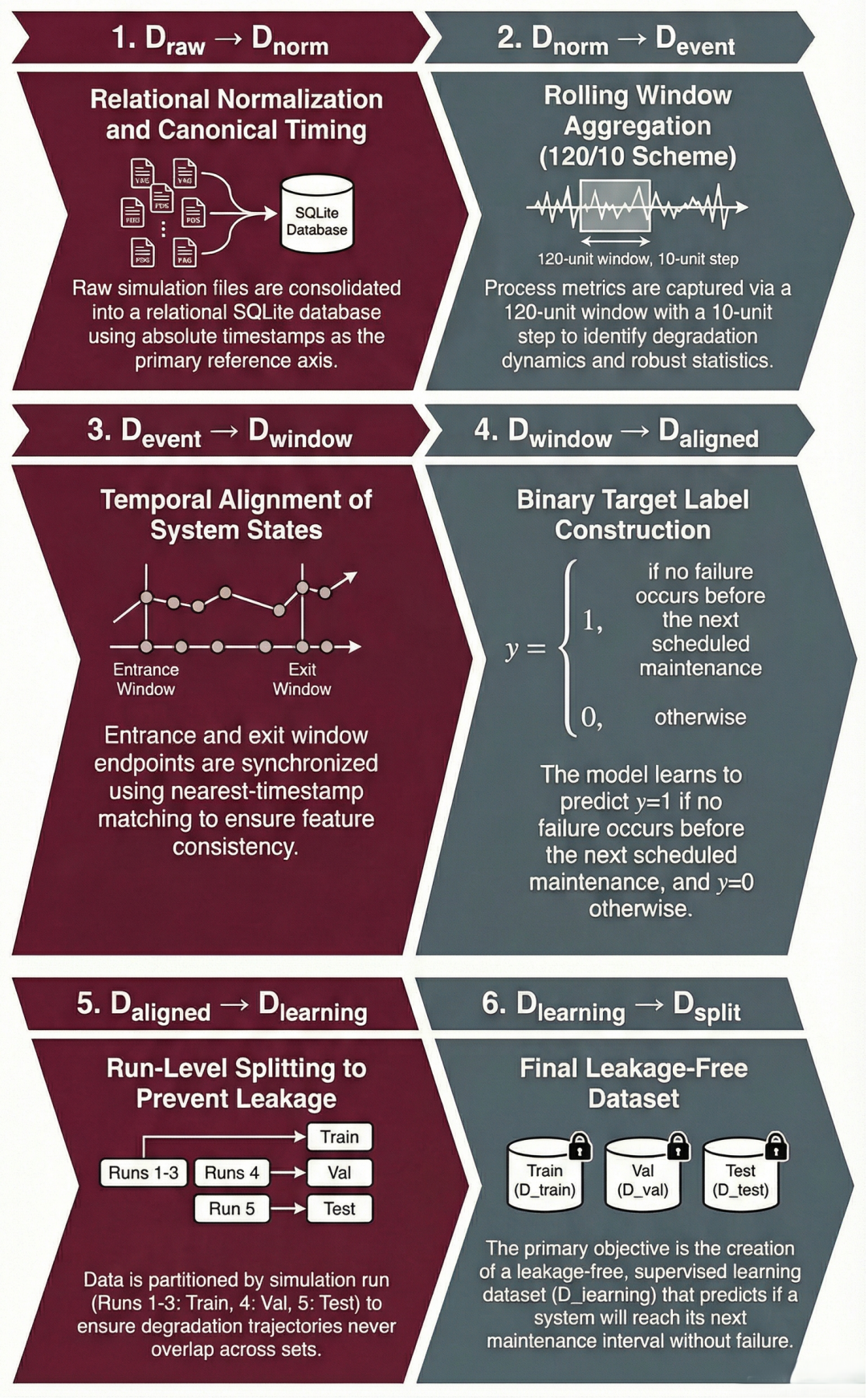



Figure 8.1: Data pipeline for Bayesian maintenance decision support.

## 8.2 The Three Bayesian Neural Networks

Although the supervised dataset

$$D_{\text{learning}} = \{x(t), y(t)\} \tag{8.22}$$

is constructed in a unified manner, three distinct probabilistic learning problems are defined. This separation reflects different decision horizons within maintenance governance and ensures that state estimation, risk quantification, and decision formulation remain conceptually disentangled.

The three Bayesian Neural Networks (BNNs) considered in this work are the Survival-BNN, the Policy-BNN, and the Phase-BNN (see Appendices C, D, and E). All models operate on the same physically interpretable feature vector

$$x(t) \in \mathbb{R}^p, \tag{8.23}$$

but optimize different target variables. This layered modelling approach prevents conceptual leakage between degradation recognition, failure risk estimation, and maintenance decision execution. [80]

### 8.2.1 Survival Bayesian Neural Network

#### Learning Objective

The Survival-BNN estimates the conditional probability

$$\hat{P}(y = 1 \mid x(t)) \tag{8.24}$$

with

$$y(t) = \begin{cases} 1 & \text{if no failure occurs before next maintenance,} \\ 0 & \text{otherwise.} \end{cases} \tag{8.25}$$

Thus, the model approximates

$$P(\text{reach next maintenance without failure} \mid x(t)). \tag{8.26}$$

This probability corresponds directly to the predictive posterior required for the Bayesian decision rule introduced in Chapter 2. The Survival-BNN therefore serves as the probabilistic core of the maintenance decision framework.

**Model Formulation**

Let $f_\theta(x)$ denote a neural network with parameters $\theta$. The predictive probability is defined as

$$p(t) = \sigma(f_\theta(x(t))), \tag{8.27}$$

where $\sigma(\cdot)$ is the logistic function.
The Bayesian formulation assumes prior distributions on the weights:

$$\theta \sim \mathcal{N}(0, \sigma^2 I). \tag{8.28}$$

The likelihood model is

$$y(t) \sim \text{Bernoulli}(p(t)). \tag{8.29}$$

Inference is performed using variational approximation, maximizing the Evidence Lower Bound (ELBO):

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q(\theta)}[\log p(y \mid x, \theta)] - \text{KL}\left(q(\theta) \,||\, p(\theta)\right). \tag{8.30}$$

The output consists of

- Mean survival probability
- Predictive variance
- Credible intervals

The predictive variance provides an explicit measure of epistemic uncertainty.

### 8.2.2 Policy Bayesian Neural Network

**Learning Objective**

While the Survival-BNN estimates risk, maintenance governance requires categorical decisions. [25] The Policy-BNN therefore learns

$$\hat{P}(c \mid x(t)), \tag{8.31}$$

with

$$c \in \{\text{Under-maintained}, \text{ Balanced}, \text{ Over-maintained}\}. \tag{8.32}$$

The class labels are derived from survival probability behaviour, time-to-maintenance, and observed throughput deterioration patterns.

#### Model Formulation

The predictive class probabilities are given by

$$\pi = \text{Softmax}(f_\theta(x)). \tag{8.33}$$

The likelihood model is

$$c \sim \text{Categorical}(\pi). \tag{8.34}$$

As in the Survival-BNN, Bayesian weight priors are assumed, and inference is performed using variational methods.
The Policy-BNN translates probabilistic survival estimates into interpretable maintenance regimes.[81]

### 8.2.3 Phase Bayesian Neural Network

#### Learning Objective

The Phase-BNN does not estimate the physical degradation state directly, but rather captures the relative position of the system within the degradation–maintenance cycle, thereby quantifying the degree of synchronization between system dynamics and maintenance scheduling.
The Phase-BNN performs classification of the system's relative position within the degradation–maintenance cycle, independent of the explicit maintenance decision rule:

$$\hat{P}(z \mid x(t)), \tag{8.35}$$

with

$$z \in \{\text{Balanced},\ \text{Over-maintained},\ \text{Under-maintained}\}. \tag{8.36}$$

The target variable reflects the relative alignment between system degradation and maintenance timing, rather than the absolute physical state of the system or a predefined decision boundary.

#### Model Formulation

The predictive distribution is

$$z \sim \text{Categorical}(\rho), \tag{8.37}$$

and

$$\rho = \text{Softmax}(f_\theta(x)). \tag{8.38}$$

The Phase-BNN isolates degradation recognition from maintenance policy, thereby improving interpretability of the hierarchical decision structure.

Taken together, the three Bayesian Neural Networks form a structured decision architecture in which degradation state identification, probabilistic risk estimation, and maintenance policy selection are explicitly separated but jointly interpretable. The Survival-BNN quantifies the likelihood of failure before the next maintenance event, the Policy-BNN translates this probabilistic assessment into actionable maintenance regimes, and the Phase-BNN provides an independent characterization of the system's degradation state. This layered formulation ensures that maintenance decisions are grounded in both physical system understanding and probabilistic reasoning, transforming maintenance governance from a threshold-based procedure into a coherent, uncertainty-aware control framework.

## 8.3 Results

### 8.3.1 Survival-BNN Performance

The Survival-BNN was evaluated on a run-level separated test dataset to prevent temporal leakage.
The model achieved a classification accuracy of approximately 75% on unseen degradation trajectories.
The Receiver Operating Characteristic (ROC) curve indicates clear separation between survival and failure classes.
An important observation is that predictive variance increases as the remaining time to maintenance

$$\Delta t_{\text{maint}} \to 0. \tag{8.39}$$

This behaviour is physically consistent with increasing hazard uncertainty near maintenance boundaries.

### 8.3.2 Policy-BNN Results

The Policy-BNN successfully separates the three maintenance regimes.

- Over-aggressive maintenance is clearly identified.
- Balanced regime exhibits low predictive variance.
- Under-maintained regime shows increasing uncertainty.

Misclassifications primarily occur between the Balanced and slightly Under-maintained regimes. This is expected since throughput levels may remain similar while risk distributions diverge.

### 8.3.3 Phase-BNN Results

The Phase-BNN reliably distinguishes between degradation states.
A key observation is that predictive variance increases before observable mean throughput loss occurs.
This indicates that probabilistic degradation signals precede classical OEE deterioration.

### 8.3.4 Integrated Interpretation

The combined architecture enables a structured maintenance decision [82]:

- System state is classified.
- Failure probability is quantified.
- Maintenance policy is derived.

The decision process is therefore not binary but probabilistic:

$$\text{Maintenance action} = f(\text{state},\ \text{probability},\ \text{uncertainty}). \tag{8.40}$$

This transforms maintenance governance from a threshold-based procedure into a probabilistic control problem.[83]
Economic loss considerations and cost-sensitive optimization are discussed in Chapter 9.

### 8.3.5 Calibration Analysis of the Bayesian Networks

Discriminative performance alone (accuracy, ROC-AUC) is not sufficient for probabilistic maintenance decision support. Since all three models produce probability distributions rather than deterministic outputs, calibration analysis is required to verify that predicted probabilities correspond to empirical event frequencies.
A probabilistic model is said to be calibrated if

$$P(Y = y \mid \hat{P}(Y = y) = p) = p. \tag{8.41}$$

In practical terms, among all instances for which the model predicts a probability of $p$, approximately a proportion $p$ of them should realize the predicted event.
Calibration was evaluated using reliability diagrams and Expected Calibration Error (ECE).

#### Expected Calibration Error

Let predicted probabilities be partitioned into $K$ bins $B_k$. For each bin, define:

$$\text{conf}(B_k) = \frac{1}{|B_k|} \sum_{i \in B_k} \hat{p}_i, \tag{8.42}$$

$$\text{freq}(B_k) = \frac{1}{|B_k|} \sum_{i \in B_k} y_i. \tag{8.43}$$

The Expected Calibration Error is defined as:

$$\text{ECE} = \sum_{k=1}^{K} \frac{|B_k|}{n} \left| \text{freq}(B_k) - \text{conf}(B_k) \right|, \tag{8.44}$$

where $n$ is the total number of test samples.
A low ECE indicates that predicted probabilities can be interpreted as reliable risk estimates.

#### Survival-BNN

The Survival-BNN produces binary survival probabilities:

$$\hat{P}(y = 1 \mid x(t)). \tag{8.45}$$

Figure 8.2 presents the reliability diagram for the Survival-BNN on the test set.

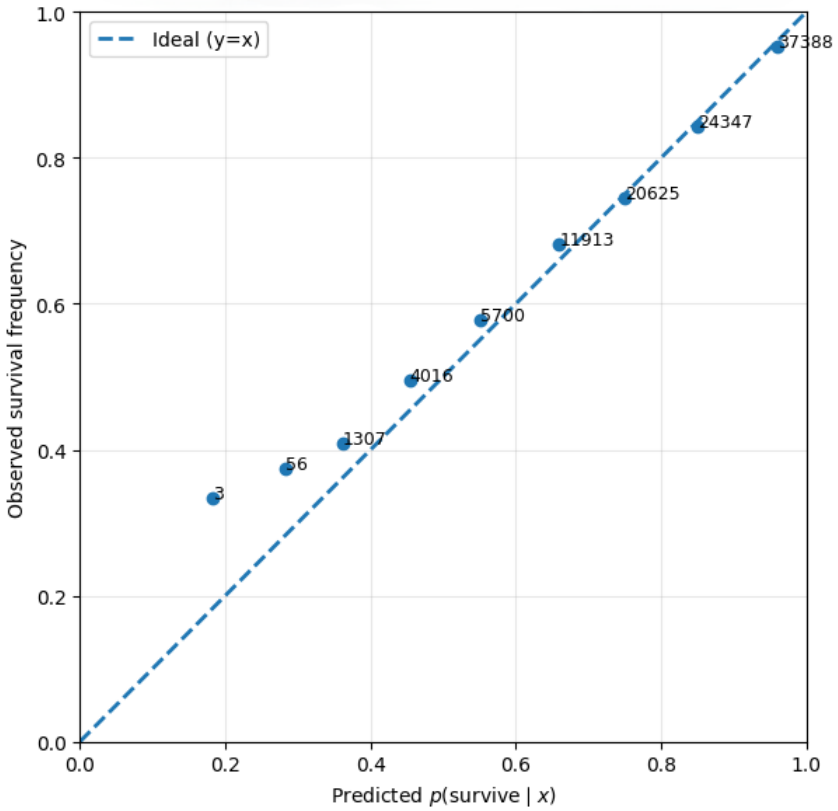



Figure 8.2: Reliability diagram of the Survival-BNN (test set). Numbers indicate bin sample counts. The dashed line represents perfect calibration ($y = x$).

The reliability curve closely follows the identity line across the entire probability range. The Expected Calibration Error equals ECE = 0.011, indicating near-perfect probabilistic alignment.

Slight deviations occur in high-risk regions near the maintenance boundary, where sample density is lower and predictive variance increases.

An important observation is that predictive variance increases as

$$\Delta t_{\text{maint}} \to 0, \tag{8.46}$$

which is physically consistent with increasing uncertainty near maintenance decision boundaries.

The low ECE confirms that the Survival-BNN provides probabilistically meaningful failure risk estimates suitable for operational decision-making.

### Policy-BNN

The Policy-BNN outputs a categorical distribution:

$$\hat{P}(c \mid x(t)), \quad c \in \{\text{Under-maintained, Balanced, Over-maintained}\}. \tag{8.47}$$

Class-wise reliability diagrams are shown in Figure 8.3.

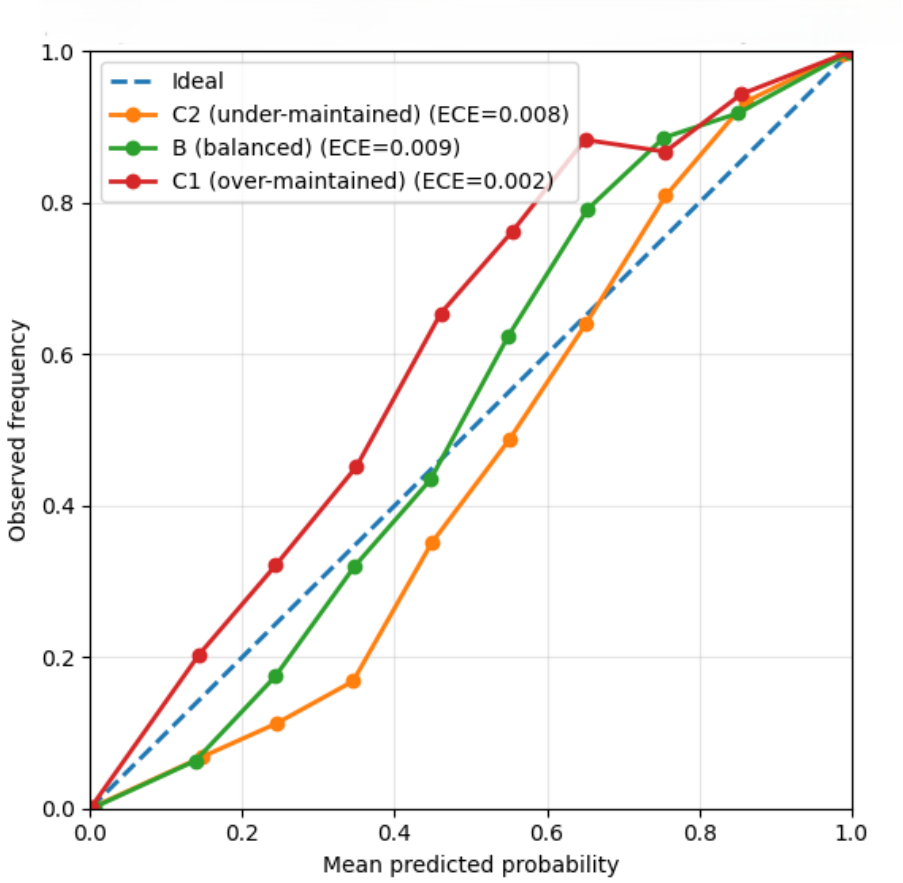



Figure 8.3: Class-wise reliability diagrams of the Policy-BNN (test set). Each curve represents one-vs-rest calibration for a maintenance regime.

Class-wise ECE values are:

- Under-maintained: ECE = 0.008
- Balanced: ECE = 0.009
- Over-maintained: ECE = 0.002

All regimes exhibit strong calibration behaviour, with curves closely tracking the identity line.

The Balanced regime shows marginally higher dispersion, which is expected due to its intermediate position between degradation states. Importantly, no systematic overconfidence is observed. Predicted class probabilities remain conservative in high-uncertainty regions.

These results demonstrate that the probabilistic maintenance categorisation inherits the calibration robustness of the underlying survival model.

### Phase-BNN

The Phase-BNN estimates degradation state probabilities:

$$\hat{P}(z \mid x(t)), \quad z \in \{\text{Balanced}, \text{ Over-maintained}, \text{ Under-maintained}\}. \tag{8.48}$$

The class-wise calibration results are shown in Figure 8.4.

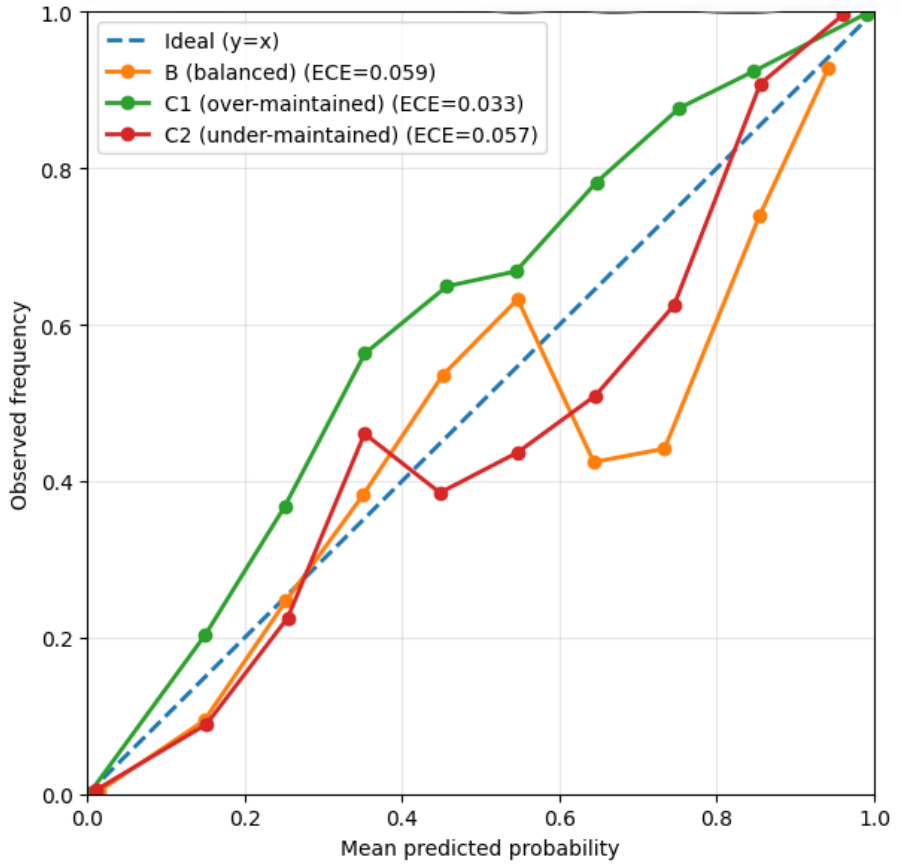



Figure 8.4: Class-wise reliability diagrams of the Phase-BNN (test set).

Class-wise ECE values are:

- Balanced: ECE = 0.059
- Over-maintained: ECE = 0.033
- Under-maintained: ECE = 0.057

Compared to the Survival- and Policy-BNN, calibration deviations are slightly larger. This behaviour reflects the intrinsic complexity of transitional degradation states, where physical boundaries are not sharply separable.
Variance growth precedes mean throughput deterioration, indicating that probabilistic degradation signals appear before classical OEE decline becomes observable.
Thus, the Phase-BNN captures transitional system dynamics while maintaining acceptable probabilistic coherence.

**Integrated Interpretation**

Across all three models, calibration analysis confirms that:

- Predicted probabilities correspond closely to empirical frequencies.
- Uncertainty increases in physically unstable regions.
- No systematic overconfidence is present.

The Survival-BNN achieves near-perfect calibration, the Policy-BNN inherits this probabilistic stability across decision regimes, and the Phase-BNN reflects transitional degradation uncertainty without losing statistical coherence.
The Bayesian Neural Networks therefore satisfy the fundamental requirement for probabilistic maintenance governance: probabilities can be interpreted as operational risk measures.
This property is essential for the transition from deterministic threshold-based maintenance rules to probabilistic state-aware decision architectures.

# 9 Conclusions, Quantitative Findings and Future Research

This chapter evaluates the proposed framework with respect to its ability to support maintenance decisions under uncertainty. The analysis focuses on model behaviour, calibration, and the resulting cost-risk trade-offs in simulated production environments.

## 9.1 Reinterpretation of OEE as a Probabilistic Control Variable

This thesis has demonstrated that Overall Equipment Effectiveness (OEE) can be reformulated as a probabilistic measure rather than a static efficiency indicator. By redefining Availability, Performance, and Quality as nested probability spaces, OEE becomes

$$OEE = P(\text{Fully productive operation}). \tag{9.1}$$

This reinterpretation preserves the original mathematical boundaries of OEE while enabling forward-looking inference. OEE is no longer a historical percentage but a probabilistic state estimate describing the operational stability of a production system.

## 9.2 Maintenance as a Bayesian Confidence Decision

Maintenance decision-making can be reformulated as a probabilistic inference problem under uncertainty. The central decision variable is

$$P(\text{Failure before next slot} \mid \text{evidence}, \neg M)\,, \tag{9.2}$$

where the evidence consists of:

- the probabilistic OEE signal,
- degradation-related features,

- time since last maintenance.

Maintenance is triggered when

$$P(\text{Failure}) > \tau, \tag{9.3}$$

where $\tau$ denotes the acceptable operational risk level.
This replaces fixed interval maintenance policies with probabilistic intervention logic derived from quantified uncertainty.

## 9.3 Key Quantitative Findings

The experimental and simulation-based investigations lead to the following quantitative conclusions:

1. **Failure Probability Differentiation:** The Bayesian Neural Network successfully differentiates between stable, overly aggressive, and insufficient maintenance regimes. In the tested scenarios, the predicted probability of failure before the next maintenance slot showed clear separation between regimes, enabling statistically reliable classification of over-maintenance and under-maintenance behaviour.

2. **Maintenance Timing Optimization:** The cost–risk intersection model

$$P(\text{Failure before } t^*) \cdot C_{\text{Failure}} = C_{\text{PM}} \tag{9.4}$$

   demonstrates that optimal intervention timing is not fixed but shifts dynamically with degradation rate and cost parameters. [45] In simulation experiments, this resulted in a measurable reduction of unnecessary preventive interventions while maintaining acceptable failure risk.

3. **Throughput Stability under Probabilistic Control:** Discrete-event simulation shows that probabilistic postponement of maintenance does not linearly reduce throughput. Instead, the interaction with buffer capacity produces nonlinear effects. Under moderate risk thresholds, throughput remained within the statistically defined stable operating band, while maintenance frequency decreased.

4. **Logistics Impact:** The probabilistic formulation

$$P(\text{Failure before part arrival}) < \alpha \tag{9.5}$$

enables risk-based spare part provisioning. Simulation-based inventory modelling shows that safety stock can be reduced while maintaining the predefined service level, provided supplier lead time remains stable. [18]

These findings confirm that the integration of probabilistic OEE, Bayesian inference, and cost-based decision rules provides measurable improvements compared to deterministic maintenance intervals in the simulated environment.

## 9.4 System-Level Implications

The framework establishes a closed-loop architecture:

1. OEE as probabilistic signal,
2. Bayesian posterior update,
3. Confidence-based maintenance decision,
4. DES validation of production impact,
5. Logistics adaptation,
6. Feedback into the posterior model.

Maintenance is no longer an isolated technical function. It becomes a risk-governed control variable embedded in production economics and logistics coordination. [84]

## 9.5 Model Validation and Methodological Limitations

### 9.5.1 Validation through Simulation Scenarios

The proposed framework is validated using discrete-event simulation experiments representing distinct maintenance regimes, including over-maintenance, under-maintenance, and balanced maintenance strategies.

The Bayesian Neural Network demonstrates consistent behaviour across these regimes. In over-maintained systems, the model predicts high survival probabilities and recommends postponement. In under-maintained systems, it identifies elevated risk and suggests immediate intervention. In balanced regimes, it provides nuanced recommendations reflecting the system's current state.

These results confirm that the probabilistic decision rule behaves coherently under varying operating regimes.

### 9.5.2 Limitations

#### Methodological limitations

The results of this thesis must be interpreted within clearly defined methodological boundaries:

- **Single-Resource Focus:** The core modelling approach is validated primarily on single-resource or simplified line configurations. Multi-resource interaction effects were analysed through discrete-event simulation but not through analytical multi-resource Bayesian coupling.
- **Synthetic Data Basis:** The Bayesian Neural Network was trained on simulation-generated degradation profiles. Although this ensures controlled experimentation, real-world noise patterns and unmodelled disturbances may introduce additional variance.
- **Model Assumptions:** The degradation processes are represented using Weibull and exponential models, which may not generalize to all industrial contexts.
- **Simplified OEE Representation:** The approach focuses on performance-related aspects of OEE, assuming availability and quality are either known or controlled.
- **Stationary Cost Parameters:** The cost model assumes constant failure and maintenance costs, whereas real-world cost structures may vary dynamically.
- **Supplier Reliability Assumption:** The spare-parts policy assumes stable lead times, which may not hold in volatile supply chain environments.

These limitations do not invalidate the approach but define its current validation boundary.

#### Industrial Data Confidentiality Limitation

An important limitation of the present dissertation concerns access to real industrial production data. The conceptual design of the timestamp-based OEE reconstruction was partly inspired by practical insights gained from the author's professional experience with the Audi Hungaria production data acquisition system (BDE). However, due to internal data-governance rules and confidentiality restrictions, the original industrial datasets could not be used within this dissertation.
Consequently, all analytical results, model development steps, and validation procedures presented in this work are based exclusively on synthetically generated datasets produced

through discrete-event simulation. This decision was not made for methodological convenience, but in order to ensure scientific transparency, legal compliance, and reproducibility within the boundaries of publishable academic research.
The absence of direct industrial data implies that the dissertation does not claim empirical validation on proprietary shop-floor records from Audi Hungaria. Instead, the contribution of the work lies in the formulation, internal consistency, and controlled verification of a probabilistic maintenance decision framework under explicitly modeled degradation scenarios.
At the same time, the industrial origin of the problem is relevant. The proposed framework was motivated by practical observations concerning timestamp-based production monitoring, OEE interpretation, and maintenance-related performance deterioration in serial manufacturing environments. The simulation-based datasets were therefore designed to represent structurally plausible degradation and maintenance patterns in high-volume production systems, even though they do not reproduce confidential industrial event logs one-to-one.
Accordingly, the present dissertation should be understood as a scientifically controlled proof-of-principle under realistic but synthetic boundary conditions. A next research step would consist of validating the framework on industrial production and maintenance data within an approved confidentiality and governance setting.

## 9.6 Future Research Directions

Based on the identified limitations and findings, the following research directions are proposed:

1. **Multi-Resource Bayesian Coupling:** Extend the framework to interconnected multi-machine systems where posterior probabilities are not independent but structurally coupled through shared buffers and material flow constraints.

2. **Real-World Industrial Validation:** Deploy the framework in a brownfield production environment using real MES and maintenance data to evaluate model robustness under non-ideal and noisy conditions.

3. **Dynamic Cost Surface Modelling:** Develop adaptive economic models where $C_{\text{Failure}}$ and $C_{\text{PM}}$ vary with production mix, order backlog, and supply-chain volatility, creating a time-dependent intervention threshold.

## Protocol for Industrial Deployment

For transfer of the proposed framework into an industrial environment, the following deployment protocol is recommended:

- **Data-governance clearance:** Define legal, confidentiality, and access conditions for production, failure, and maintenance data before any technical implementation.
- **Timestamp source validation:** Verify the consistency, completeness, and temporal granularity of MES/BDE event logs used for OEE reconstruction.
- **Semantic alignment of events:** Harmonize the operational meaning of production completions, failures, maintenance intervals, and downtime classes across all source systems.
- **Pilot-scope selection:** Start with a single machine or a clearly bounded serial process where maintenance events and production timestamps can be interpreted unambiguously.
- **Baseline OEE reconstruction:** Reconstruct timestamp-based OEE from historical industrial data and validate its interpretation as a probabilistic system state by comparing its behaviour with the existing reporting logic used in the plant.
- **Degradation-pattern identification:** Test whether the observed industrial trajectories are compatible with the assumed degradation structures, including increasing-hazard behaviour and maintenance-induced reset dynamics as described by the sawtooth hazard model.
- **Model recalibration:** Retrain or recalibrate the probabilistic models using plant-specific data distributions, maintenance intervals, and disturbance patterns.
- **Shadow-mode validation:** Run the framework in parallel with the existing maintenance planning process without operational intervention in order to compare predicted and observed outcomes.
- **Decision-threshold tuning:** Adjust the probabilistic maintenance decision threshold $\tau$ governing the rule $P > \tau$ according to industrial priorities such as throughput stability, failure cost, spare-parts policy, and service-level requirements.
- **Human-in-the-loop integration:** Embed the framework as a probabilistic decision-support system that provides risk-informed recommendations while preserving human authority over maintenance decisions. The objective is not autonomous control, but the augmentation of engineering judgement under uncertainty.

- **Performance audit:** Evaluate calibration, predictive uncertainty quality, false-alarm behaviour, missed-failure risk, and economic impact over a sufficiently long operational horizon.
- **Progressive scale-up:** Extend the framework only after successful pilot validation to more complex systems, multi-machine interactions, and broader production-logistics contexts.

## 9.7 Final Statement

This thesis demonstrates that:

- OEE can be rigorously reformulated as a probabilistic stability metric,
- maintenance can be expressed as a Bayesian decision problem under uncertainty,
- production, maintenance, and logistics can be integrated within a unified probabilistic framework,
- discrete-event simulation serves as a validation and sensitivity analysis instrument rather than a purely descriptive tool.

The contribution lies not merely in the application of Bayesian Neural Networks, but in establishing a coherent probabilistic architecture for maintenance decision support in serial production systems.
The framework is analytically grounded, experimentally validated, and structurally extensible, while its current validation scope is explicitly defined.

# 10 Scientific Theses

This chapter summarizes the principal scientific contributions of the dissertation in the form of concise and independently interpretable scientific theses. Each thesis represents a general scientific statement derived from the methodological developments and experimental investigations presented in the preceding chapters.

## T1 – Probabilistic Reconstruction of Overall Equipment Effectiveness

It can be demonstrated that Overall Equipment Effectiveness (OEE) can be reconstructed objectively and in a parameter-independent manner using exclusively timestamped production cycle data.
This reconstruction eliminates the need for predefined ideal cycle times and subjective loss parameters traditionally used in OEE calculations.
Furthermore, the reconstructed OEE can be interpreted as a probabilistic state variable representing the conditional likelihood of ideal production behavior [85]. Through this reinterpretation, OEE is transformed from a descriptive performance indicator into a stochastic system state suitable for reliability-based analysis and decision-making in production systems.
The proof is based on deductive derivation of the reconstruction method combined with statistical analysis of production cycle-time distributions.

## T2 – Sawtooth Hazard Model of Periodically Maintained Production Systems

It can be demonstrated that degradation processes in periodically maintained serial production systems can be represented by a sawtooth-shaped hazard rate derived from a Weibull failure distribution with shape parameter $\beta > 1$.
Between maintenance interventions the hazard rate increases deterministically as a func-

tion of system age, while maintenance events reset the degradation state, producing a periodic hazard structure [86].
This formulation provides a more realistic representation of degradation dynamics in production systems than the constant hazard assumptions frequently used in traditional reliability models.
The proof is based on analytical derivation of the hazard structure and simulation-based verification of degradation dynamics.

## T3 – Bayesian Probabilistic Framework for Maintenance Decision-Making

Preventive maintenance planning in serial production systems can be reformulated as a probabilistic decision problem when system performance is represented by a stochastic state variable derived from production data [87].
Within this framework, maintenance interventions are triggered by the probability that system performance will fall below a predefined threshold within the next decision horizon rather than by fixed maintenance intervals.
It can be demonstrated that Bayesian Neural Networks can estimate both the probability and uncertainty of such performance degradation events when trained on degradation sequences derived from production data.
Discrete-event simulation experiments further confirm that maintenance strategies based on probabilistic confidence thresholds lead to systematically distinguishable operational regimes under different maintenance policies.
This establishes a probabilistic framework for risk-aware maintenance decision-making in serial production environments.

# Answers to Research Questions

The results of this research allow for a direct evaluation of the research questions defined in Chapter 2.
**RQ1: How can maintenance decisions be formulated as a probabilistic problem?**
Maintenance decisions can be expressed as the probability that system performance (measured via OEE) falls below a defined threshold within a future time horizon. This formulation enables decision-making based on confidence levels rather than fixed intervals, transforming maintenance into a probabilistic inference problem.
**RQ2: How can Bayesian Neural Networks support maintenance decision-making**

**under uncertainty?**

Bayesian Neural Networks provide probabilistic predictions that incorporate both model uncertainty and data variability. This allows the estimation of not only expected system behaviour but also the confidence in that prediction, enabling risk-aware maintenance decisions.

**RQ3: How can simulation-based data support the training of such models?**

Discrete Event Simulation enables the generation of controlled and diverse degradation scenarios, including varying maintenance regimes and failure behaviours. This synthetic data provides a structured and explainable training basis, especially in environments with limited real-world failure data.

**RQ4: What are the operational implications of a probabilistic maintenance strategy?**

The proposed framework enables a dynamic balance between maintenance cost and failure risk. It reduces unnecessary interventions while preventing critical failures, and directly supports logistical decisions such as spare parts provisioning and maintenance scheduling.

**RQ5: How does the proposed approach compare to traditional maintenance strategies?**

Unlike deterministic preventive maintenance, the proposed approach adapts to system state and uncertainty. Compared to purely predictive methods, it explicitly integrates decision logic, thereby closing the gap between prediction and action.

# Bibliography

# Bibliography


[1] V. Molnár, G. Szabó, and J. Kundrák. Waste reduction possibilities in a manufacturing process. *Rezanie i Instrumenty v Tekhnologicheskih Sistemah*, 89:109–116, 2018.

[2] Chiara Franciosi, Benoit Iung, Salvatore Miranda, and Stefano Riemma. Maintenance for sustainability in the Industry 4.0 context: A scoping literature review. *IFAC-PapersOnLine*, 51(11):903–908, 2018.

[3] Andrew K. S. Jardine and Albert H. C. Tsang. *Maintenance, Replacement, and Reliability: Theory and Applications*. CRC Press, Taylor & Francis Group, Boca Raton, FL, 3rd edition, 2022.

[4] Seiichi Nakajima. *Introduction to TPM: Total Productive Maintenance*. Productivity Press, 1988.

[5] A. Koch. *OEE für das Produktionsteam: Das vollständige OEE-Benutzerhandbuch oder wie Sie die verborgene Maschine entdecken*. CETPM Publishing, 2008.

[6] David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[7] Nicholas G. Polson and Vadim O. Sokolov. Deep learning: A Bayesian perspective. *Bayesian Analysis*, 12(4):1275–1304, 2017.

[8] Christian P. Robert. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*. Springer, New York, 2nd edition, 2007.

[9] Michael Grieves and John Vickers. Digital Twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In *Transdisciplinary Perspectives on Complex Systems*, pages 85–113. Springer, 2017.

[10] Fei Tao and Meng Zhang. Digital Twin shop-floor: A new shop-floor paradigm towards smart manufacturing. *IEEE Access*, 6:20418–20427, 2018.

[11] Armen Der Kiureghian and Ole Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009.

[12] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 1184–1193, 2018.

[13] G. Budai-Balke, R. Dekker, and R. P. Nicolai. A review of planning models for maintenance and production. Technical Report EI 2006-44, Econometric Institute, Erasmus University Rotterdam, 2006.

[14] K. A. Kobbacy, D. P. Murthy, G. Budai, R. Dekker, and R. P. Nicolai. Maintenance and production: A review of planning models. In *Complex System Maintenance Handbook*, pages 321–344. Springer, 2008.

[15] El Houssaine Aghezzaf, Mohamed A. Jamali, and Daoud Ait-Kadi. An integrated production and preventive maintenance planning model. *European Journal of Operational Research*, 181(2):679–685, 2007.

[16] Dirk Hartmann and Herman Van der Auweraar. The executable Digital Twin: Merging the digital and the physics worlds, 2022.

[17] N. Eleftheroglou, G. Galanopoulos, and T. Loutas. Similarity learning hidden semi-markov model for adaptive prognostics of composite structures. *Reliability Engineering & System Safety*, 243:109808, 2024.

[18] M. Liu, J. Qin, D.-G. Lu, W.-H. Zhang, J.-S. Zhu, and M. H. Faber. Towards resilience of offshore wind farms: A framework and application to asset integrity management. *Applied Energy*, 322:119429, 2022.

[19] C. Tsoumpris and G. Theotokatos. A decision-making approach for the health-aware energy management of ship hybrid power plants. *Reliability Engineering & System Safety*, 235:109263, 2023.

[20] Marc Hermans and Péter Tamás. Overall equipment efficiency, total productive maintenance and Digital Twin Technologies – a literature review. *Academic Journal of Manufacturing Engineering*, 22(2):129–137, 2024.

[21] J. Pereira, F. J. G. Silva, J. A. Bastos, L. P. Ferreira, and J. C. O. Matias. Application of the A3 methodology for the improvement of an assembly line. *Procedia Manufacturing*, 38, 2019.

[22] S. P. Pires, O. Sénéchal, E. F. R. Loures, and J. F. Jimenez. An approach to the prioritization of sustainable maintenance drivers in the TBL framework. *IFAC-PapersOnLine*, 49, 2016.

[23] Borut Buchmeister, Darko Friscic, and Iztok Palcic. Bullwhip effect study in a constrained supply chain. *Procedia Engineering*, 69, 2014.

[24] A. P. Kuznetsov, H.-J. Koriath, A. V. Kalyashina, and T. Langer. Equivalence assessment method for the resource efficiency of equipment, technologies and production systems. *Procedia Manufacturing*, 21, 2018.

[25] Itxaro Errandonea, Sergio Beltrán, and Saioa Arrizabalaga. Digital Twin for maintenance: A literature review. *Computers in Industry*, 123:103316, 2020.

[26] Erwin Rauch, Patrick Dallasega, and Dominik T. Matt. Maintenance transformation through Industry 4.0 technologies: A literature review and future research directions. *Computers in Industry*, 120:103251, 2020.

[27] I. T. Christou, N. Kefalakis, A. Zalonis, J. Soldatos, and R. Bröchler. End-to-end industrial IoT platform for actionable predictive maintenance. *IFAC-PapersOnLine*, 53:173–178, 2020.

[28] Vijay K. Vaishnavi and William Kuechler Jr. *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. CRC Press, Boca Raton, FL, 2 edition, 2015.

[29] B. K. Choi and D. Kang. *Modeling and Simulation of Discrete Event Systems*. John Wiley & Sons, Nashville, TN, 2013.

[30] Siemens. Plant simulation help file, 2024.

[31] M. Frantzén, S. Bandaru, and A. H. C. Ng. Digital Twin-based decision support of dynamic maintenance task prioritization using simulation-based optimization and genetic programming. *Decision Analytics Journal*, 3:100039, 2022.

[32] Murray R. Spiegel. *Calculus of Finite Differences and Difference Equations*. McGraw-Hill, New York, 1994.

[33] Saber N. Elaydi. *An Introduction to Difference Equations*. Springer-Verlag, New York, 1996.

[34] Seymour Lipschutz and Marc Lars Lipson. *Discrete Mathematics*. McGraw-Hill, New York, 3 edition, 2007.

[35] Aleksandr Y. Khinchin. *Mathematical Methods in the Theory of Queuing*. Dover Publications, New York, 2013.

[36] John F. Shortle, James M. Thompson, Donald Gross, and Carl M. Harris. *Fundamentals of Queueing Theory*. Wiley, Hoboken, 2018.

[37] H. Zupan and N. Herakovic. Production line balancing with discrete event simulation: A case study. *IFAC-PapersOnLine*, 48:2305–2311, 2015.

[38] P. Ruane, P. Walsh, and J. Cosgrove. Development of a digital model and metamodel to improve the performance of an automated manufacturing line. *Journal of Manufacturing Systems*, 65:538–549, 2022.

[39] L. M. Tumbajoy, M. Muñoz-Añasco, and S. Thiede. Enabling Industry 4.0 impact assessment with manufacturing system simulation: An OEE-based methodology. *Procedia CIRP*, 107:681–686, 2022.

[40] Rita Gamberini, Luca Galloni, Francesco Lolli, and Bianca Rimini. On the analysis of effectiveness in a manufacturing cell: A critical implementation of existing approaches. *Procedia Manufacturing*, 11, 2017.

[41] Lucia Pascale, Marin Mainea, Paul Ciprian Patic, and Luminita Duta. Mathematical decision model to improve TPM indicators. *IFAC Proceedings Volumes*, 45, 2012.

[42] Erwin Kreyszig. *Advanced Engineering Mathematics*. John Wiley & Sons, 9 edition, 2006.

[43] Anand S. Relkar and K. N. Nandurkar. Optimizing and analysing overall equipment effectiveness (OEE) through design of experiments (DOE). *Procedia Engineering*, 38, 2012.

[44] Murray R. Spiegel, John J. Schiller, and R. Alu Srinivasan. *Probability and Statistics*. McGraw-Hill, New York, 4 edition, 2013.

[45] N. Bastos, E. R. Loures, E. A. P. Santos, and M. A. B. de Paula. Production process efficiency analysis: An approach based on colored Petri Nets. *IFAC Proceedings Volumes*, 41(2):1863–1868, 2008.

[46] A. Tayal, N. S. Kalsi, M. K. Gupta, D. Y. Pimenov, M. Sarikaya, and C. I. Pruncu. Effectiveness improvement in manufacturing industry: Trilogy study and open innovation dynamics. *Journal of Open Innovation: Technology, Market, and Complexity*, 7(1):7, 2021.

[47] Taufik Djatna and Imam Muharram Alitu. An application of association rule mining in Total Productive Maintenance strategy: An analysis and modelling in wooden door manufacturing industry. *Procedia Manufacturing*, 4, 2015.

[48] Siri Adolph, Patrick Kübler, Joachim Metternich, and Eberhard Abele. Overall commissioning effectiveness: Systematic identification of value-added shares in material supply. *Procedia CIRP*, 41, 2016.

[49] A. Abu-Samah, M. K. Shahzad, E. Zamai, and A. B. Said. Failure prediction methodology for improved proactive maintenance using Bayesian approach. *IFAC-PapersOnLine*, 48(3):844–851, 2015.

[50] Richard Hedman, Mukund Subramaniyan, and Peter Almström. Analysis of critical factors for automatic measurement of OEE. *Procedia CIRP*, 57, 2016.

[51] M. Mâinea, L. Dută, P. C. Patic, and I. Căciulă. A method to optimize the overall equipment effectiveness. *IFAC Proceedings Volumes*, 43:237–241, 2010.

[52] X. Wang, K. S. Wang, and Y. Liu. Multivariate SPC methods for controlling manufacturing process parameters: A literature review and comparative study. *Computers in Industry*, 117:103212, 2020.

[53] S. Arangio and F. Bontempi. Structural health monitoring of a cable–stayed bridge with Bayesian neural networks. *Structure and Infrastructure Engineering*, 11(4):575–587, 2015.

[54] D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[55] X. Zhang and S. Mahadevan. Bayesian neural networks for flight trajectory prediction and safety assessment. *Decision Support Systems*, 131:113246, 2020.

[56] A. G. Wilson and P. Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. CoRR, 2020. abs/2002.08791.

[57] N. G. Polson and V. Sokolov. Deep learning: a Bayesian perspective. *Bayesian Analysis*, 12(4):1275–1304, 2017.

[58] A. Der Kiureghian and O. Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009.

[59] N. Öztürk and B. Ayvaz. A propagation breakdown management model for the industrial environment: Integration of maintenance and production. *Computers in Industry*, 121:103253, 2020.

[60] D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *5th International Conference on Learning Representations (ICLR 2017), Conference Track Proceedings*, 2017.

[61] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural network. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *ICML'17*, pages 1321–1330, 2017.

[62] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems 32*, pages 13991–14002. Curran Associates, Inc., 2019.

[63] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. Wilson. What are Bayesian neural network posteriors really like? CoRR, 2021. abs/2104.14421.

[64] S. C.-H. Yang, W. K. Vong, R. B. Sojitra, T. Folke, and P. Shafto. Mitigating belief projection in explainable artificial intelligence via Bayesian teaching. *Scientific Reports*, 11(1):9863, May 2021.

[65] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[66] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning*, ICML'16, pages 1050–1059, 2016.

[67] J. Mitros and B. M. Namee. On the validity of Bayesian neural networks for uncertainty estimation. In *AICS*, 2019.

[68] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems 30*, pages 6402–6413. Curran Associates, Inc., 2017.

[69] A. Chan, A. Alaa, Z. Qian, and M. Van Der Schaar. Unlabelled data improves Bayesian uncertainty calibration under covariate shift. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1392–1402, Virtual, July 2020.

[70] M. Opper and O. Winther. A Bayesian approach to on-line learning. In *On-line learning in neural networks*, pages 363–378. 1998.

[71] H. Ritter, A. Botev, and D. Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 3742–3752, 2018.

[72] H. M. D. Kabir, A. Khosravi, M. A. Hosen, and S. Nahavandi. Neural network-based uncertainty quantification: A survey of methodologies and applications. *IEEE Access*, 6:36218–36234, 2018.

[73] K. Krishnamoorthy. *Handbook of Statistical Distributions with Applications*. Chapman & Hall/CRC, Philadelphia, PA, 2 edition, 2020.

[74] A. K. S. Jardine and A. H. C. Tsang. The role of emerging technologies in physical asset management. In *Maintenance, Replacement, and Reliability*, pages 227–262. CRC Press, Boca Raton, 2021.

[75] R. Keith Mobley. *Maintenance Fundamentals*. Butterworth-Heinemann, Woburn, MA, 2 edition, 2014.

[76] H. Chen, L. Li, and Y. Sun. Risk assessment of aero engine failure based on monte carlo simulation. *Procedia Engineering*, 80:415–423, 2014.

[77] C. Forbes, M. Evans, N. Hastings, and B. Peacock. *Statistical Distributions*. Wiley-Blackwell, Hoboken, NJ, 4 edition, 2010.

[78] H. Rinne. *The Weibull Distribution*. Chapman & Hall/CRC, Philadelphia, PA, 2020.

[79] S. Woo. Modern definitions in reliability engineering. In *Reliability Design of Mechanical Systems*, pages 53–99. Springer, Singapore, 2020.

[80] H. Abdo, J. M. Flaus, and H. Bouguila. Explicit and implicit Bayesian Network-based methods for the maintenance decision-making process. *Computers in Industry*, 117:103215, 2020.

[81] Y. H. Choi, G. Y. Na, and J. Yang. Fuzzy-inference-based decision-making method for the systematization of statistical process capability control. *Computers in Industry*, 123:103296, 2020.

[82] Fatih Camci and S. Guclu. Machine learning and reasoning for predictive maintenance in industry. *Computers in Industry*, 117:103214, 2020.

[83] S. Jaskó, A. Skrop, T. Holczinger, T. Chován, and J. Abonyi. Development of manufacturing execution systems in accordance with Industry 4.0 requirements. *Computers in Industry*, 123:103300, 2020.

[84] A. Landström, P. Almström, M. Winroth, C. Andersson, A. E. Öberg, M. Kurdve, S. Shahbazi, M. Wiktorsson, C. Windmark, and M. Zackrisson. A life cycle approach to business performance measurement systems. *Procedia Manufacturing*, 25:126–133, 2018.

[85] Marc Hermans and Péter Tamás. OEE as a tool for stability and continuity. In Péter Tamás et al., editors, *CECOL 2024*, pages 15–40. Springer Nature Switzerland, 2024.

[86] Marc Hermans and Sándor Fegyverneki. Optimizing maintenance strategies through simulation modeling: A plant simulation approach. *International Journal of Engineering and Applied Sciences*, 11(7):17–26, July 2024.

[87] Marc Hermans and Péter Tamás. Applying Bayesian neural networks to optimize maintenance logistics. *Acta Logistica*, 13(1):162–173, March 2026.

# Appendices

# A Normalisation Process

# DB Normalizer

## Imports + paths

```
import re
import sqlite3
from pathlib import Path

# --- If DB files are in the same folder as this notebook, this is enough ---
SOURCE_DB = Path("myExperiments2.db")          # <-- change to your real filename
TARGET_DB = Path("normalized_Exp_DB.db")      # output filename

print("Working directory:", Path().resolve())
print("SOURCE exists:", SOURCE_DB.exists(), "|", SOURCE_DB)
print("TARGET will be:", TARGET_DB)
```

```
Working directory: C:\Learning\University\Thesis\Model
SOURCE exists: True | myExperiments2.db
TARGET will be: normalized_Exp_DB.db
```

## Helpers: quoting, table listing, PRAGMA

```
def quote_ident(name: str) -> str:
    """SQLite identifier quoting for names that may contain spaces/special chars."""
    return '"' + name.replace('"', '""') + '"'

def list_tables(conn) -> list[str]:
    return [r[0] for r in conn.execute(
        "SELECT name FROM sqlite_master WHERE type='table' AND name NOT LIKE 'sqlite_%';"
    ).fetchall()]

def pragma_table_info(conn, table: str):
    return conn.execute(f"PRAGMA table_info({quote_ident(table)})").fetchall()

def table_columns(conn, table: str) -> list[str]:
    return [r[1] for r in pragma_table_info(conn, table)]
```

## Parsing of repeating run tables

```
RUN_TABLE_RE = re.compile(
    r'^(?P<prefix>TS_In|TS_Out|MNotes|FNotes)_(?P<expnr>.+)_R(?P<run>\d+)$'
)

def parse_run_table(name: str):
    m = RUN_TABLE_RE.match(name)
    if not m:
        return None
    d = m.groupdict()
    d["run"] = int(d["run"])
    return d
```

# Connect to DBs

```
# Open source
src = sqlite3.connect(SOURCE_DB)
src.row_factory = sqlite3.Row

# Recreate target cleanly
if TARGET_DB.exists():
    TARGET_DB.unlink()

tgt = sqlite3.connect(TARGET_DB)
tgt.row_factory = sqlite3.Row
tgt.execute("PRAGMA foreign_keys = ON;")

print("Connected. Target recreated.")
```

```
Connected. Target recreated.
```

# Create base normalized schema

```
tgt.executescript("""
CREATE TABLE IF NOT EXISTS Experiment (
    ExperimentID INTEGER PRIMARY KEY AUTOINCREMENT,
    ExpNr        TEXT NOT NULL UNIQUE,

    Availability TEXT,
    "Distribution - Duration" TEXT,
    MTTR         REAL,
    Sigma        REAL,
    "Distribution - Interval" TEXT,
    Mu           REAL,
    Beta         REAL,
    Eta          REAL,
    "Interval Time" REAL,
    "Repair Time" REAL,
    "Brooker active" TEXT,
    "Nominal CT" REAL,
    "Customer CT" REAL,
    "Max CT" REAL,
    Rho          REAL
);

CREATE TABLE IF NOT EXISTS Run (
    RunID        INTEGER PRIMARY KEY AUTOINCREMENT,
    ExperimentID INTEGER NOT NULL,
    RunNo        INTEGER NOT NULL,
    UNIQUE(ExperimentID, RunNo),
    FOREIGN KEY (ExperimentID) REFERENCES Experiment(ExperimentID)
);
""")
tgt.commit()
print("Base schema created.")
```

```
Base schema created.
```

# Import Exp_Overview into Experiment

```
tables = list_tables(src)
if "Exp_Overview" not in tables:
    raise RuntimeError("Exp_Overview not found in source DB.")

rows = src.execute("SELECT * FROM Exp_Overview").fetchall()

def get_row_value(r, colname):
    return r[colname] if colname in r.keys() else None

insert_sql = """
INSERT OR IGNORE INTO Experiment(
    ExpNr, Availability, "Distribution - Duration", MTTR, Sigma,
    "Distribution - Interval", Mu, Beta, Eta, "Interval Time",
    "Repair Time", "Brooker active", "Nominal CT", "Customer CT", "Max CT", Rho
) VALUES (
    ?, ?, ?, ?, ?,
    ?, ?, ?, ?, ?,
    ?, ?, ?, ?, ?, ?
);
"""

data = []
for r in rows:
    data.append((
        get_row_value(r, "ExpNr"),
        get_row_value(r, "Availability"),
        get_row_value(r, "Distribution - Duration"),
        get_row_value(r, "MTTR"),
        get_row_value(r, "Sigma"),
        get_row_value(r, "Distribution - Interval"),
        get_row_value(r, "Mu"),
        get_row_value(r, "Beta"),
        get_row_value(r, "Eta"),
        get_row_value(r, "Interval Time"),
        get_row_value(r, "Repair Time"),
        get_row_value(r, "Brooker active"),
        get_row_value(r, "Nominal CT"),
        get_row_value(r, "Customer CT"),
        get_row_value(r, "Max CT"),
        get_row_value(r, "Rho"),
    ))

tgt.executemany(insert_sql, data)
tgt.commit()

n_exp = tgt.execute("SELECT COUNT(*) AS n FROM Experiment").fetchone()["n"]
print("Experiments inserted:", n_exp)
```

```
Experiments inserted: 52
```

# Key helpers: get/create ExperimentID and RunID

```
def get_experiment_id(conn, expnr: str) -> int:
    row = conn.execute("SELECT ExperimentID FROM Experiment WHERE ExpNr=?",
```

```
(expnr,)).fetchone()
    if row:
        return row["ExperimentID"]
    # If a run exists for an experiment that was not in Exp_Overview, create minimal
entry.
    conn.execute("INSERT INTO Experiment(ExpNr) VALUES (?)", (expnr,))
    return conn.execute("SELECT last_insert_rowid() AS id").fetchone()["id"]

def get_run_id(conn, experiment_id: int, runno: int) -> int:
    row = conn.execute(
        "SELECT RunID FROM Run WHERE ExperimentID=? AND RunNo=?",
        (experiment_id, runno)
    ).fetchone()
    if row:
        return row["RunID"]
    conn.execute("INSERT INTO Run(ExperimentID, RunNo) VALUES (?,?)", (experiment_id,
runno))
    return conn.execute("SELECT last_insert_rowid() AS id").fetchone()["id"]
```

# Consolidation utilities

```
def ensure_consolidated_table(tgt_conn, consolidated_name: str, src_table: str):
    """
    Create consolidated table (if needed) with:
    RunID + all columns of src_table (same names/types).
    """
    cols = pragma_table_info(src, src_table)
    col_defs = []
    for c in cols:
        col_name = c[1]
        col_type = c[2] if c[2] else "TEXT"
        col_defs.append(f"{quote_ident(col_name)} {col_type}")

    ddl = f"""
    CREATE TABLE IF NOT EXISTS {quote_ident(consolidated_name)} (
        RunID INTEGER NOT NULL,
        {", ".join(col_defs)},
        FOREIGN KEY (RunID) REFERENCES Run(RunID)
    );
    """
    tgt_conn.executescript(ddl)

def import_run_table(src_conn, tgt_conn, src_table: str, consolidated_name: str, run_id:
int) -> int:
    cols = table_columns(src_conn, src_table)

    sel_cols = ", ".join(quote_ident(c) for c in cols)
    ins_cols = ", ".join([quote_ident("RunID")] + [quote_ident(c) for c in cols])
    placeholders = ", ".join(["?"] * (1 + len(cols)))

    rows = src_conn.execute(f"SELECT {sel_cols} FROM
{quote_ident(src_table)}").fetchall()
    data = [(run_id, *[r[c] for c in cols]) for r in rows]

    tgt_conn.executemany(
        f"INSERT INTO {quote_ident(consolidated_name)} ({ins_cols}) VALUES
({placeholders})",
        data
```

```
    )
    return len(rows)
```

# Normalize all repeating run tables

```
tables = list_tables(src)

run_tables = []
for t in tables:
    p = parse_run_table(t)
    if p:
        p["table"] = t
        run_tables.append(p)

print("Run tables discovered:", len(run_tables))
print("Example:", run_tables[0] if run_tables else None)

counts = {"TS_In": 0, "TS_Out": 0, "MNotes": 0, "FNotes": 0}

for item in run_tables:
    prefix = item["prefix"]
    expnr  = item["expnr"]
    runno  = item["run"]
    src_table = item["table"]

    exp_id = get_experiment_id(tgt, expnr)
    run_id = get_run_id(tgt, exp_id, runno)

    consolidated = prefix  # consolidated table names are TS_In, TS_Out, MNotes, FNotes
    ensure_consolidated_table(tgt, consolidated, src_table)
    n = import_run_table(src, tgt, src_table, consolidated, run_id)
    counts[prefix] += n

tgt.commit()

print("Imported row counts:", counts)
print("Runs created:", tgt.execute("SELECT COUNT(*) AS n FROM Run").fetchone()["n"])
```

```
Run tables discovered: 1040
Example: {'prefix': 'TS_In', 'expnr': 'Exp_A_79_DC', 'run': 1, 'table':
'TS_In_Exp_A_79_DC_R1'}
Imported row counts: {'TS_In': 5357046, 'TS_Out': 5306720, 'MNotes': 44285, 'FNotes':
12457}
Runs created: 260
```

# Copy experiment-level tables

```
def copy_table(src_conn, tgt_conn, table_name: str) -> bool:
    row = src_conn.execute(
        "SELECT sql FROM sqlite_master WHERE type='table' AND name=?",
        (table_name,)
    ).fetchone()
    if not row or not row[0]:
        print(f"Skipping (no CREATE sql): {table_name}")
        return False
```

```
    # Create in target
    tgt_conn.execute(row[0])

    cols = table_columns(src_conn, table_name)
    col_list = ", ".join(quote_ident(c) for c in cols)
    placeholders = ", ".join(["?"] * len(cols))

    rows = src_conn.execute(f"SELECT {col_list} FROM
{quote_ident(table_name)}").fetchall()
    tgt_conn.executemany(
        f"INSERT INTO {quote_ident(table_name)} ({col_list}) VALUES ({placeholders})",
        [[r[c] for c in cols] for r in rows]
    )
    tgt_conn.commit()
    return True

for tname in ["Exp_Results", "Exp_DetailedResults", "Exp_pValues"]:
    exists = tname in tables
    print("Exists in source:", tname, "->", exists)
    if exists:
        ok = copy_table(src, tgt, tname)
        print("Copied:", tname, "->", ok)
```

```
Exists in source: Exp_Results -> True
Copied: Exp_Results -> True
Exists in source: Exp_DetailedResults -> True
Copied: Exp_DetailedResults -> True
Exists in source: Exp_pValues -> True
Copied: Exp_pValues -> True
```

# Add ExperimentID to copied Exp_* tables

```
def add_and_fill_experiment_id(conn, table_name: str, expnr_col: str):
    cols = table_columns(conn, table_name)
    if "ExperimentID" not in cols:
        conn.execute(f"ALTER TABLE {quote_ident(table_name)} ADD COLUMN ExperimentID
INTEGER;")

    conn.execute(f"""
        UPDATE {quote_ident(table_name)}
        SET ExperimentID = (
            SELECT e.ExperimentID
            FROM Experiment e
            WHERE e.ExpNr = {quote_ident(table_name)}.{quote_ident(expnr_col)}
        )
        WHERE ExperimentID IS NULL;
    """)
    conn.commit()

for tname in ["Exp_Resuts", "Exp_DetailedResuts", "Exp_pValues"]:
    if tname not in list_tables(tgt):
        continue

    cols = table_columns(tgt, tname)
    print("\n", tname, "columns:", cols)

    # Heuristic: most likely column is ExpNr. If not, you will see it in the printed
list.
```

```
    if "ExpNr" in cols:
        add_and_fill_experiment_id(tgt, tname, expnr_col="ExpNr")
        filled = tgt.execute(
            f"SELECT COUNT(*) AS n FROM {quote_ident(tname)} WHERE ExperimentID IS NOT
NULL"
        ).fetchone()["n"]
        total = tgt.execute(f"SELECT COUNT(*) AS n FROM {quote_ident(tname)}").fetchone()
["n"]
        print(f"{tname}: ExperimentID filled for {filled}/{total} rows (via ExpNr).")
    else:
        print(f"{tname}: No 'ExpNr' column found. If another column stores experiment
name, tell me its name.")
```

```
 Exp_pValues columns: ['', 'Exp_A_84_DC', 'Exp_A_89_DC', 'Exp_A_94_DC', 'Exp_A_99_DC',
'Exp_A_79_CC', 'Exp_A_84_CC', 'Exp_A_89_CC', 'Exp_A_94_CC', 'Exp_A_99_CC', 'Exp_B_79_DC',
'Exp_B_84_DC', 'Exp_B_89_DC', 'Exp_B_94_DC', 'Exp_B_99_DC', 'Exp_B_79_CC', 'Exp_B_84_CC',
'Exp_B_89_CC', 'Exp_B_94_CC', 'Exp_B_99_CC', 'Exp_C1_79_DC', 'Exp_C1_84_DC',
'Exp_C1_89_DC', 'Exp_C1_94_DC', 'Exp_C1_99_DC', 'Exp_C1_79_CC', 'Exp_C1_84_CC',
'Exp_C1_89_CC', 'Exp_C1_94_CC', 'Exp_C1_99_CC', 'Exp_C2_79_DC', 'Exp_C2_84_DC',
'Exp_C2_89_DC', 'Exp_C2_94_DC', 'Exp_C2_99_DC', 'Exp_C2_79_CC', 'Exp_C2_84_CC',
'Exp_C2_89_CC', 'Exp_C2_94_CC', 'Exp_C2_99_CC', 'Exp_D_xs_DC', 'Exp_D_vs_DC',
'Exp_D_s_DC', 'Exp_D_m_DC', 'Exp_D_vm_DC', 'Exp_D_xm_DC', 'Exp_D_xs_CC', 'Exp_D_vs_CC',
'Exp_D_s_CC', 'Exp_D_m_CC', 'Exp_D_vm_CC', 'Exp_D_xm_CC']
Exp_pValues: No 'ExpNr' column found. If another column stores experiment name, tell me
its name.
```

# Indexes

```
tgt.executescript("""
CREATE INDEX IF NOT EXISTS idx_experiment_expnr ON Experiment(ExpNr);
CREATE INDEX IF NOT EXISTS idx_run_exp_runno    ON Run(ExperimentID, RunNo);

CREATE INDEX IF NOT EXISTS idx_tsin_run   ON TS_In(RunID);
CREATE INDEX IF NOT EXISTS idx_tsout_run  ON TS_Out(RunID);
CREATE INDEX IF NOT EXISTS idx_mnotes_run ON MNotes(RunID);
CREATE INDEX IF NOT EXISTS idx_fnotes_run ON FNotes(RunID);
""")
tgt.commit()
print("Indexes created.")
```

```
Indexes created.
```

# Validation checks

```
print("Experiments:", tgt.execute("SELECT COUNT(*) AS n FROM Experiment").fetchone()
["n"])
print("Runs:",        tgt.execute("SELECT COUNT(*) AS n FROM Run").fetchone()["n"])

for t in ["TS_In", "TS_Out", "MNotes", "FNotes"]:
    if t in list_tables(tgt):
        n = tgt.execute(f"SELECT COUNT(*) AS n FROM {quote_ident(t)}").fetchone()["n"]
        print(f"{t} rows:", n)
    else:
        print(f"{t} not present in target.")

for t in ["Exp_Resuts", "Exp_DetailedResuts", "Exp_pValues"]:
    if t in list_tables(tgt):
```

```
        n = tgt.execute(f"SELECT COUNT(*) AS n FROM {quote_ident(t)}").fetchone()["n"]
        print(f"{t} rows:", n)

# Show a few runs with experiment name
rows = tgt.execute("""
SELECT r.RunID, e.ExpNr, r.RunNo
FROM Run r
JOIN Experiment e ON e.ExperimentID = r.ExperimentID
ORDER BY e.ExpNr, r.RunNo
LIMIT 10;
""").fetchall()

print("\nFirst runs:")
for r in rows:
    print(dict(r))
```

```
Experiments: 52
Runs: 260
TS_In rows: 5357046
TS_Out rows: 5306720
MNotes rows: 44285
FNotes rows: 12457
Exp_pValues rows: 51

First runs:
{'RunID': 26, 'ExpNr': 'Exp_A_79_CC', 'RunNo': 1}
{'RunID': 27, 'ExpNr': 'Exp_A_79_CC', 'RunNo': 2}
{'RunID': 28, 'ExpNr': 'Exp_A_79_CC', 'RunNo': 3}
{'RunID': 29, 'ExpNr': 'Exp_A_79_CC', 'RunNo': 4}
{'RunID': 30, 'ExpNr': 'Exp_A_79_CC', 'RunNo': 5}
{'RunID': 1, 'ExpNr': 'Exp_A_79_DC', 'RunNo': 1}
{'RunID': 2, 'ExpNr': 'Exp_A_79_DC', 'RunNo': 2}
{'RunID': 3, 'ExpNr': 'Exp_A_79_DC', 'RunNo': 3}
{'RunID': 4, 'ExpNr': 'Exp_A_79_DC', 'RunNo': 4}
{'RunID': 5, 'ExpNr': 'Exp_A_79_DC', 'RunNo': 5}
```

# Proper cleanup

```
tgt.commit()
src.close()
tgt.close()
print("Connections closed cleanly.")
```

```
Connections closed cleanly.
```

# B Enrichment Process

# Imports and Connections

```
import re
import sqlite3
from pathlib import Path
import pandas as pd
import numpy as np
import json

# --- If DB files are in the same folder as this notebook, this is enough ---
TARGET_DB = Path("normalized_Exp_DB.db")      # output filename

print("Working directory:", Path().resolve())
print("TARGET will be:", TARGET_DB)
```

```
Working directory: C:\Learning\University\Thesis\Model
TARGET will be: normalized_Exp_DB.db
```

# Utility

```
def calculate_basic_metrics(df: pd.DataFrame) -> dict:
    """
    Compute summary metrics from structured OEE dataframe.

    Expected columns:
        [ Run, TimeStamp, Total, Total_Cumul,
        NiO, NiO_Cumul, DeltaT, DeltaT_Cumul, CT_tot, CT_act]

    Returns:
        dict with keys:
            NettoBetriebsZeit, MTTR, Total_Failtime, NrOfFails, CT_act_stats,
            CT_tot_stats, NiO, Output_tot, Total_time, PBZ, Parts_per_Cycle
    """
    if df.empty:
        return {}

    # ------------------------------------------------------------
    # Helper: get_statistics
    # ------------------------------------------------------------
    def get_statistics(series: pd.Series, parts_per_cycle: float) -> dict:
        """
        Return basic statistics, bounds, and deviation for a numeric column,
        normalized by parts_per_cycle.
        """
        s = pd.to_numeric(series, errors="coerce").dropna()
        if s.empty:
            return {}

        # Normalize per part
        if parts_per_cycle and parts_per_cycle != 0:
            s = s / parts_per_cycle

        q1, q2, q3 = np.percentile(s, [25, 50, 75])
        iqr = q3 - q1
        lb = max(0, q1 - 1.5 * iqr)
```

```
    ub = q3 + 1.5 * iqr

    # Determine mode
    mode_vals = s.mode(dropna=True)

    if mode_vals.empty:
        modus = q2
    else:
        non_zero_modes = mode_vals[mode_vals != 0]
        if not non_zero_modes.empty:
            modus = non_zero_modes.iloc[0]
        else:
            modus = q2

    # Additional safeguard
    if modus < q1:
        modus = q2

    stats = {
        "min": round(float(s.min()), 1),
        "Q1": round(float(q1), 1),
        "Q2": round(float(q2), 1),
        "Q3": round(float(q3), 1),
        "max": round(float(s.max()), 1),
        "avg": round(float(s.mean()), 1),
        "std": round(float(s.std(ddof=1)), 4),   # sample standard deviation
        "IQR": round(float(iqr), 1),
        "LB": round(float(lb), 1),
        "UB": round(float(ub), 1),
        "Modus": round(float(modus), 1)
    }
    return stats


results = {}

# 1. Parts per cycle → mean Output_Total rounded to nearest integer
results["Parts_per_Cycle"] = int(round(df["Total"].mean(), 0))

# 4. PBZ → max(DeltaT_Cumul)
results["PBZ"] = float(df["DeltaT_Cumul"].max())

# 5. Total_time = PBZ + Pauses_tot + Unplanned_tot
results["Total_time"] = results["PBZ"]

# 6. Output_tot = max(Total_Cumul)
results["Output_tot"] = int(df["Total_Cumul"].max())

# 7. NiO = max(NiO_Cumul)
results["NiO"] = int(df["NiO_Cumul"].max())

# 8. Get statistics CT_tot and CT_actuel
results["CT_tot_stats"] = get_statistics(df["CT_tot"], results["Parts_per_Cycle"])
stats = get_statistics(df["CT_act"], results["Parts_per_Cycle"])
results["CT_act_stats"] = stats

# ----------------------------------------------------------
# Failure time detection & replacement
# ----------------------------------------------------------
df = df.copy()
```

```
    df["Fail_time"] = 0.0

    parts_per_cycle = results["Parts_per_Cycle"]

    if stats:
        myMod = stats["Modus"]
        q3 = stats["Q3"]
        thr = q3 * parts_per_cycle  # threshold for failures

        mask_fail = df["CT_act"] > thr
        df.loc[mask_fail, "Fail_time"] = df.loc[mask_fail, "CT_act"] - myMod *
parts_per_cycle
        df.loc[mask_fail, "CT_act"] = myMod * parts_per_cycle  # replace with nominal
cycle

        # ------------------------------------------------------
        # Failure summary
        # ------------------------------------------------------
        fail_series = df["Fail_time"]
        results["NrOfFails"] = int((fail_series > 0).sum())
        results["Total_Failtime"] = float(fail_series.sum())
        results["MTTR"] = (
            results["Total_Failtime"] / results["NrOfFails"]
            if results["NrOfFails"] > 0
            else 0.0
        )
        results["NettoBetriebsZeit"] = results["PBZ"] - results["Total_Failtime"]

    return results
```

```
def OEE_Calc(OEE_Data: dict) -> dict:
    """Compute full OEE dictionary from pre-aggregated OEE_Data."""

    # --- Base values ---
    PBZ = OEE_Data.get("PBZ", 0)
    Total_time = OEE_Data.get("Total_time", 0)
    NettoBetriebsZeit = OEE_Data.get("NettoBetriebsZeit", 0)
    Output_total = OEE_Data.get("Output_tot", 0)
    NiO = OEE_Data.get("NiO", 0)
    Parts_per_Cycle = OEE_Data.get("Parts_per_Cycle", 1)
    CT_stats = OEE_Data.get("CT_act_stats", {})
    Q2 = CT_stats.get("Q2", 0)
    Modus = CT_stats.get("Modus", 0)
    Avg = CT_stats.get("avg", 0)

    # --- 1. Utility ---
    Utility = PBZ / Total_time if Total_time else 0

    # --- 2. Availability ---
    Availability = NettoBetriebsZeit / PBZ if PBZ else 0

    # --- 3. Performance (Modus-based) ---
    NOR = (Output_total * Q2 * Parts_per_Cycle) / NettoBetriebsZeit if NettoBetriebsZeit
else 0
    SpR = Modus /Q2 if Modus else 0
    Performance = NOR * SpR

    # --- 5. Quality ---
    Quality = 1 - (NiO / Output_total) if Output_total else 0
```

```
    # --- 6. OEE results ---
    OEE = Availability * Performance * Quality

    # --- 7. Return dictionary ---
    return {
        # Base factors
        "Utility_%": round(Utility * 100, 1),
        "Availability_%": round(Availability * 100, 1),
        "Quality_%": round(Quality * 100, 1),

        # Performance (Modus-based)
        "Net_Operating_Rate_%": round(NOR*100, 4),
        "Speed_rate_%": round(SpR*100, 4),
        "Performance_%": round(Performance * 100, 1),

        # Overall OEE
        "OEE_%": round(OEE * 100, 1)
    }
```

# Connect and Create OEE Tables

```
tgt = sqlite3.connect(TARGET_DB)
tgt.row_factory = sqlite3.Row

print("Connected. Target Connected.")

tgt.executescript("""
CREATE TABLE IF NOT EXISTS OEE_In (
    Run INTEGER,
    TimeStamp REAL,
    Total INTEGER,
    Total_Cumul INTEGER,
    NiO INTEGER,
    NiO_Cumul INTEGER,
    DeltaT REAL,
    DeltaT_Cumul REAL,
    CT_tot REAL,
    CT_act REAL
);

CREATE TABLE IF NOT EXISTS OEE_Out (
    Run INTEGER,
    TimeStamp REAL,
    Total INTEGER,
    Total_Cumul INTEGER,
    NiO INTEGER,
    NiO_Cumul INTEGER,
    DeltaT REAL,
    DeltaT_Cumul REAL,
    CT_tot REAL,
    CT_act REAL
);
""")

tgt.commit()

print("OEE tables created.")
```

```
Connected. Target Connected.
OEE tables created.
```

# OEE In

## Create Metrics Table

```
tgt.execute("""
CREATE TABLE IF NOT EXISTS OEE_In_WindowMetrics (
    Run INTEGER NOT NULL,
    EndTimeStamp REAL NOT NULL,
    EndRow INTEGER NOT NULL,
    WindowEvents INTEGER NOT NULL,
    StepEvents INTEGER NOT NULL,
    BaseMetricsJSON TEXT NOT NULL,
    OEEMetricsJSON TEXT NOT NULL,
    EndBufferLevel INTEGER NOT NULL,
    PRIMARY KEY (Run, EndRow)
);
""")
tgt.commit()
```

## Load TS_in into Python Dataframe / RunID

```
# Optional but recommended: clear table before rebuilding
tgt.execute("DELETE FROM OEE_In;")
tgt.commit()

# Get all distinct runs
run_ids = [r["RunID"] for r in tgt.execute(
    "SELECT DISTINCT RunID FROM TS_In ORDER BY RunID"
).fetchall()]

print("Runs detected:", run_ids)

for run_id in run_ids:

    # Load only required base columns
    df = pd.read_sql_query(f"""
        SELECT
            RunID AS Run,
            TimeStamp,
            Total,
            NiO
        FROM TS_In
        WHERE RunID = {run_id}
        ORDER BY TimeStamp
    """, tgt)

        # Copy cumulative values
    df["Total_Cumul"] = df["Total"]
    df["NiO_Cumul"]   = df["NiO"]

    # Unwind cumulative into increments
    df["Total"] = df["Total_Cumul"].diff().fillna(df["Total_Cumul"])
```

```
    df["NiO"]   = df["NiO_Cumul"].diff().fillna(df["NiO_Cumul"])

    # Safety: eliminate possible negative artifacts
    df["Total"] = df["Total"].clip(lower=0)
    df["NiO"]   = df["NiO"].clip(lower=0)

    # --- DeltaT ---
    df["DeltaT"] = df["TimeStamp"].diff()

    # Median of DeltaT excluding first NaN and excluding non-positive deltas
    valid_dt = df["DeltaT"].iloc[1:]
    valid_dt = valid_dt[valid_dt > 0]

    if len(valid_dt) == 0:
        # Fallback if run has only 1 row or timestamps are not increasing
        dt0 = 0.0
    else:
        dt0 = float(valid_dt.median())

    df.loc[0, "DeltaT"] = dt0

    # (Optional but consistent) cumulative time
    df["DeltaT_Cumul"] = df["DeltaT"].cumsum()

    # --- Cycle Times ---
    df["CT_act"] = df["DeltaT"]

    # Avoid division by zero
    df["CT_tot"] = np.where(
        df["Total_Cumul"] > 0,
        df["DeltaT_Cumul"] / df["Total_Cumul"],
        0.0
    )

    # Append to OEE_In
    df.to_sql("OEE_In", tgt, if_exists="append", index=False)

    print(f"Run {run_id} appended. Rows: {len(df)}")

tgt.commit()

print("Step 2 completed.")
```

```
Runs detected: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,
65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106,
107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124,
125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160,
161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178,
179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196,
197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214,
215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232,
233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250,
251, 252, 253, 254, 255, 256, 257, 258, 259, 260]
Run 1 appended. Rows: 20233
Run 2 appended. Rows: 20290
Run 3 appended. Rows: 20079
Run 4 appended. Rows: 20044
Run 5 appended. Rows: 19455
Run 6 appended. Rows: 20295
Run 7 appended. Rows: 20829
Run 8 appended. Rows: 20096
Run 9 appended. Rows: 20048
Run 10 appended. Rows: 19505
Run 11 appended. Rows: 20296
Run 12 appended. Rows: 20907
Run 13 appended. Rows: 20102
Run 14 appended. Rows: 20048
Run 15 appended. Rows: 19513
Run 16 appended. Rows: 20296
Run 17 appended. Rows: 20923
Run 18 appended. Rows: 20106
Run 19 appended. Rows: 20050
Run 20 appended. Rows: 19522
Run 21 appended. Rows: 20296
Run 22 appended. Rows: 20923
Run 23 appended. Rows: 20106
Run 24 appended. Rows: 20053
Run 25 appended. Rows: 19524
Run 26 appended. Rows: 20770
Run 27 appended. Rows: 20505
Run 28 appended. Rows: 20655
Run 29 appended. Rows: 20600
Run 30 appended. Rows: 20216
Run 31 appended. Rows: 21172
Run 32 appended. Rows: 21340
Run 33 appended. Rows: 20976
Run 34 appended. Rows: 20907
Run 35 appended. Rows: 20354
Run 36 appended. Rows: 21210
Run 37 appended. Rows: 21799
Run 38 appended. Rows: 20991
Run 39 appended. Rows: 20909
Run 40 appended. Rows: 20376
Run 41 appended. Rows: 21210
Run 42 appended. Rows: 21821
Run 43 appended. Rows: 20996
Run 44 appended. Rows: 20910
Run 45 appended. Rows: 20384
Run 46 appended. Rows: 21210
```

```
Run 47 appended. Rows: 21833
Run 48 appended. Rows: 20998
Run 49 appended. Rows: 20911
Run 50 appended. Rows: 20389
Run 51 appended. Rows: 20540
Run 52 appended. Rows: 20001
Run 53 appended. Rows: 20655
Run 54 appended. Rows: 20160
Run 55 appended. Rows: 20367
Run 56 appended. Rows: 20604
Run 57 appended. Rows: 20071
Run 58 appended. Rows: 20710
Run 59 appended. Rows: 20167
Run 60 appended. Rows: 20451
Run 61 appended. Rows: 20604
Run 62 appended. Rows: 20071
Run 63 appended. Rows: 20710
Run 64 appended. Rows: 20169
Run 65 appended. Rows: 20451
Run 66 appended. Rows: 20604
Run 67 appended. Rows: 20071
Run 68 appended. Rows: 20710
Run 69 appended. Rows: 20169
Run 70 appended. Rows: 20451
Run 71 appended. Rows: 20604
Run 72 appended. Rows: 20071
Run 73 appended. Rows: 20710
Run 74 appended. Rows: 20169
Run 75 appended. Rows: 20451
Run 76 appended. Rows: 20862
Run 77 appended. Rows: 20551
Run 78 appended. Rows: 20655
Run 79 appended. Rows: 20600
Run 80 appended. Rows: 20607
Run 81 appended. Rows: 21680
Run 82 appended. Rows: 21785
Run 83 appended. Rows: 21971
Run 84 appended. Rows: 21929
Run 85 appended. Rows: 21871
Run 86 appended. Rows: 21733
Run 87 appended. Rows: 21858
Run 88 appended. Rows: 22288
Run 89 appended. Rows: 22062
Run 90 appended. Rows: 21925
Run 91 appended. Rows: 21733
Run 92 appended. Rows: 21858
Run 93 appended. Rows: 22288
Run 94 appended. Rows: 22070
Run 95 appended. Rows: 21928
Run 96 appended. Rows: 21733
Run 97 appended. Rows: 21858
Run 98 appended. Rows: 22288
Run 99 appended. Rows: 22072
Run 100 appended. Rows: 21928
Run 101 appended. Rows: 18306
Run 102 appended. Rows: 18380
Run 103 appended. Rows: 18456
Run 104 appended. Rows: 18549
Run 105 appended. Rows: 18304
Run 106 appended. Rows: 18306
```

```
Run 107 appended. Rows: 18380
Run 108 appended. Rows: 18456
Run 109 appended. Rows: 18549
Run 110 appended. Rows: 18304
Run 111 appended. Rows: 18306
Run 112 appended. Rows: 18380
Run 113 appended. Rows: 18456
Run 114 appended. Rows: 18549
Run 115 appended. Rows: 18304
Run 116 appended. Rows: 18306
Run 117 appended. Rows: 18380
Run 118 appended. Rows: 18456
Run 119 appended. Rows: 18549
Run 120 appended. Rows: 18304
Run 121 appended. Rows: 18306
Run 122 appended. Rows: 18380
Run 123 appended. Rows: 18456
Run 124 appended. Rows: 18549
Run 125 appended. Rows: 18304
Run 126 appended. Rows: 19359
Run 127 appended. Rows: 19306
Run 128 appended. Rows: 19430
Run 129 appended. Rows: 19372
Run 130 appended. Rows: 19276
Run 131 appended. Rows: 19359
Run 132 appended. Rows: 19306
Run 133 appended. Rows: 19430
Run 134 appended. Rows: 19379
Run 135 appended. Rows: 19277
Run 136 appended. Rows: 19359
Run 137 appended. Rows: 19306
Run 138 appended. Rows: 19430
Run 139 appended. Rows: 19379
Run 140 appended. Rows: 19277
Run 141 appended. Rows: 19359
Run 142 appended. Rows: 19306
Run 143 appended. Rows: 19430
Run 144 appended. Rows: 19379
Run 145 appended. Rows: 19277
Run 146 appended. Rows: 19359
Run 147 appended. Rows: 19306
Run 148 appended. Rows: 19430
Run 149 appended. Rows: 19379
Run 150 appended. Rows: 19277
Run 151 appended. Rows: 20664
Run 152 appended. Rows: 20551
Run 153 appended. Rows: 20655
Run 154 appended. Rows: 20600
Run 155 appended. Rows: 20607
Run 156 appended. Rows: 20721
Run 157 appended. Rows: 21250
Run 158 appended. Rows: 20963
Run 159 appended. Rows: 20808
Run 160 appended. Rows: 20810
Run 161 appended. Rows: 20721
Run 162 appended. Rows: 21299
Run 163 appended. Rows: 20963
Run 164 appended. Rows: 20811
Run 165 appended. Rows: 20817
Run 166 appended. Rows: 20721
```

```
Run 167 appended. Rows: 21299
Run 168 appended. Rows: 20963
Run 169 appended. Rows: 20811
Run 170 appended. Rows: 20817
Run 171 appended. Rows: 20721
Run 172 appended. Rows: 21299
Run 173 appended. Rows: 20963
Run 174 appended. Rows: 20811
Run 175 appended. Rows: 20817
Run 176 appended. Rows: 20862
Run 177 appended. Rows: 20551
Run 178 appended. Rows: 20655
Run 179 appended. Rows: 20600
Run 180 appended. Rows: 20607
Run 181 appended. Rows: 22178
Run 182 appended. Rows: 21673
Run 183 appended. Rows: 21971
Run 184 appended. Rows: 21890
Run 185 appended. Rows: 21917
Run 186 appended. Rows: 22483
Run 187 appended. Rows: 21878
Run 188 appended. Rows: 22729
Run 189 appended. Rows: 21935
Run 190 appended. Rows: 22248
Run 191 appended. Rows: 22492
Run 192 appended. Rows: 21878
Run 193 appended. Rows: 22729
Run 194 appended. Rows: 21943
Run 195 appended. Rows: 22278
Run 196 appended. Rows: 22492
Run 197 appended. Rows: 21878
Run 198 appended. Rows: 22729
Run 199 appended. Rows: 21945
Run 200 appended. Rows: 22278
Run 201 appended. Rows: 20021
Run 202 appended. Rows: 19728
Run 203 appended. Rows: 20109
Run 204 appended. Rows: 19706
Run 205 appended. Rows: 19943
Run 206 appended. Rows: 20348
Run 207 appended. Rows: 19739
Run 208 appended. Rows: 20371
Run 209 appended. Rows: 20216
Run 210 appended. Rows: 19864
Run 211 appended. Rows: 20456
Run 212 appended. Rows: 20028
Run 213 appended. Rows: 20425
Run 214 appended. Rows: 20085
Run 215 appended. Rows: 20069
Run 216 appended. Rows: 20817
Run 217 appended. Rows: 20184
Run 218 appended. Rows: 21065
Run 219 appended. Rows: 20530
Run 220 appended. Rows: 20613
Run 221 appended. Rows: 20728
Run 222 appended. Rows: 20392
Run 223 appended. Rows: 20935
Run 224 appended. Rows: 20604
Run 225 appended. Rows: 20766
Run 226 appended. Rows: 20963
```

```
Run 227 appended. Rows: 20659
Run 228 appended. Rows: 21130
Run 229 appended. Rows: 20858
Run 230 appended. Rows: 20831
Run 231 appended. Rows: 21484
Run 232 appended. Rows: 21274
Run 233 appended. Rows: 21820
Run 234 appended. Rows: 21877
Run 235 appended. Rows: 21587
Run 236 appended. Rows: 21872
Run 237 appended. Rows: 21498
Run 238 appended. Rows: 21958
Run 239 appended. Rows: 21788
Run 240 appended. Rows: 21741
Run 241 appended. Rows: 21739
Run 242 appended. Rows: 21687
Run 243 appended. Rows: 21971
Run 244 appended. Rows: 21885
Run 245 appended. Rows: 21761
Run 246 appended. Rows: 21681
Run 247 appended. Rows: 21785
Run 248 appended. Rows: 21971
Run 249 appended. Rows: 21929
Run 250 appended. Rows: 21917
Run 251 appended. Rows: 21786
Run 252 appended. Rows: 21785
Run 253 appended. Rows: 21971
Run 254 appended. Rows: 21929
Run 255 appended. Rows: 21901
Run 256 appended. Rows: 21806
Run 257 appended. Rows: 21785
Run 258 appended. Rows: 21971
Run 259 appended. Rows: 21929
Run 260 appended. Rows: 21917
Step 2 completed.
```

## Fill the metrics table

```
WINDOW = 120
STEP = 10

tgt.execute("DELETE FROM OEE_In_WindowMetrics;")
tgt.commit()

rows_to_insert = []

run_ids = [r["Run"] for r in tgt.execute(
    "SELECT DISTINCT Run FROM OEE_In ORDER BY Run"
).fetchall()]

for run_id in run_ids:
    df_run = pd.read_sql_query("""
        SELECT
            o.Run,
            o.TimeStamp,
            o.Total, o.Total_Cumul,
            o.NiO, o.NiO_Cumul,
            o.DeltaT, o.DeltaT_Cumul,
            o.CT_tot, o.CT_act,
```

```
            t.Buffer_Level
        FROM OEE_In o
        LEFT JOIN TS_In t
          ON t.RunID = o.Run AND t.TimeStamp = o.TimeStamp
        WHERE o.Run = ?
        ORDER BY o.TimeStamp
    """, tgt, params=(run_id,))

    n = len(df_run)
    if n < WINDOW:
        continue

    for end_idx in range(WINDOW - 1, n, STEP):
        start_idx = end_idx - WINDOW + 1
        df_window = df_run.iloc[start_idx:end_idx + 1].copy()

        base_metrics = calculate_basic_metrics(df_window.drop(columns=["Buffer_Level"],
errors="ignore"))
        if not base_metrics:
            continue

        oee_metrics = OEE_Calc(base_metrics)
        if not oee_metrics:
            continue

        end_buf = df_run.iloc[end_idx]["Buffer_Level"]
        end_buf = None if pd.isna(end_buf) else int(end_buf)

        rows_to_insert.append((
            int(run_id),
            float(df_run.iloc[end_idx]["TimeStamp"]),
            int(end_idx),
            int(WINDOW),
            int(STEP),
            end_buf,
            json.dumps(base_metrics, ensure_ascii=False),
            json.dumps(oee_metrics, ensure_ascii=False),
        ))

    print("Next RunId:", run_id, " Total rows queued:", len(rows_to_insert))

tgt.executemany("""
    INSERT OR REPLACE INTO OEE_In_WindowMetrics
    (Run, EndTimeStamp, EndRow, WindowEvents, StepEvents, EndBufferLevel,
BaseMetricsJSON, OEEMetricsJSON)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?)
""", rows_to_insert)

tgt.commit()
print("Inserted rows:", len(rows_to_insert))
```

```
Next RunId: 1  Total rows queued: 2012
Next RunId: 2  Total rows queued: 4030
Next RunId: 3  Total rows queued: 6026
Next RunId: 4  Total rows queued: 8019
Next RunId: 5  Total rows queued: 9953
Next RunId: 6  Total rows queued: 11971
Next RunId: 7  Total rows queued: 14042
Next RunId: 8  Total rows queued: 16040
Next RunId: 9  Total rows queued: 18033
Next RunId: 10  Total rows queued: 19972
Next RunId: 11  Total rows queued: 21990
Next RunId: 12  Total rows queued: 24069
Next RunId: 13  Total rows queued: 26068
Next RunId: 14  Total rows queued: 28061
Next RunId: 15  Total rows queued: 30001
Next RunId: 16  Total rows queued: 32019
Next RunId: 17  Total rows queued: 34100
Next RunId: 18  Total rows queued: 36099
Next RunId: 19  Total rows queued: 38093
Next RunId: 20  Total rows queued: 40034
Next RunId: 21  Total rows queued: 42052
Next RunId: 22  Total rows queued: 44133
Next RunId: 23  Total rows queued: 46132
Next RunId: 24  Total rows queued: 48126
Next RunId: 25  Total rows queued: 50067
Next RunId: 26  Total rows queued: 52133
Next RunId: 27  Total rows queued: 54172
Next RunId: 28  Total rows queued: 56226
Next RunId: 29  Total rows queued: 58275
Next RunId: 30  Total rows queued: 60285
Next RunId: 31  Total rows queued: 62391
Next RunId: 32  Total rows queued: 64514
Next RunId: 33  Total rows queued: 66600
Next RunId: 34  Total rows queued: 68679
Next RunId: 35  Total rows queued: 70703
Next RunId: 36  Total rows queued: 72813
Next RunId: 37  Total rows queued: 74981
Next RunId: 38  Total rows queued: 77069
Next RunId: 39  Total rows queued: 79148
Next RunId: 40  Total rows queued: 81174
Next RunId: 41  Total rows queued: 83284
Next RunId: 42  Total rows queued: 85455
Next RunId: 43  Total rows queued: 87543
Next RunId: 44  Total rows queued: 89623
Next RunId: 45  Total rows queued: 91650
Next RunId: 46  Total rows queued: 93760
Next RunId: 47  Total rows queued: 95932
Next RunId: 48  Total rows queued: 98020
Next RunId: 49  Total rows queued: 100100
Next RunId: 50  Total rows queued: 102127
Next RunId: 51  Total rows queued: 104170
Next RunId: 52  Total rows queued: 106159
Next RunId: 53  Total rows queued: 108213
Next RunId: 54  Total rows queued: 110218
Next RunId: 55  Total rows queued: 112243
Next RunId: 56  Total rows queued: 114292
Next RunId: 57  Total rows queued: 116288
Next RunId: 58  Total rows queued: 118348
Next RunId: 59  Total rows queued: 120353
Next RunId: 60  Total rows queued: 122387
```

```
Next RunId: 61  Total rows queued: 124436
Next RunId: 62  Total rows queued: 126432
Next RunId: 63  Total rows queued: 128492
Next RunId: 64  Total rows queued: 130497
Next RunId: 65  Total rows queued: 132531
Next RunId: 66  Total rows queued: 134580
Next RunId: 67  Total rows queued: 136576
Next RunId: 68  Total rows queued: 138636
Next RunId: 69  Total rows queued: 140641
Next RunId: 70  Total rows queued: 142675
Next RunId: 71  Total rows queued: 144724
Next RunId: 72  Total rows queued: 146720
Next RunId: 73  Total rows queued: 148780
Next RunId: 74  Total rows queued: 150785
Next RunId: 75  Total rows queued: 152819
Next RunId: 76  Total rows queued: 154894
Next RunId: 77  Total rows queued: 156938
Next RunId: 78  Total rows queued: 158992
Next RunId: 79  Total rows queued: 161041
Next RunId: 80  Total rows queued: 163090
Next RunId: 81  Total rows queued: 165247
Next RunId: 82  Total rows queued: 167414
Next RunId: 83  Total rows queued: 169600
Next RunId: 84  Total rows queued: 171781
Next RunId: 85  Total rows queued: 173957
Next RunId: 86  Total rows queued: 176119
Next RunId: 87  Total rows queued: 178293
Next RunId: 88  Total rows queued: 180510
Next RunId: 89  Total rows queued: 182705
Next RunId: 90  Total rows queued: 184886
Next RunId: 91  Total rows queued: 187048
Next RunId: 92  Total rows queued: 189222
Next RunId: 93  Total rows queued: 191439
Next RunId: 94  Total rows queued: 193635
Next RunId: 95  Total rows queued: 195816
Next RunId: 96  Total rows queued: 197978
Next RunId: 97  Total rows queued: 200152
Next RunId: 98  Total rows queued: 202369
Next RunId: 99  Total rows queued: 204565
Next RunId: 100  Total rows queued: 206746
Next RunId: 101  Total rows queued: 208565
Next RunId: 102  Total rows queued: 210392
Next RunId: 103  Total rows queued: 212226
Next RunId: 104  Total rows queued: 214069
Next RunId: 105  Total rows queued: 215888
Next RunId: 106  Total rows queued: 217707
Next RunId: 107  Total rows queued: 219534
Next RunId: 108  Total rows queued: 221368
Next RunId: 109  Total rows queued: 223211
Next RunId: 110  Total rows queued: 225030
Next RunId: 111  Total rows queued: 226849
Next RunId: 112  Total rows queued: 228676
Next RunId: 113  Total rows queued: 230510
Next RunId: 114  Total rows queued: 232353
Next RunId: 115  Total rows queued: 234172
Next RunId: 116  Total rows queued: 235991
Next RunId: 117  Total rows queued: 237818
Next RunId: 118  Total rows queued: 239652
Next RunId: 119  Total rows queued: 241495
Next RunId: 120  Total rows queued: 243314
```

```
Next RunId: 121  Total rows queued: 245133
Next RunId: 122  Total rows queued: 246960
Next RunId: 123  Total rows queued: 248794
Next RunId: 124  Total rows queued: 250637
Next RunId: 125  Total rows queued: 252456
Next RunId: 126  Total rows queued: 254380
Next RunId: 127  Total rows queued: 256299
Next RunId: 128  Total rows queued: 258231
Next RunId: 129  Total rows queued: 260157
Next RunId: 130  Total rows queued: 262073
Next RunId: 131  Total rows queued: 263997
Next RunId: 132  Total rows queued: 265916
Next RunId: 133  Total rows queued: 267848
Next RunId: 134  Total rows queued: 269774
Next RunId: 135  Total rows queued: 271690
Next RunId: 136  Total rows queued: 273614
Next RunId: 137  Total rows queued: 275533
Next RunId: 138  Total rows queued: 277465
Next RunId: 139  Total rows queued: 279391
Next RunId: 140  Total rows queued: 281307
Next RunId: 141  Total rows queued: 283231
Next RunId: 142  Total rows queued: 285150
Next RunId: 143  Total rows queued: 287082
Next RunId: 144  Total rows queued: 289008
Next RunId: 145  Total rows queued: 290924
Next RunId: 146  Total rows queued: 292848
Next RunId: 147  Total rows queued: 294767
Next RunId: 148  Total rows queued: 296699
Next RunId: 149  Total rows queued: 298625
Next RunId: 150  Total rows queued: 300541
Next RunId: 151  Total rows queued: 302596
Next RunId: 152  Total rows queued: 304640
Next RunId: 153  Total rows queued: 306694
Next RunId: 154  Total rows queued: 308743
Next RunId: 155  Total rows queued: 310792
Next RunId: 156  Total rows queued: 312853
Next RunId: 157  Total rows queued: 314967
Next RunId: 158  Total rows queued: 317052
Next RunId: 159  Total rows queued: 319121
Next RunId: 160  Total rows queued: 321191
Next RunId: 161  Total rows queued: 323252
Next RunId: 162  Total rows queued: 325370
Next RunId: 163  Total rows queued: 327455
Next RunId: 164  Total rows queued: 329525
Next RunId: 165  Total rows queued: 331595
Next RunId: 166  Total rows queued: 333656
Next RunId: 167  Total rows queued: 335774
Next RunId: 168  Total rows queued: 337859
Next RunId: 169  Total rows queued: 339929
Next RunId: 170  Total rows queued: 341999
Next RunId: 171  Total rows queued: 344060
Next RunId: 172  Total rows queued: 346178
Next RunId: 173  Total rows queued: 348263
Next RunId: 174  Total rows queued: 350333
Next RunId: 175  Total rows queued: 352403
Next RunId: 176  Total rows queued: 354478
Next RunId: 177  Total rows queued: 356522
Next RunId: 178  Total rows queued: 358576
Next RunId: 179  Total rows queued: 360625
Next RunId: 180  Total rows queued: 362674
```

```
Next RunId: 181  Total rows queued: 364880
Next RunId: 182  Total rows queued: 367036
Next RunId: 183  Total rows queued: 369222
Next RunId: 184  Total rows queued: 371400
Next RunId: 185  Total rows queued: 373580
Next RunId: 186  Total rows queued: 375817
Next RunId: 187  Total rows queued: 377993
Next RunId: 188  Total rows queued: 380254
Next RunId: 189  Total rows queued: 382436
Next RunId: 190  Total rows queued: 384649
Next RunId: 191  Total rows queued: 386887
Next RunId: 192  Total rows queued: 389063
Next RunId: 193  Total rows queued: 391324
Next RunId: 194  Total rows queued: 393507
Next RunId: 195  Total rows queued: 395723
Next RunId: 196  Total rows queued: 397961
Next RunId: 197  Total rows queued: 400137
Next RunId: 198  Total rows queued: 402398
Next RunId: 199  Total rows queued: 404581
Next RunId: 200  Total rows queued: 406797
Next RunId: 201  Total rows queued: 408788
Next RunId: 202  Total rows queued: 410749
Next RunId: 203  Total rows queued: 412748
Next RunId: 204  Total rows queued: 414707
Next RunId: 205  Total rows queued: 416690
Next RunId: 206  Total rows queued: 418713
Next RunId: 207  Total rows queued: 420675
Next RunId: 208  Total rows queued: 422701
Next RunId: 209  Total rows queued: 424711
Next RunId: 210  Total rows queued: 426686
Next RunId: 211  Total rows queued: 428720
Next RunId: 212  Total rows queued: 430711
Next RunId: 213  Total rows queued: 432742
Next RunId: 214  Total rows queued: 434739
Next RunId: 215  Total rows queued: 436734
Next RunId: 216  Total rows queued: 438804
Next RunId: 217  Total rows queued: 440811
Next RunId: 218  Total rows queued: 442906
Next RunId: 219  Total rows queued: 444948
Next RunId: 220  Total rows queued: 446998
Next RunId: 221  Total rows queued: 449059
Next RunId: 222  Total rows queued: 451087
Next RunId: 223  Total rows queued: 453169
Next RunId: 224  Total rows queued: 455218
Next RunId: 225  Total rows queued: 457283
Next RunId: 226  Total rows queued: 459368
Next RunId: 227  Total rows queued: 461422
Next RunId: 228  Total rows queued: 463524
Next RunId: 229  Total rows queued: 465598
Next RunId: 230  Total rows queued: 467670
Next RunId: 231  Total rows queued: 469807
Next RunId: 232  Total rows queued: 471923
Next RunId: 233  Total rows queued: 474094
Next RunId: 234  Total rows queued: 476270
Next RunId: 235  Total rows queued: 478417
Next RunId: 236  Total rows queued: 480593
Next RunId: 237  Total rows queued: 482731
Next RunId: 238  Total rows queued: 484915
Next RunId: 239  Total rows queued: 487082
Next RunId: 240  Total rows queued: 489245
```

```
Next RunId: 241  Total rows queued: 491407
Next RunId: 242  Total rows queued: 493564
Next RunId: 243  Total rows queued: 495750
Next RunId: 244  Total rows queued: 497927
Next RunId: 245  Total rows queued: 500092
Next RunId: 246  Total rows queued: 502249
Next RunId: 247  Total rows queued: 504416
Next RunId: 248  Total rows queued: 506602
Next RunId: 249  Total rows queued: 508783
Next RunId: 250  Total rows queued: 510963
Next RunId: 251  Total rows queued: 513130
Next RunId: 252  Total rows queued: 515297
Next RunId: 253  Total rows queued: 517483
Next RunId: 254  Total rows queued: 519664
Next RunId: 255  Total rows queued: 521843
Next RunId: 256  Total rows queued: 524012
Next RunId: 257  Total rows queued: 526179
Next RunId: 258  Total rows queued: 528365
Next RunId: 259  Total rows queued: 530546
Next RunId: 260  Total rows queued: 532726
Inserted rows: 532726
```

# OEE Out

## Create Metrics Table

```
tgt.execute("""
CREATE TABLE IF NOT EXISTS OEE_Out_WindowMetrics (
    Run INTEGER NOT NULL,
    EndTimeStamp REAL NOT NULL,
    EndRow INTEGER NOT NULL,
    WindowEvents INTEGER NOT NULL,
    StepEvents INTEGER NOT NULL,
    BaseMetricsJSON TEXT NOT NULL,
    OEEMetricsJSON TEXT NOT NULL,
    PRIMARY KEY (Run, EndRow)
);
""")
tgt.commit()
```

## Load TS_out into Python Dataframe / RunID

```
# Optional but recommended: clear table before rebuilding
tgt.execute("DELETE FROM OEE_Out;")
tgt.commit()

# Get all distinct runs
run_ids = [r["RunID"] for r in tgt.execute(
    "SELECT DISTINCT RunID FROM TS_Out ORDER BY RunID"
).fetchall()]

print("Runs detected:", run_ids)

for run_id in run_ids:
```

```
    df = pd.read_sql_query("""
        SELECT
            RunID AS Run,
            TimeStamp,
            Total,
            NiO
        FROM TS_Out
        WHERE RunID = ?
        ORDER BY TimeStamp
    """, tgt, params=(run_id,))

    if df.empty:
        continue

    # Copy cumulative values
    df["Total_Cumul"] = df["Total"]
    df["NiO_Cumul"]   = df["NiO"]

    # Unwind cumulative into increments
    df["Total"] = df["Total_Cumul"].diff().fillna(df["Total_Cumul"]).clip(lower=0)
    df["NiO"]   = df["NiO_Cumul"].diff().fillna(df["NiO_Cumul"]).clip(lower=0)

    # --- DeltaT ---
    df["DeltaT"] = df["TimeStamp"].diff()

    valid_dt = df["DeltaT"].iloc[1:]
    valid_dt = valid_dt[valid_dt > 0]

    dt0 = float(valid_dt.median()) if len(valid_dt) > 0 else 0.0
    df.loc[0, "DeltaT"] = dt0

    # Ensure no NaN remains
    df["DeltaT"] = df["DeltaT"].fillna(0.0)

    # Cumulative time
    df["DeltaT_Cumul"] = df["DeltaT"].cumsum()

    # Cycle times
    df["CT_act"] = df["DeltaT"]
    df["CT_tot"] = np.where(
        df["Total_Cumul"] > 0,
        df["DeltaT_Cumul"] / df["Total_Cumul"],
        0.0
    )

    # Append to OEE_Out
    df.to_sql("OEE_Out", tgt, if_exists="append", index=False)

    print(f"Run {run_id} appended. Rows: {len(df)}")

tgt.commit()
print("OEE_Out rebuild completed.")
```

```
Runs detected: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,
65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106,
107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124,
125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160,
161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178,
179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196,
197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214,
215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232,
233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250,
251, 252, 253, 254, 255, 256, 257, 258, 259, 260]
Run 1 appended. Rows: 20023
Run 2 appended. Rows: 20080
Run 3 appended. Rows: 19869
Run 4 appended. Rows: 19834
Run 5 appended. Rows: 19245
Run 6 appended. Rows: 20085
Run 7 appended. Rows: 20619
Run 8 appended. Rows: 19886
Run 9 appended. Rows: 19838
Run 10 appended. Rows: 19295
Run 11 appended. Rows: 20086
Run 12 appended. Rows: 20697
Run 13 appended. Rows: 19892
Run 14 appended. Rows: 19838
Run 15 appended. Rows: 19303
Run 16 appended. Rows: 20086
Run 17 appended. Rows: 20713
Run 18 appended. Rows: 19896
Run 19 appended. Rows: 19840
Run 20 appended. Rows: 19312
Run 21 appended. Rows: 20086
Run 22 appended. Rows: 20713
Run 23 appended. Rows: 19896
Run 24 appended. Rows: 19843
Run 25 appended. Rows: 19314
Run 26 appended. Rows: 20560
Run 27 appended. Rows: 20295
Run 28 appended. Rows: 20510
Run 29 appended. Rows: 20542
Run 30 appended. Rows: 20006
Run 31 appended. Rows: 20962
Run 32 appended. Rows: 21130
Run 33 appended. Rows: 20766
Run 34 appended. Rows: 20697
Run 35 appended. Rows: 20144
Run 36 appended. Rows: 21000
Run 37 appended. Rows: 21589
Run 38 appended. Rows: 20781
Run 39 appended. Rows: 20699
Run 40 appended. Rows: 20166
Run 41 appended. Rows: 21000
Run 42 appended. Rows: 21611
Run 43 appended. Rows: 20786
Run 44 appended. Rows: 20700
Run 45 appended. Rows: 20174
Run 46 appended. Rows: 21000
```

```
Run 47 appended. Rows: 21623
Run 48 appended. Rows: 20788
Run 49 appended. Rows: 20701
Run 50 appended. Rows: 20179
Run 51 appended. Rows: 20330
Run 52 appended. Rows: 19791
Run 53 appended. Rows: 20471
Run 54 appended. Rows: 19950
Run 55 appended. Rows: 20157
Run 56 appended. Rows: 20394
Run 57 appended. Rows: 19861
Run 58 appended. Rows: 20500
Run 59 appended. Rows: 19957
Run 60 appended. Rows: 20241
Run 61 appended. Rows: 20394
Run 62 appended. Rows: 19861
Run 63 appended. Rows: 20500
Run 64 appended. Rows: 19959
Run 65 appended. Rows: 20241
Run 66 appended. Rows: 20394
Run 67 appended. Rows: 19861
Run 68 appended. Rows: 20500
Run 69 appended. Rows: 19959
Run 70 appended. Rows: 20241
Run 71 appended. Rows: 20394
Run 72 appended. Rows: 19861
Run 73 appended. Rows: 20500
Run 74 appended. Rows: 19959
Run 75 appended. Rows: 20241
Run 76 appended. Rows: 20826
Run 77 appended. Rows: 20551
Run 78 appended. Rows: 20652
Run 79 appended. Rows: 20600
Run 80 appended. Rows: 20603
Run 81 appended. Rows: 21470
Run 82 appended. Rows: 21596
Run 83 appended. Rows: 21947
Run 84 appended. Rows: 21735
Run 85 appended. Rows: 21661
Run 86 appended. Rows: 21523
Run 87 appended. Rows: 21648
Run 88 appended. Rows: 22078
Run 89 appended. Rows: 21852
Run 90 appended. Rows: 21715
Run 91 appended. Rows: 21523
Run 92 appended. Rows: 21648
Run 93 appended. Rows: 22078
Run 94 appended. Rows: 21860
Run 95 appended. Rows: 21718
Run 96 appended. Rows: 21523
Run 97 appended. Rows: 21648
Run 98 appended. Rows: 22078
Run 99 appended. Rows: 21862
Run 100 appended. Rows: 21718
Run 101 appended. Rows: 18096
Run 102 appended. Rows: 18170
Run 103 appended. Rows: 18246
Run 104 appended. Rows: 18339
Run 105 appended. Rows: 18094
Run 106 appended. Rows: 18096
```

```
Run 107 appended. Rows: 18170
Run 108 appended. Rows: 18246
Run 109 appended. Rows: 18339
Run 110 appended. Rows: 18094
Run 111 appended. Rows: 18096
Run 112 appended. Rows: 18170
Run 113 appended. Rows: 18246
Run 114 appended. Rows: 18339
Run 115 appended. Rows: 18094
Run 116 appended. Rows: 18096
Run 117 appended. Rows: 18170
Run 118 appended. Rows: 18246
Run 119 appended. Rows: 18339
Run 120 appended. Rows: 18094
Run 121 appended. Rows: 18096
Run 122 appended. Rows: 18170
Run 123 appended. Rows: 18246
Run 124 appended. Rows: 18339
Run 125 appended. Rows: 18094
Run 126 appended. Rows: 19149
Run 127 appended. Rows: 19096
Run 128 appended. Rows: 19220
Run 129 appended. Rows: 19162
Run 130 appended. Rows: 19066
Run 131 appended. Rows: 19149
Run 132 appended. Rows: 19096
Run 133 appended. Rows: 19220
Run 134 appended. Rows: 19169
Run 135 appended. Rows: 19067
Run 136 appended. Rows: 19149
Run 137 appended. Rows: 19096
Run 138 appended. Rows: 19220
Run 139 appended. Rows: 19169
Run 140 appended. Rows: 19067
Run 141 appended. Rows: 19149
Run 142 appended. Rows: 19096
Run 143 appended. Rows: 19220
Run 144 appended. Rows: 19169
Run 145 appended. Rows: 19067
Run 146 appended. Rows: 19149
Run 147 appended. Rows: 19096
Run 148 appended. Rows: 19220
Run 149 appended. Rows: 19169
Run 150 appended. Rows: 19067
Run 151 appended. Rows: 20454
Run 152 appended. Rows: 20452
Run 153 appended. Rows: 20546
Run 154 appended. Rows: 20552
Run 155 appended. Rows: 20549
Run 156 appended. Rows: 20511
Run 157 appended. Rows: 21040
Run 158 appended. Rows: 20753
Run 159 appended. Rows: 20598
Run 160 appended. Rows: 20600
Run 161 appended. Rows: 20511
Run 162 appended. Rows: 21089
Run 163 appended. Rows: 20753
Run 164 appended. Rows: 20601
Run 165 appended. Rows: 20607
Run 166 appended. Rows: 20511
```

```
Run 167 appended. Rows: 21089
Run 168 appended. Rows: 20753
Run 169 appended. Rows: 20601
Run 170 appended. Rows: 20607
Run 171 appended. Rows: 20511
Run 172 appended. Rows: 21089
Run 173 appended. Rows: 20753
Run 174 appended. Rows: 20601
Run 175 appended. Rows: 20607
Run 176 appended. Rows: 20820
Run 177 appended. Rows: 20446
Run 178 appended. Rows: 20652
Run 179 appended. Rows: 20600
Run 180 appended. Rows: 20546
Run 181 appended. Rows: 21968
Run 182 appended. Rows: 21463
Run 183 appended. Rows: 21940
Run 184 appended. Rows: 21680
Run 185 appended. Rows: 21729
Run 186 appended. Rows: 22273
Run 187 appended. Rows: 21668
Run 188 appended. Rows: 22519
Run 189 appended. Rows: 21725
Run 190 appended. Rows: 22038
Run 191 appended. Rows: 22282
Run 192 appended. Rows: 21668
Run 193 appended. Rows: 22519
Run 194 appended. Rows: 21733
Run 195 appended. Rows: 22068
Run 196 appended. Rows: 22282
Run 197 appended. Rows: 21668
Run 198 appended. Rows: 22519
Run 199 appended. Rows: 21735
Run 200 appended. Rows: 22068
Run 201 appended. Rows: 19811
Run 202 appended. Rows: 19518
Run 203 appended. Rows: 19899
Run 204 appended. Rows: 19496
Run 205 appended. Rows: 19733
Run 206 appended. Rows: 20138
Run 207 appended. Rows: 19529
Run 208 appended. Rows: 20161
Run 209 appended. Rows: 20006
Run 210 appended. Rows: 19654
Run 211 appended. Rows: 20246
Run 212 appended. Rows: 19818
Run 213 appended. Rows: 20215
Run 214 appended. Rows: 19875
Run 215 appended. Rows: 19859
Run 216 appended. Rows: 20607
Run 217 appended. Rows: 19974
Run 218 appended. Rows: 20855
Run 219 appended. Rows: 20320
Run 220 appended. Rows: 20403
Run 221 appended. Rows: 20518
Run 222 appended. Rows: 20182
Run 223 appended. Rows: 20725
Run 224 appended. Rows: 20394
Run 225 appended. Rows: 20556
Run 226 appended. Rows: 20753
```

```
Run 227 appended. Rows: 20449
Run 228 appended. Rows: 20920
Run 229 appended. Rows: 20648
Run 230 appended. Rows: 20621
Run 231 appended. Rows: 21274
Run 232 appended. Rows: 21064
Run 233 appended. Rows: 21610
Run 234 appended. Rows: 21667
Run 235 appended. Rows: 21377
Run 236 appended. Rows: 21662
Run 237 appended. Rows: 21288
Run 238 appended. Rows: 21748
Run 239 appended. Rows: 21578
Run 240 appended. Rows: 21531
Run 241 appended. Rows: 21529
Run 242 appended. Rows: 21477
Run 243 appended. Rows: 21828
Run 244 appended. Rows: 21675
Run 245 appended. Rows: 21551
Run 246 appended. Rows: 21471
Run 247 appended. Rows: 21680
Run 248 appended. Rows: 21947
Run 249 appended. Rows: 21772
Run 250 appended. Rows: 21721
Run 251 appended. Rows: 21576
Run 252 appended. Rows: 21708
Run 253 appended. Rows: 21947
Run 254 appended. Rows: 21804
Run 255 appended. Rows: 21691
Run 256 appended. Rows: 21596
Run 257 appended. Rows: 21674
Run 258 appended. Rows: 21942
Run 259 appended. Rows: 21799
Run 260 appended. Rows: 21753
OEE_Out rebuild completed.
```

## Fill the metrics table

```
WINDOW = 120
STEP = 10

tgt.execute("DELETE FROM OEE_Out_WindowMetrics;")
tgt.commit()

rows_to_insert = []

run_ids = [r["Run"] for r in tgt.execute(
    "SELECT DISTINCT Run FROM OEE_Out ORDER BY Run"
).fetchall()]

for run_id in run_ids:
    df_run = pd.read_sql_query("""
        SELECT Run, TimeStamp, Total, Total_Cumul, NiO, NiO_Cumul, DeltaT, DeltaT_Cumul,
CT_tot, CT_act
        FROM OEE_Out
        WHERE Run = ?
        ORDER BY TimeStamp
    """, tgt, params=(run_id,))
```

```
    n = len(df_run)
    if n < WINDOW:
        continue

    for end_idx in range(WINDOW - 1, n, STEP):
        start_idx = end_idx - WINDOW + 1
        df_window = df_run.iloc[start_idx:end_idx + 1]

        base_metrics = calculate_basic_metrics(df_window)
        if not base_metrics:
            continue

        oee_metrics = OEE_Calc(base_metrics)
        if not oee_metrics:
            continue

        rows_to_insert.append((
            int(run_id),
            float(df_run.iloc[end_idx]["TimeStamp"]),
            int(end_idx),
            int(WINDOW),
            int(STEP),
            json.dumps(base_metrics, ensure_ascii=False),
            json.dumps(oee_metrics, ensure_ascii=False),
        ))

    print("Next RunId:", run_id, " Total rows queued:", len(rows_to_insert))

tgt.executemany("""
    INSERT OR REPLACE INTO OEE_Out_WindowMetrics
    (Run, EndTimeStamp, EndRow, WindowEvents, StepEvents, BaseMetricsJSON,
OEEMetricsJSON)
    VALUES (?, ?, ?, ?, ?, ?, ?)
""", rows_to_insert)

tgt.commit()
print("Inserted rows:", len(rows_to_insert))
```

```
Next RunId: 1  Total rows queued: 1991
Next RunId: 2  Total rows queued: 3988
Next RunId: 3  Total rows queued: 5963
Next RunId: 4  Total rows queued: 7935
Next RunId: 5  Total rows queued: 9848
Next RunId: 6  Total rows queued: 11845
Next RunId: 7  Total rows queued: 13895
Next RunId: 8  Total rows queued: 15872
Next RunId: 9  Total rows queued: 17844
Next RunId: 10  Total rows queued: 19762
Next RunId: 11  Total rows queued: 21759
Next RunId: 12  Total rows queued: 23817
Next RunId: 13  Total rows queued: 25795
Next RunId: 14  Total rows queued: 27767
Next RunId: 15  Total rows queued: 29686
Next RunId: 16  Total rows queued: 31683
Next RunId: 17  Total rows queued: 33743
Next RunId: 18  Total rows queued: 35721
Next RunId: 19  Total rows queued: 37694
Next RunId: 20  Total rows queued: 39614
Next RunId: 21  Total rows queued: 41611
Next RunId: 22  Total rows queued: 43671
Next RunId: 23  Total rows queued: 45649
Next RunId: 24  Total rows queued: 47622
Next RunId: 25  Total rows queued: 49542
Next RunId: 26  Total rows queued: 51587
Next RunId: 27  Total rows queued: 53605
Next RunId: 28  Total rows queued: 55645
Next RunId: 29  Total rows queued: 57688
Next RunId: 30  Total rows queued: 59677
Next RunId: 31  Total rows queued: 61762
Next RunId: 32  Total rows queued: 63864
Next RunId: 33  Total rows queued: 65929
Next RunId: 34  Total rows queued: 67987
Next RunId: 35  Total rows queued: 69990
Next RunId: 36  Total rows queued: 72079
Next RunId: 37  Total rows queued: 74226
Next RunId: 38  Total rows queued: 76293
Next RunId: 39  Total rows queued: 78351
Next RunId: 40  Total rows queued: 80356
Next RunId: 41  Total rows queued: 82445
Next RunId: 42  Total rows queued: 84595
Next RunId: 43  Total rows queued: 86662
Next RunId: 44  Total rows queued: 88721
Next RunId: 45  Total rows queued: 90727
Next RunId: 46  Total rows queued: 92816
Next RunId: 47  Total rows queued: 94967
Next RunId: 48  Total rows queued: 97034
Next RunId: 49  Total rows queued: 99093
Next RunId: 50  Total rows queued: 101099
Next RunId: 51  Total rows queued: 103121
Next RunId: 52  Total rows queued: 105089
Next RunId: 53  Total rows queued: 107125
Next RunId: 54  Total rows queued: 109109
Next RunId: 55  Total rows queued: 111113
Next RunId: 56  Total rows queued: 113141
Next RunId: 57  Total rows queued: 115116
Next RunId: 58  Total rows queued: 117155
Next RunId: 59  Total rows queued: 119139
Next RunId: 60  Total rows queued: 121152
```

```
Next RunId: 61  Total rows queued: 123180
Next RunId: 62  Total rows queued: 125155
Next RunId: 63  Total rows queued: 127194
Next RunId: 64  Total rows queued: 129178
Next RunId: 65  Total rows queued: 131191
Next RunId: 66  Total rows queued: 133219
Next RunId: 67  Total rows queued: 135194
Next RunId: 68  Total rows queued: 137233
Next RunId: 69  Total rows queued: 139217
Next RunId: 70  Total rows queued: 141230
Next RunId: 71  Total rows queued: 143258
Next RunId: 72  Total rows queued: 145233
Next RunId: 73  Total rows queued: 147272
Next RunId: 74  Total rows queued: 149256
Next RunId: 75  Total rows queued: 151269
Next RunId: 76  Total rows queued: 153340
Next RunId: 77  Total rows queued: 155384
Next RunId: 78  Total rows queued: 157438
Next RunId: 79  Total rows queued: 159487
Next RunId: 80  Total rows queued: 161536
Next RunId: 81  Total rows queued: 163672
Next RunId: 82  Total rows queued: 165820
Next RunId: 83  Total rows queued: 168003
Next RunId: 84  Total rows queued: 170165
Next RunId: 85  Total rows queued: 172320
Next RunId: 86  Total rows queued: 174461
Next RunId: 87  Total rows queued: 176614
Next RunId: 88  Total rows queued: 178810
Next RunId: 89  Total rows queued: 180984
Next RunId: 90  Total rows queued: 183144
Next RunId: 91  Total rows queued: 185285
Next RunId: 92  Total rows queued: 187438
Next RunId: 93  Total rows queued: 189634
Next RunId: 94  Total rows queued: 191809
Next RunId: 95  Total rows queued: 193969
Next RunId: 96  Total rows queued: 196110
Next RunId: 97  Total rows queued: 198263
Next RunId: 98  Total rows queued: 200459
Next RunId: 99  Total rows queued: 202634
Next RunId: 100  Total rows queued: 204794
Next RunId: 101  Total rows queued: 206592
Next RunId: 102  Total rows queued: 208398
Next RunId: 103  Total rows queued: 210211
Next RunId: 104  Total rows queued: 212033
Next RunId: 105  Total rows queued: 213831
Next RunId: 106  Total rows queued: 215629
Next RunId: 107  Total rows queued: 217435
Next RunId: 108  Total rows queued: 219248
Next RunId: 109  Total rows queued: 221070
Next RunId: 110  Total rows queued: 222868
Next RunId: 111  Total rows queued: 224666
Next RunId: 112  Total rows queued: 226472
Next RunId: 113  Total rows queued: 228285
Next RunId: 114  Total rows queued: 230107
Next RunId: 115  Total rows queued: 231905
Next RunId: 116  Total rows queued: 233703
Next RunId: 117  Total rows queued: 235509
Next RunId: 118  Total rows queued: 237322
Next RunId: 119  Total rows queued: 239144
Next RunId: 120  Total rows queued: 240942
```

```
Next RunId: 121  Total rows queued: 242740
Next RunId: 122  Total rows queued: 244546
Next RunId: 123  Total rows queued: 246359
Next RunId: 124  Total rows queued: 248181
Next RunId: 125  Total rows queued: 249979
Next RunId: 126  Total rows queued: 251882
Next RunId: 127  Total rows queued: 253780
Next RunId: 128  Total rows queued: 255691
Next RunId: 129  Total rows queued: 257596
Next RunId: 130  Total rows queued: 259491
Next RunId: 131  Total rows queued: 261394
Next RunId: 132  Total rows queued: 263292
Next RunId: 133  Total rows queued: 265203
Next RunId: 134  Total rows queued: 267108
Next RunId: 135  Total rows queued: 269003
Next RunId: 136  Total rows queued: 270906
Next RunId: 137  Total rows queued: 272804
Next RunId: 138  Total rows queued: 274715
Next RunId: 139  Total rows queued: 276620
Next RunId: 140  Total rows queued: 278515
Next RunId: 141  Total rows queued: 280418
Next RunId: 142  Total rows queued: 282316
Next RunId: 143  Total rows queued: 284227
Next RunId: 144  Total rows queued: 286132
Next RunId: 145  Total rows queued: 288027
Next RunId: 146  Total rows queued: 289930
Next RunId: 147  Total rows queued: 291828
Next RunId: 148  Total rows queued: 293739
Next RunId: 149  Total rows queued: 295644
Next RunId: 150  Total rows queued: 297539
Next RunId: 151  Total rows queued: 299573
Next RunId: 152  Total rows queued: 301607
Next RunId: 153  Total rows queued: 303650
Next RunId: 154  Total rows queued: 305694
Next RunId: 155  Total rows queued: 307737
Next RunId: 156  Total rows queued: 309777
Next RunId: 157  Total rows queued: 311870
Next RunId: 158  Total rows queued: 313934
Next RunId: 159  Total rows queued: 315982
Next RunId: 160  Total rows queued: 318031
Next RunId: 161  Total rows queued: 320071
Next RunId: 162  Total rows queued: 322168
Next RunId: 163  Total rows queued: 324232
Next RunId: 164  Total rows queued: 326281
Next RunId: 165  Total rows queued: 328330
Next RunId: 166  Total rows queued: 330370
Next RunId: 167  Total rows queued: 332467
Next RunId: 168  Total rows queued: 334531
Next RunId: 169  Total rows queued: 336580
Next RunId: 170  Total rows queued: 338629
Next RunId: 171  Total rows queued: 340669
Next RunId: 172  Total rows queued: 342766
Next RunId: 173  Total rows queued: 344830
Next RunId: 174  Total rows queued: 346879
Next RunId: 175  Total rows queued: 348928
Next RunId: 176  Total rows queued: 350999
Next RunId: 177  Total rows queued: 353032
Next RunId: 178  Total rows queued: 355086
Next RunId: 179  Total rows queued: 357135
Next RunId: 180  Total rows queued: 359178
```

```
Next RunId: 181  Total rows queued: 361363
Next RunId: 182  Total rows queued: 363498
Next RunId: 183  Total rows queued: 365681
Next RunId: 184  Total rows queued: 367838
Next RunId: 185  Total rows queued: 369999
Next RunId: 186  Total rows queued: 372215
Next RunId: 187  Total rows queued: 374370
Next RunId: 188  Total rows queued: 376610
Next RunId: 189  Total rows queued: 378771
Next RunId: 190  Total rows queued: 380963
Next RunId: 191  Total rows queued: 383180
Next RunId: 192  Total rows queued: 385335
Next RunId: 193  Total rows queued: 387575
Next RunId: 194  Total rows queued: 389737
Next RunId: 195  Total rows queued: 391932
Next RunId: 196  Total rows queued: 394149
Next RunId: 197  Total rows queued: 396304
Next RunId: 198  Total rows queued: 398544
Next RunId: 199  Total rows queued: 400706
Next RunId: 200  Total rows queued: 402901
Next RunId: 201  Total rows queued: 404871
Next RunId: 202  Total rows queued: 406811
Next RunId: 203  Total rows queued: 408789
Next RunId: 204  Total rows queued: 410727
Next RunId: 205  Total rows queued: 412689
Next RunId: 206  Total rows queued: 414691
Next RunId: 207  Total rows queued: 416632
Next RunId: 208  Total rows queued: 418637
Next RunId: 209  Total rows queued: 420626
Next RunId: 210  Total rows queued: 422580
Next RunId: 211  Total rows queued: 424593
Next RunId: 212  Total rows queued: 426563
Next RunId: 213  Total rows queued: 428573
Next RunId: 214  Total rows queued: 430549
Next RunId: 215  Total rows queued: 432523
Next RunId: 216  Total rows queued: 434572
Next RunId: 217  Total rows queued: 436558
Next RunId: 218  Total rows queued: 438632
Next RunId: 219  Total rows queued: 440653
Next RunId: 220  Total rows queued: 442682
Next RunId: 221  Total rows queued: 444722
Next RunId: 222  Total rows queued: 446729
Next RunId: 223  Total rows queued: 448790
Next RunId: 224  Total rows queued: 450818
Next RunId: 225  Total rows queued: 452862
Next RunId: 226  Total rows queued: 454926
Next RunId: 227  Total rows queued: 456959
Next RunId: 228  Total rows queued: 459040
Next RunId: 229  Total rows queued: 461093
Next RunId: 230  Total rows queued: 463144
Next RunId: 231  Total rows queued: 465260
Next RunId: 232  Total rows queued: 467355
Next RunId: 233  Total rows queued: 469505
Next RunId: 234  Total rows queued: 471660
Next RunId: 235  Total rows queued: 473786
Next RunId: 236  Total rows queued: 475941
Next RunId: 237  Total rows queued: 478058
Next RunId: 238  Total rows queued: 480221
Next RunId: 239  Total rows queued: 482367
Next RunId: 240  Total rows queued: 484509
```

```
Next RunId: 241  Total rows queued: 486650
Next RunId: 242  Total rows queued: 488786
Next RunId: 243  Total rows queued: 490957
Next RunId: 244  Total rows queued: 493113
Next RunId: 245  Total rows queued: 495257
Next RunId: 246  Total rows queued: 497393
Next RunId: 247  Total rows queued: 499550
Next RunId: 248  Total rows queued: 501733
Next RunId: 249  Total rows queued: 503899
Next RunId: 250  Total rows queued: 506060
Next RunId: 251  Total rows queued: 508206
Next RunId: 252  Total rows queued: 510365
Next RunId: 253  Total rows queued: 512548
Next RunId: 254  Total rows queued: 514717
Next RunId: 255  Total rows queued: 516875
Next RunId: 256  Total rows queued: 519023
Next RunId: 257  Total rows queued: 521179
Next RunId: 258  Total rows queued: 523362
Next RunId: 259  Total rows queued: 525530
Next RunId: 260  Total rows queued: 527694
Inserted rows: 527694
```

# Closing the DB

```
tgt.commit()
tgt.close()
print("Connections closed cleanly.")
```

```
Connections closed cleanly.
```

# C Survival-BNN Notebook

# Imports

```
import sqlite3
import numpy as np
import pandas as pd

import torch
import torch.nn as nn

import pyro
import pyro.distributions as dist
from pyro.nn import PyroModule, PyroSample
from pyro.infer import SVI, Trace_ELBO, Predictive
from pyro.infer.autoguide import AutoDiagonalNormal

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score,
accuracy_score
```

# Reproducibility

```
SEED = 42
np.random.seed(SEED)
torch.manual_seed(SEED)
pyro.set_rng_seed(SEED)
pyro.clear_param_store()

DEVICE = torch.device("cpu")  # you installed CPU torch
```

# Load data from SQLite

```
DB_PATH = "normalized_Exp_DB_892.db"  # adjust if needed

def read_table(db_path, table_name):
    with sqlite3.connect(db_path) as con:
        return pd.read_sql_query(f"SELECT * FROM {table_name};", con)

df_train = read_table(DB_PATH, "BNN_Train")
df_val   = read_table(DB_PATH, "BNN_Val")
df_test  = read_table(DB_PATH, "BNN_Test")

print("Shapes:", df_train.shape, df_val.shape, df_test.shape)
df_train.head()
```

```
Shapes: (319442, 18) (105912, 18) (105355, 18)
```

| | Run | In_EndTimeStamp | y | Dt_ToNextMaint | DeltaT | CT_Median_In | CT_IQR_In | CT_P95_In | C |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1.735715e+09 | 0 | 3099.781487 | 155.178588 | 26.4 | 38.9 | 105.9 | |
| **1** | 1 | 1.735716e+09 | 0 | 2832.492627 | 112.110272 | 28.1 | 39.1 | 106.2 | |
| **2** | 1 | 1.735716e+09 | 0 | 2516.325295 | 74.747890 | 25.2 | 38.4 | 105.2 | |
| **3** | 1 | 1.735716e+09 | 0 | 2110.632948 | 148.440237 | 26.0 | 38.3 | 105.8 | |
| **4** | 1 | 1.735717e+09 | 0 | 1623.988375 | 303.084810 | 27.5 | 38.8 | 107.7 | |

# Define feature columns

```
LABEL_COL = "y"
DROP_COLS = ["Run", "In_EndTimeStamp"]  # not used as features

def prepare_df(df):
    df = df.copy()

    # Label -> int 0/1
    df[LABEL_COL] = pd.to_numeric(df[LABEL_COL], errors="coerce")
    df = df.dropna(subset=[LABEL_COL])
    df[LABEL_COL] = df[LABEL_COL].astype(int)

    # Ensure TimeSinceMaintEnd numeric if present
    if "TimeSinceMaintEnd" in df.columns:
        df["TimeSinceMaintEnd"] = pd.to_numeric(df["TimeSinceMaintEnd"], errors="coerce")

    # Build numeric feature list
    feature_cols = []
    for c in df.columns:
        if c == LABEL_COL or c in DROP_COLS:
            continue
        if pd.api.types.is_numeric_dtype(df[c]):
            feature_cols.append(c)

    # Median impute feature NaNs
    for c in feature_cols:
        if df[c].isna().any():
            df[c] = df[c].fillna(df[c].median())

    return df, feature_cols

df_train, FEATURE_COLS = prepare_df(df_train)
df_val,   _            = prepare_df(df_val)
df_test,  _            = prepare_df(df_test)

print("Features:", FEATURE_COLS)
print(df_train[FEATURE_COLS].dtypes)
print("Class balance train:\n", df_train[LABEL_COL].value_counts(normalize=True))
```

```
Features: ['Dt_ToNextMaint', 'DeltaT', 'CT_Median_In', 'CT_IQR_In', 'CT_P95_In',
'CT_Median_Out', 'CT_IQR_Out', 'CT_P95_Out', 'DowntimeWin', 'FailCountWin', 'MTTRWin',
'TimeSinceFailEnd', 'HasFailureHistory', 'TimeSinceMaintEnd', 'BufferLevel']
Dt_ToNextMaint       float64
DeltaT               float64
CT_Median_In         float64
CT_IQR_In            float64
CT_P95_In            float64
CT_Median_Out        float64
CT_IQR_Out           float64
CT_P95_Out           float64
DowntimeWin          float64
FailCountWin         float64
MTTRWin              float64
TimeSinceFailEnd     float64
HasFailureHistory      int64
TimeSinceMaintEnd    float64
BufferLevel            int64
dtype: object
Class balance train:
 y
1    0.814038
0    0.185962
Name: proportion, dtype: float64
```

# Convert to tensors + standardize

```
scaler = StandardScaler()

X_train = scaler.fit_transform(df_train[FEATURE_COLS].values.astype(np.float32))
X_val   = scaler.transform(df_val[FEATURE_COLS].values.astype(np.float32))
X_test  = scaler.transform(df_test[FEATURE_COLS].values.astype(np.float32))

y_train = df_train[LABEL_COL].values.astype(np.float32)
y_val   = df_val[LABEL_COL].values.astype(np.float32)
y_test  = df_test[LABEL_COL].values.astype(np.float32)

X_train_t = torch.tensor(X_train, dtype=torch.float32, device=DEVICE)
X_val_t   = torch.tensor(X_val, dtype=torch.float32, device=DEVICE)
X_test_t  = torch.tensor(X_test, dtype=torch.float32, device=DEVICE)

y_train_t = torch.tensor(y_train, dtype=torch.float32, device=DEVICE)
y_val_t   = torch.tensor(y_val, dtype=torch.float32, device=DEVICE)
y_test_t  = torch.tensor(y_test, dtype=torch.float32, device=DEVICE)

X_train_t.shape, y_train_t.shape
```

```
(torch.Size([319442, 15]), torch.Size([319442]))
```

# Define a Bayesian model

```
class BayesianLastLayerBNN(PyroModule):
    def __init__(self, in_dim, hidden_dim=32):
        super().__init__()
```

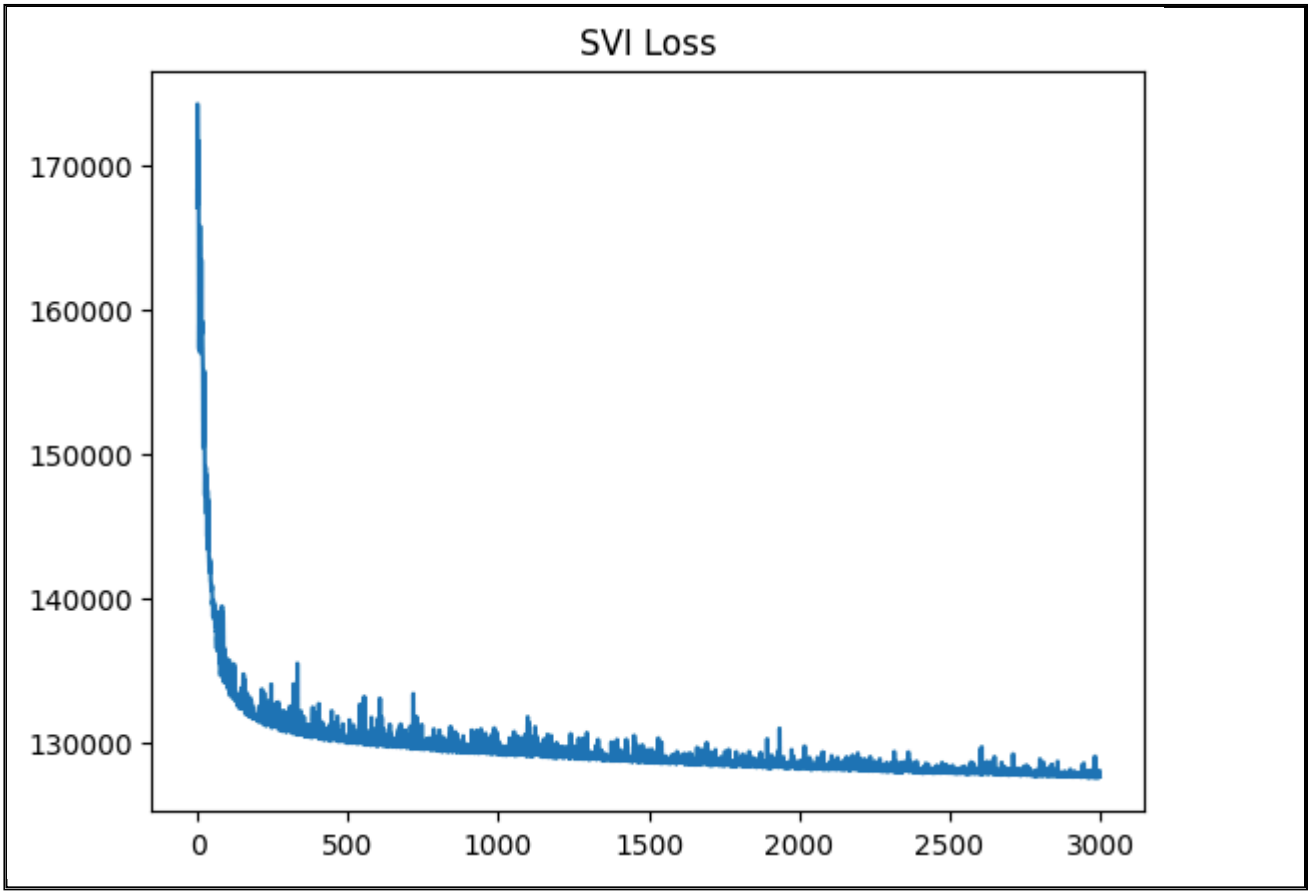
```
        self.encoder = nn.Sequential(
            nn.Linear(in_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
        )

        self.out = PyroModule[nn.Linear](hidden_dim, 1)
        self.out.weight = PyroSample(dist.Normal(0., 1.).expand([1,
hidden_dim]).to_event(2))
        self.out.bias   = PyroSample(dist.Normal(0., 1.).expand([1]).to_event(1))

    def forward(self, x, y=None):
        z = self.encoder(x)
        logits = self.out(z).squeeze(-1)  # [N]
        with pyro.plate("data", x.shape[0]):
            pyro.sample("obs", dist.Bernoulli(logits=logits), obs=y)
        return logits
```

# Train with SVI

```
pyro.clear_param_store()

in_dim = X_train_t.shape[1]
model = BayesianLastLayerBNN(in_dim=in_dim, hidden_dim=32).to(DEVICE)

guide = AutoDiagonalNormal(model)
optimizer = pyro.optim.Adam({"lr": 1e-3})
svi = SVI(model, guide, optimizer, loss=Trace_ELBO())

def train_svi(num_steps=3000, print_every=300):
    losses = []
    for step in range(1, num_steps + 1):
        loss = svi.step(X_train_t, y_train_t)
        losses.append(loss)
        if step % print_every == 0:
            avg = float(np.mean(losses[-print_every:]))
            print(f"step {step:5d} | loss {avg:.2f}")
    return losses

losses = train_svi(num_steps=3000, print_every=300)
```

```
step   300 | loss 136396.09
step   600 | loss 130646.51
step   900 | loss 129910.26
step  1200 | loss 129509.51
step  1500 | loss 129076.01
step  1800 | loss 128745.89
step  2100 | loss 128508.81
step  2400 | loss 128279.63
step  2700 | loss 128064.90
step  3000 | loss 127857.89
```

```
import matplotlib.pyplot as plt
plt.plot(losses)
plt.title("SVI Loss")
plt.show()
```




# Posterior predictive sampling

```
@torch.no_grad()
def posterior_predict_survival(model, guide, x_tensor, num_samples=500):
    predictive = Predictive(model, guide=guide, num_samples=num_samples, return_sites=
("obs",))
    obs_samples = predictive(x_tensor, y=None)["obs"]  # [S, N], samples of y in {0,1}
    p_mean = obs_samples.float().mean(dim=0).cpu().numpy()
    p_std  = obs_samples.float().std(dim=0).cpu().numpy()
    return p_mean, p_std

p_val_mean,  p_val_std  = posterior_predict_survival(model, guide, X_val_t,
num_samples=500)
p_test_mean, p_test_std = posterior_predict_survival(model, guide, X_test_t,
num_samples=500)

print("Posterior predictive computed.")
print("VAL P(survive) first 5:", p_val_mean[:5])
print("Mean predictive std (VAL):", float(np.mean(p_val_std)))
```

```
Posterior predictive computed.
VAL P(survive) first 5: [0.734 0.838 0.888 0.938 0.982]
Mean predictive std (VAL): 0.33259788155555725
```

# Classical evaluation at threshold 0.5

```
def eval_threshold(name, y_true, p_survive, threshold=0.5):
    y_pred = (p_survive >= threshold).astype(int)
```

```
    acc = accuracy_score(y_true, y_pred)
    auc = roc_auc_score(y_true, p_survive) if len(np.unique(y_true)) > 1 else np.nan

    print(f"\n=== {name} (threshold={threshold}) ===")
    print("Accuracy:", acc)
    print("ROC AUC :", auc)
    print("Confusion matrix:\n", confusion_matrix(y_true, y_pred))
    print("\nReport:\n", classification_report(y_true, y_pred, digits=4))

eval_threshold("VAL",  y_val,  p_val_mean,  threshold=0.5)
eval_threshold("TEST", y_test, p_test_mean, threshold=0.5)
```

```
=== VAL (threshold=0.5) ===
Accuracy: 0.8035916610015862
ROC AUC : 0.758414951888521
Confusion matrix:
 [[ 2604 17905]
 [ 2897 82506]]

Report:
               precision    recall  f1-score   support

         0.0     0.4734    0.1270    0.2002     20509
         1.0     0.8217    0.9661    0.8880     85403

    accuracy                         0.8036    105912
   macro avg     0.6475    0.5465    0.5441    105912
weighted avg     0.7542    0.8036    0.7549    105912


=== TEST (threshold=0.5) ===
Accuracy: 0.8139338427222249
ROC AUC : 0.7556135134157711
Confusion matrix:
 [[ 2784 17116]
 [ 2487 82968]]

Report:
               precision    recall  f1-score   support

         0.0     0.5282    0.1399    0.2212     19900
         1.0     0.8290    0.9709    0.8943     85455

    accuracy                         0.8139    105355
   macro avg     0.6786    0.5554    0.5578    105355
weighted avg     0.7722    0.8139    0.7672    105355
```

# Decision rule for maintenance

```
BETA = 0.15  # <-- change this value (e.g., 0.15, 0.20, 0.25)

def maintenance_decision(p_survive, beta=BETA):
    p_fail = 1.0 - p_survive
    trigger = (p_fail > beta).astype(int)  # 1 = trigger maintenance (predict failure)
    y_pred = 1 - trigger  # convert to predicted y (1 survive, 0 fail)
    return trigger, p_fail, y_pred
```

```
def eval_maintenance(name, y_true, p_survive, beta=BETA):
    trigger, p_fail, y_pred = maintenance_decision(p_survive, beta=beta)
    acc = accuracy_score(y_true, y_pred)
    auc = roc_auc_score(y_true, p_survive) if len(np.unique(y_true)) > 1 else np.nan

    print(f"\n=== {name} (maintenance rule, beta={beta}) ===")
    print("Triggered maintenance:", int(trigger.sum()), "of", len(trigger), f"
({trigger.mean():.1%})")
    print("Accuracy:", acc)
    print("ROC AUC :", auc, "(AUC independent of beta)")
    print("Confusion matrix (y=0 fail, y=1 survive):\n", confusion_matrix(y_true,
y_pred))
    print("\nReport:\n", classification_report(y_true, y_pred, digits=4))

    return trigger, p_fail, y_pred

trigger_val,  p_fail_val,  y_pred_val  = eval_maintenance("VAL",  y_val,  p_val_mean,
beta=BETA)
trigger_test, p_fail_test, y_pred_test = eval_maintenance("TEST", y_test, p_test_mean,
beta=BETA)
```

```
=== VAL (maintenance rule, beta=0.15) ===
Triggered maintenance: 54462 of 105912 (51.4%)
Accuracy: 0.6136320719087545
ROC AUC : 0.758414951888521 (AUC independent of beta)
Confusion matrix (y=0 fail, y=1 survive):
 [[17025  3484]
 [37437 47966]]

Report:
               precision    recall  f1-score   support

         0.0     0.3126    0.8301    0.4542     20509
         1.0     0.9323    0.5616    0.7010     85403

    accuracy                         0.6136    105912
   macro avg     0.6224    0.6959    0.5776    105912
weighted avg     0.8123    0.6136    0.6532    105912


=== TEST (maintenance rule, beta=0.15) ===
Triggered maintenance: 55546 of 105355 (52.7%)
Accuracy: 0.5955388923164538
ROC AUC : 0.7556135134157711 (AUC independent of beta)
Confusion matrix (y=0 fail, y=1 survive):
 [[16417  3483]
 [39129 46326]]

Report:
               precision    recall  f1-score   support

         0.0     0.2956    0.8250    0.4352     19900
         1.0     0.9301    0.5421    0.6850     85455

    accuracy                         0.5955    105355
   macro avg     0.6128    0.6835    0.5601    105355
weighted avg     0.8102    0.5955    0.6378    105355
```

# Inspect which features matter

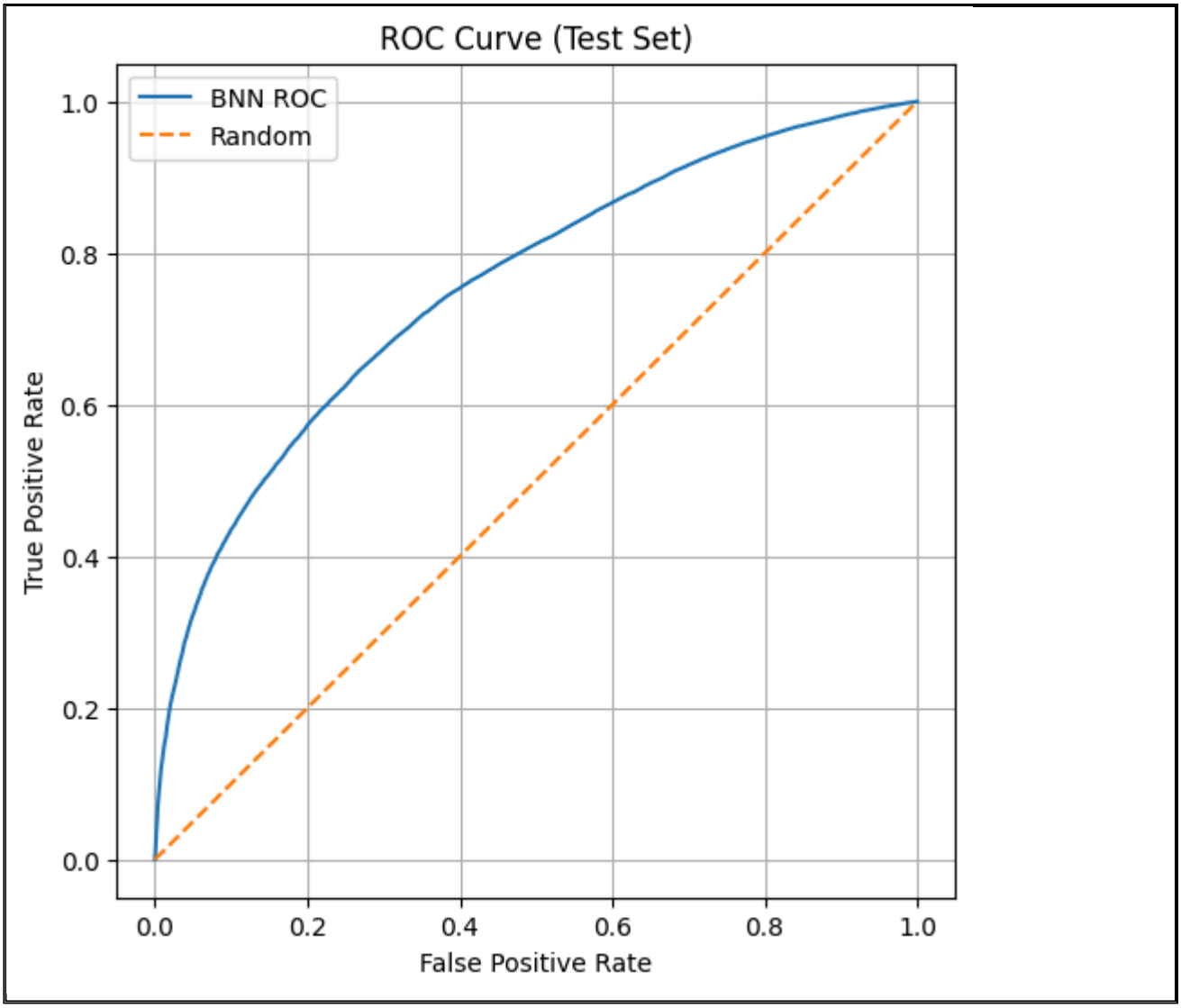
```
import numpy as np

print("\nFailure risk distribution (TEST):")
for q in [1, 5, 10, 25, 50, 75, 90, 95, 99]:
    print(f"{q:>2d}th percentile:", float(np.percentile(p_fail_test, q)))
print("Mean:", float(np.mean(p_fail_test)))
print("Median:", float(np.median(p_fail_test)))
```

```
Failure risk distribution (TEST):
 1th percentile: 0.0040000081062316895
 5th percentile: 0.012000024318695068
10th percentile: 0.022000014781951904
25th percentile: 0.054000020027160645
50th percentile: 0.16200000047683716
75th percentile: 0.28200000524520874
90th percentile: 0.4100000262260437
95th percentile: 0.5019999742507935
99th percentile: 0.6139999628067017
Mean: 0.1898721158504486
Median: 0.16200000047683716
```

# ROC curve

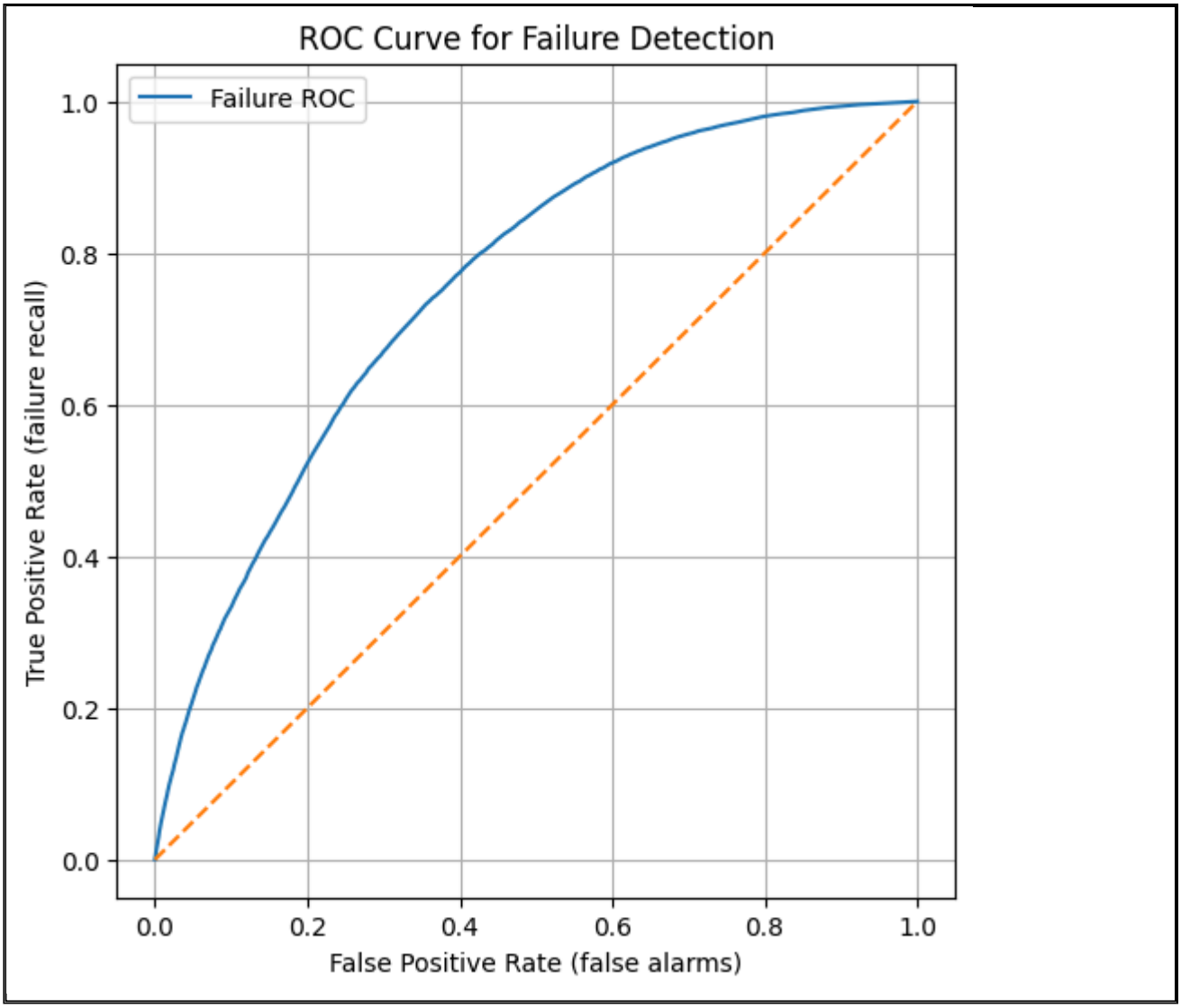
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, p_test_mean)
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, label="BNN ROC")
plt.plot([0,1], [0,1], linestyle="--", label="Random")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (Test Set)")
plt.legend()
plt.grid(True)
plt.show()
```




# ROC for failure

```
p_fail_test = 1 - p_test_mean  # failure probability

fpr_fail, tpr_fail, thresholds_fail = roc_curve(
    (y_test == 0).astype(int),  # positive class = failure
    p_fail_test
)

plt.figure(figsize=(6,6))
plt.plot(fpr_fail, tpr_fail, label="Failure ROC")
plt.plot([0,1], [0,1], linestyle="--")
plt.xlabel("False Positive Rate (false alarms)")
plt.ylabel("True Positive Rate (failure recall)")
plt.title("ROC Curve for Failure Detection")
plt.legend()
plt.grid(True)
plt.show()
```



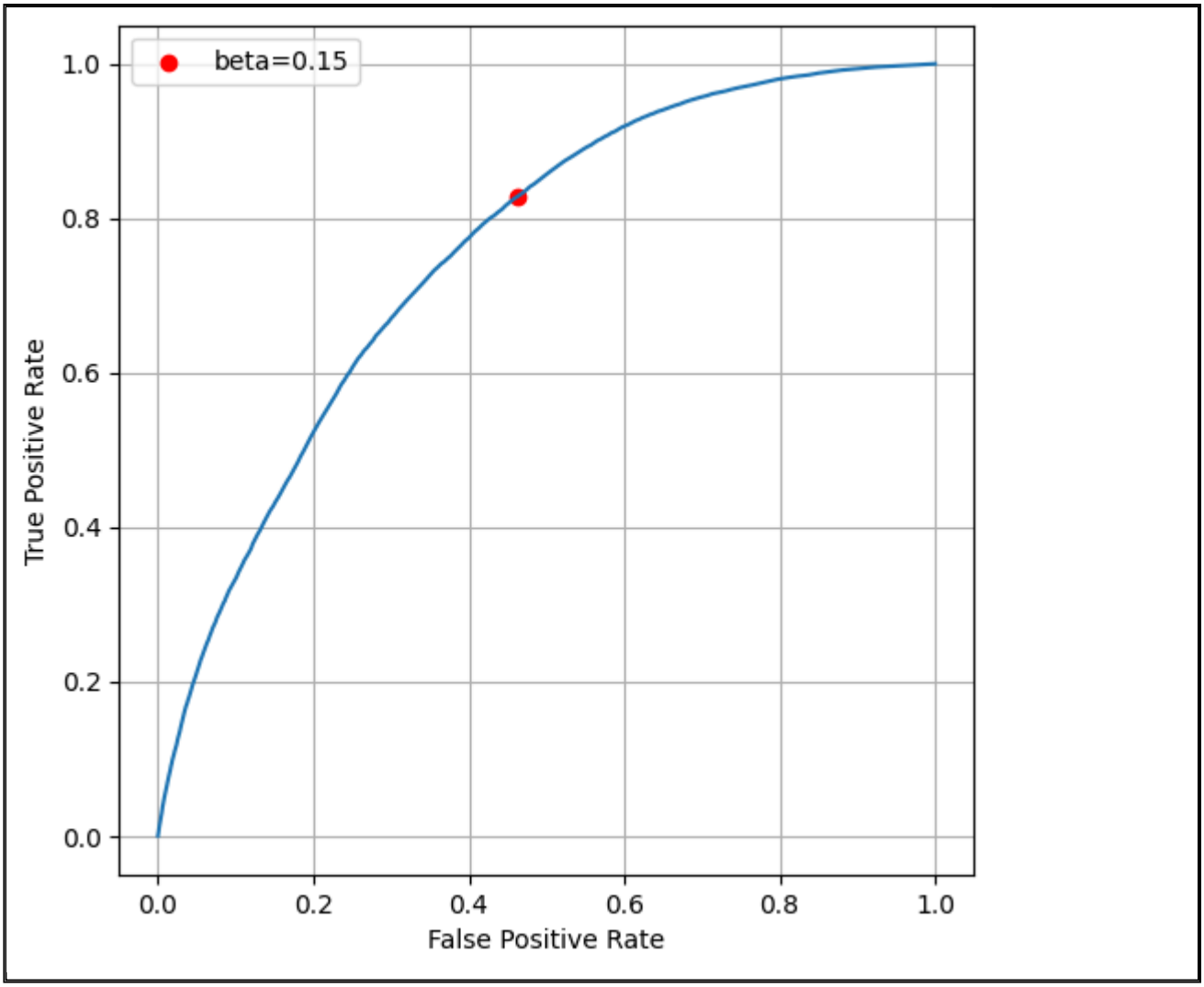

# β mapping

```
for i in range(0, len(thresholds_fail), len(thresholds_fail)//10):
    print(
        "Threshold (failure prob):", round(thresholds_fail[i], 3),
        " | Failure Recall:", round(tpr_fail[i], 3),
        " | False Alarm Rate:", round(fpr_fail[i], 3)
    )
```

```
Threshold (failure prob): inf  | Failure Recall: 0.0  | False Alarm Rate: 0.0
Threshold (failure prob): 0.666  | Failure Recall: 0.01  | False Alarm Rate: 0.001
Threshold (failure prob): 0.592  | Failure Recall: 0.049  | False Alarm Rate: 0.008
Threshold (failure prob): 0.518  | Failure Recall: 0.121  | False Alarm Rate: 0.025
Threshold (failure prob): 0.444  | Failure Recall: 0.211  | False Alarm Rate: 0.05
Threshold (failure prob): 0.37  | Failure Recall: 0.307  | False Alarm Rate: 0.087
Threshold (failure prob): 0.296  | Failure Recall: 0.465  | False Alarm Rate: 0.17
Threshold (failure prob): 0.222  | Failure Recall: 0.675  | False Alarm Rate: 0.304
Threshold (failure prob): 0.148  | Failure Recall: 0.832  | False Alarm Rate: 0.469
Threshold (failure prob): 0.074  | Failure Recall: 0.938  | False Alarm Rate: 0.643
Threshold (failure prob): 0.0  | Failure Recall: 1.0  | False Alarm Rate: 1.0
```

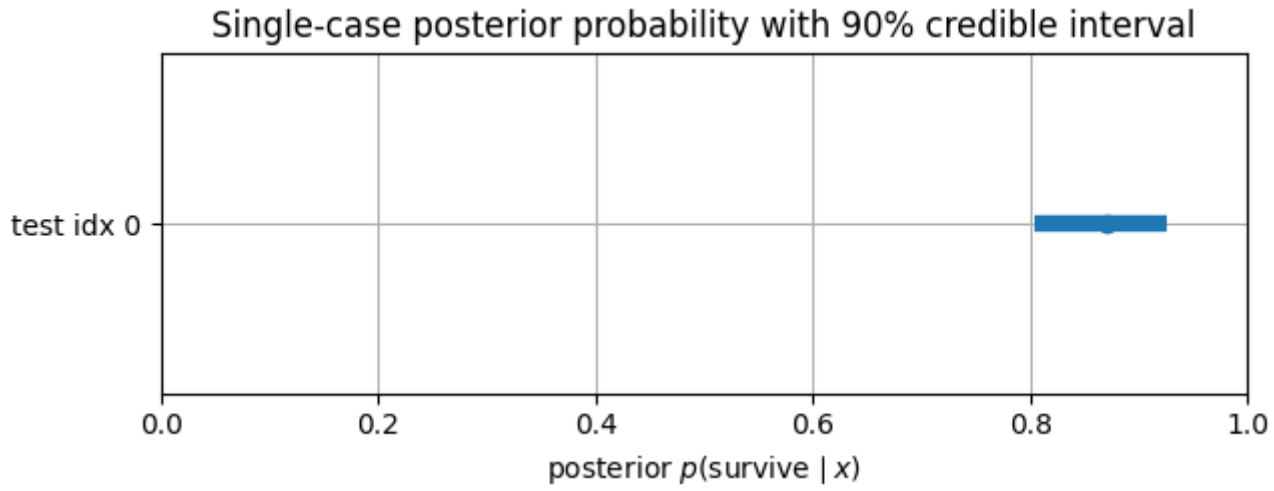
```
beta = 0.15
idx = np.argmin(np.abs(thresholds_fail - beta))
```

```
plt.figure(figsize=(6,6))
plt.plot(fpr_fail, tpr_fail)
plt.scatter(fpr_fail[idx], tpr_fail[idx], color="red", label=f"beta={beta}")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.show()
```




```
# Figure 1 — posterior p(survive) for ONE test sample (credible interval)
import numpy as np
import torch
import matplotlib.pyplot as plt
from pyro.infer import Predictive

@torch.no_grad()
def posterior_predict_p_survive_samples(model, guide, x_tensor, num_samples=1000):
    predictive = Predictive(model, guide=guide, num_samples=num_samples, return_sites=
("_RETURN",))
    logits_s = predictive(x_tensor, y=None)["_RETURN"]          # [S, N]
    p_s = torch.sigmoid(logits_s).cpu().numpy()                 # [S, N]
    return p_s

# choose one representative test index
i = 0  # change if desired
x_star = X_test_t[i:i+1]

p_s = posterior_predict_p_survive_samples(model, guide, x_star,
```

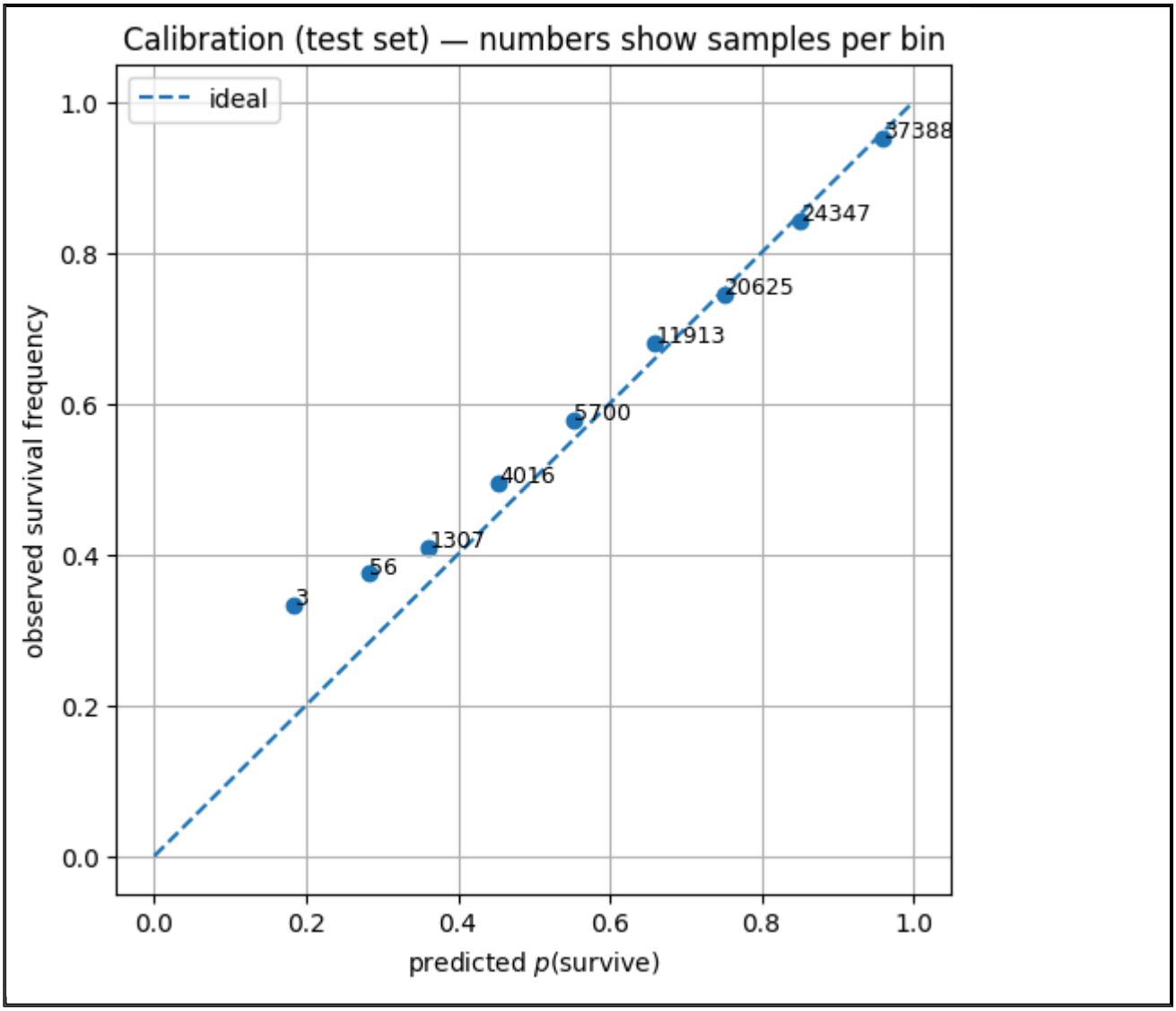
```
num_samples=2000).reshape(-1)

p_mean = float(np.mean(p_s))
p_lo   = float(np.quantile(p_s, 0.05))
p_hi   = float(np.quantile(p_s, 0.95))

plt.figure(figsize=(7, 2.2))
plt.hlines(1, p_lo, p_hi, linewidth=6)
plt.plot([p_mean], [1], marker="o")
plt.yticks([1], [f"test idx {i}"])
plt.xlim(0, 1)
plt.xlabel("posterior $p(\\mathrm{survive}\\mid x)$")
plt.title("Single-case posterior probability with 90% credible interval")
plt.grid(True)
plt.show()

print("Posterior summary:")
print("mean:", p_mean, "  5%:", p_lo, "  95%:", p_hi)
```




```
Posterior summary:
mean: 0.8711585402488708   5%: 0.8036723732948303   95%: 0.9256730079650879
```

```
# Figure 2 — calibration curve (test set)
import numpy as np
import matplotlib.pyplot as plt

p = np.asarray(p_test_mean).astype(float)
y = np.asarray(y_test).astype(int)

n_bins = 10
bins = np.linspace(0.0, 1.0, n_bins + 1)
bin_id = np.digitize(p, bins[1:-1], right=True)

p_bin = np.array([p[bin_id==k].mean() if np.any(bin_id==k) else np.nan for k in
range(n_bins)])
y_bin = np.array([y[bin_id==k].mean() if np.any(bin_id==k) else np.nan for k in
range(n_bins)])
n_bin = np.array([int(np.sum(bin_id==k)) for k in range(n_bins)])

plt.figure(figsize=(6,6))
plt.plot([0,1],[0,1], linestyle="--", label="ideal")
plt.scatter(p_bin, y_bin)
for k in range(n_bins):
    if not np.isnan(p_bin[k]):
        plt.text(p_bin[k], y_bin[k], str(n_bin[k]), fontsize=9)
```

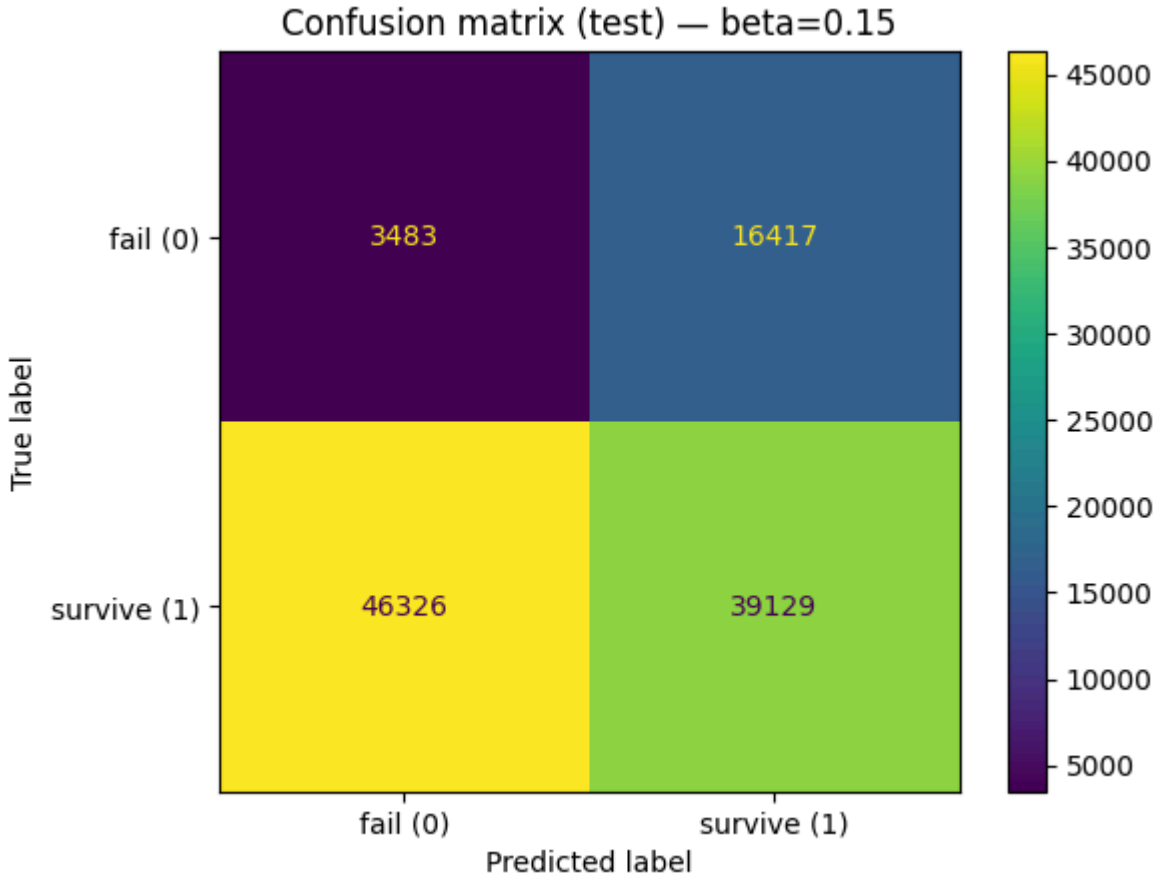
```
plt.xlabel("predicted $p(\\mathrm{survive})$")
plt.ylabel("observed survival frequency")
plt.title("Calibration (test set) — numbers show samples per bin")
plt.grid(True)
plt.legend()
plt.show()
```




```
# Figure 3 — confusion matrix under beta decision rule (test set)
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
classification_report

beta = BETA  # uses your notebook constant
y = np.asarray(y_test).astype(int)
p_survive = np.asarray(p_test_mean).astype(float)

y_pred, trigger, score = maintenance_decision(p_survive, beta=beta)

cm = confusion_matrix(y, y_pred, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["fail (0)", "survive
(1)"])

plt.figure(figsize=(5.5,5.5))
disp.plot(values_format="d")
```

```
plt.title(f"Confusion matrix (test) — beta={beta}")
plt.grid(False)
plt.show()

print(classification_report(y, y_pred, target_names=["fail (0)", "survive (1)"]))
print("Triggered maintenance count:", int(np.sum(trigger)))
```

```
<Figure size 550x550 with 0 Axes>
```

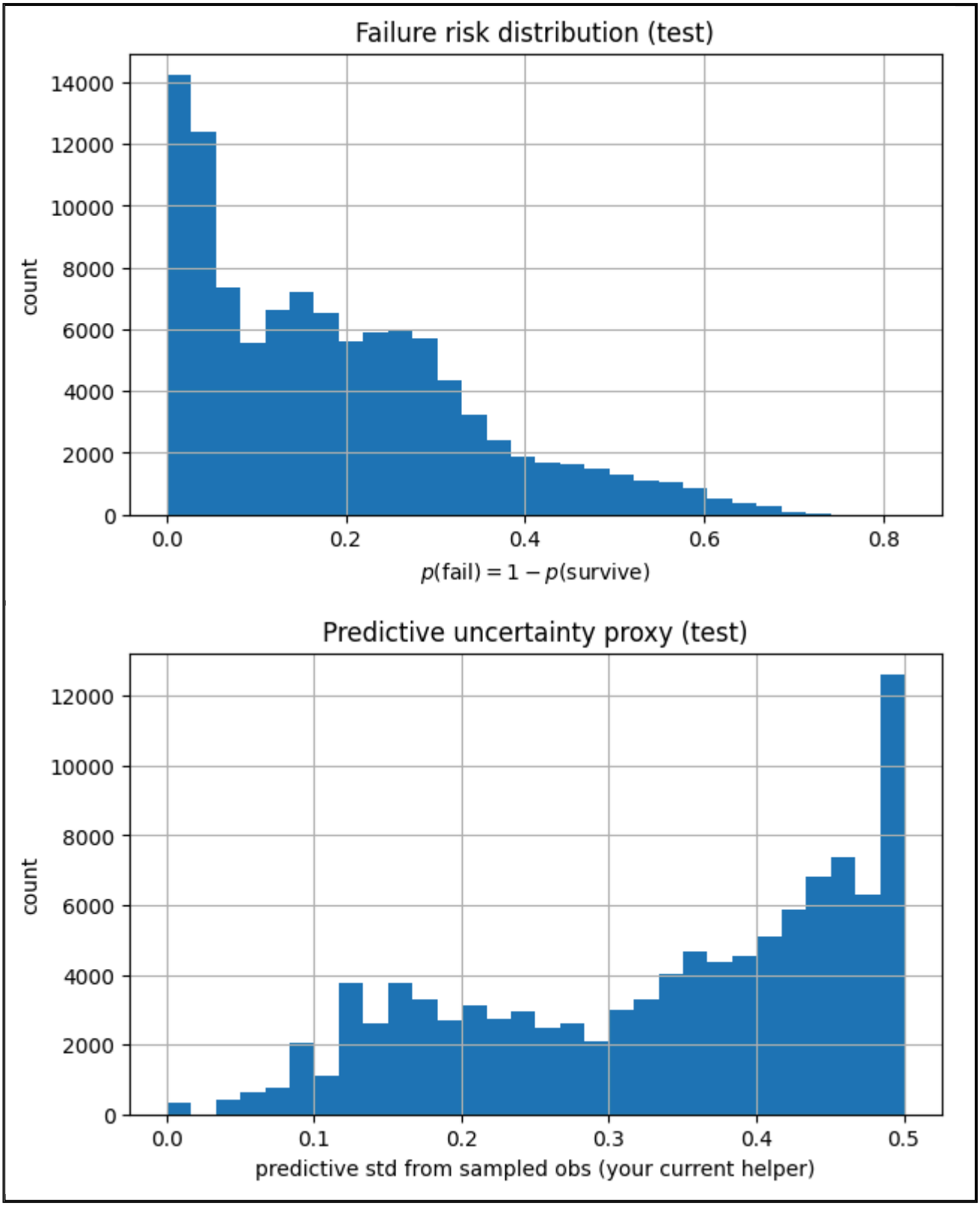



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| fail (0) | 0.07 | 0.18 | 0.10 | 19900 |
| survive (1) | 0.70 | 0.46 | 0.56 | 85455 |
| accuracy | | | 0.40 | 105355 |
| macro avg | 0.39 | 0.32 | 0.33 | 105355 |
| weighted avg | 0.58 | 0.40 | 0.47 | 105355 |

```
Triggered maintenance count: 20003
```

```
# Figure 4 — distributions: failure risk and predictive uncertainty (test set)
import numpy as np
import matplotlib.pyplot as plt

p_fail = 1.0 - np.asarray(p_test_mean).astype(float)
p_unc  = np.asarray(p_test_std).astype(float)

plt.figure(figsize=(7,4))
plt.hist(p_fail, bins=30)
plt.xlabel("$p(\\mathrm{fail}) = 1 - p(\\mathrm{survive})$")
plt.ylabel("count")
plt.title("Failure risk distribution (test)")
plt.grid(True)
plt.show()
```

```
plt.figure(figsize=(7,4))
plt.hist(p_unc, bins=30)
plt.xlabel("predictive std from sampled obs (your current helper)")
plt.ylabel("count")
plt.title("Predictive uncertainty proxy (test)")
plt.grid(True)
plt.show()
```




```
# Table 1 — test metrics summary (single row)
import numpy as np
import pandas as pd
```

```
from sklearn.metrics import accuracy_score, roc_auc_score, precision_score, recall_score,
f1_score

beta = BETA
y = np.asarray(y_test).astype(int)
p_survive = np.asarray(p_test_mean).astype(float)

y_pred, trigger, score = maintenance_decision(p_survive, beta=beta)

row = {
    "Split": "Test",
    "N": int(len(y)),
    "beta": float(beta),
    "Accuracy": float(accuracy_score(y, y_pred)),
    "AUC_survive": float(roc_auc_score(y, p_survive)) if len(np.unique(y)) > 1 else
np.nan,
    "Precision_survive": float(precision_score(y, y_pred, zero_division=0)),
    "Recall_survive": float(recall_score(y, y_pred, zero_division=0)),
    "F1_survive": float(f1_score(y, y_pred, zero_division=0)),
    "Triggered_maint_count": int(np.sum(trigger)),
    "Triggered_rate": float(np.mean(trigger)),
    "Mean_p_fail": float(np.mean(1.0 - p_survive)),
}
pd.DataFrame([row])
```

| | Split | N | beta | Accuracy | AUC_survive | Precision_survive | Recall_survive | F1_survive | Trigge |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Test | 105355 | 0.15 | 0.404461 | 0.755614 | 0.704443 | 0.45789 | 0.555017 | |

```
# Table 2 — metrics by Run (test set)
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, roc_auc_score

beta = BETA
tmp = df_test.copy()
tmp["p_survive"] = np.asarray(p_test_mean).astype(float)
tmp["p_fail"] = 1.0 - tmp["p_survive"]

y_pred, trigger, score = maintenance_decision(p_survive, beta=beta)

tmp["y_pred"] = y_pred
tmp["trigger"] = trigger

rows = []
for run, g in tmp.groupby("Run"):
    y = g["y"].astype(int).values
    p = g["p_survive"].values
    yp = g["y_pred"].astype(int).values
    rows.append({
        "Run": run,
        "N": int(len(g)),
        "Acc": float(accuracy_score(y, yp)),
        "AUC_survive": float(roc_auc_score(y, p)) if len(np.unique(y)) > 1 else np.nan,
        "Triggered_rate": float(g["trigger"].mean()),
        "Mean_p_fail": float(g["p_fail"].mean()),
    })
```

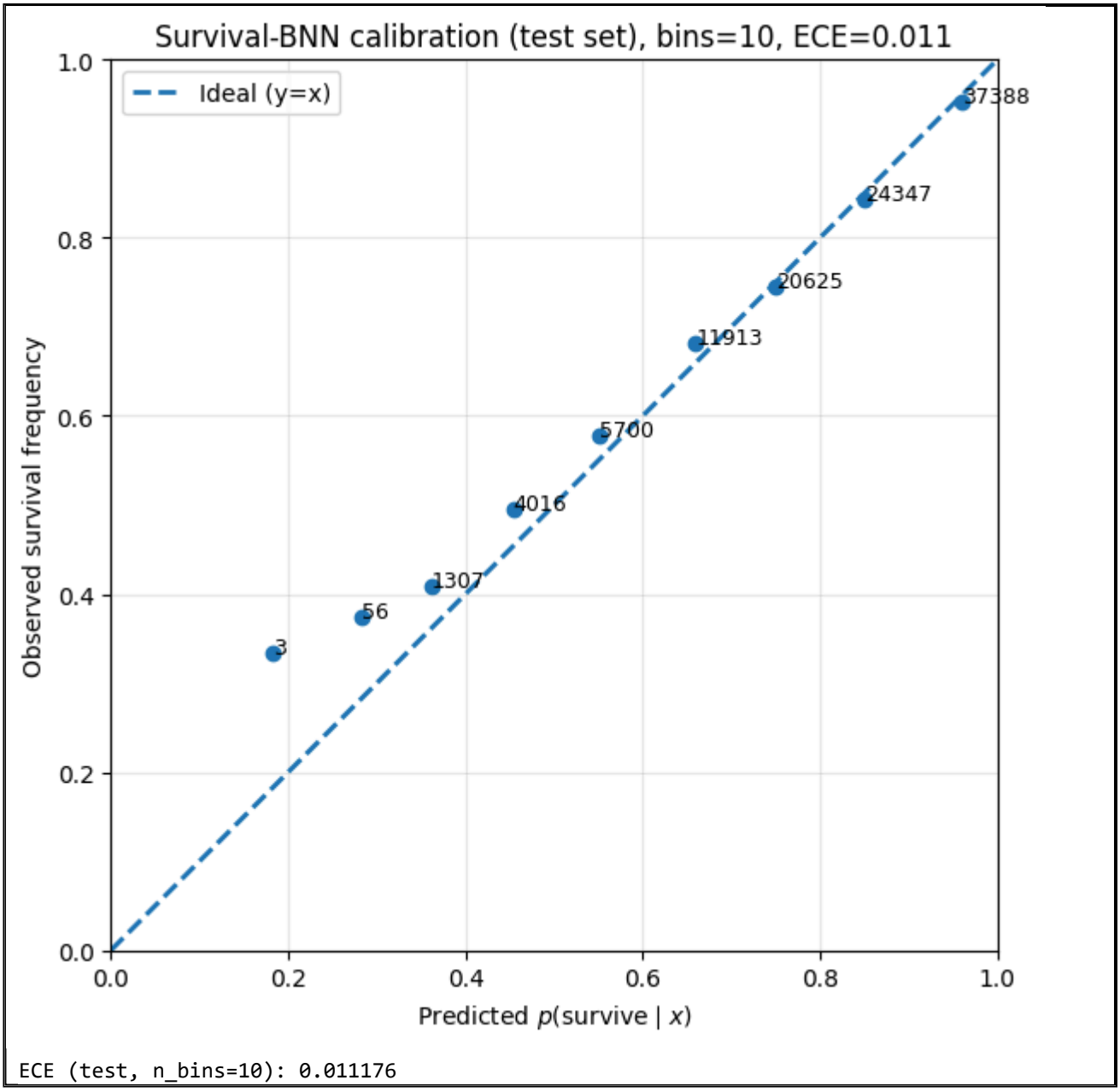
```
pd.DataFrame(rows).sort_values("Run").reset_index(drop=True)
```

| | Run | N | Acc | AUC_survive | Triggered_rate | Mean_p_fail |
|---|---|---|---|---|---|---|
| **0** | 5 | 1925 | 0.421818 | 0.706382 | 0.304296 | 0.304296 |
| **1** | 10 | 1930 | 0.417617 | 0.714826 | 0.306221 | 0.306221 |
| **2** | 15 | 1930 | 0.423834 | 0.708019 | 0.311719 | 0.311719 |
| **3** | 20 | 1931 | 0.424650 | 0.697458 | 0.312485 | 0.312485 |
| **4** | 25 | 1931 | 0.426722 | 0.702645 | 0.313816 | 0.313816 |
| **5** | 30 | 2000 | 0.435000 | 0.678118 | 0.207291 | 0.207291 |
| **6** | 35 | 2014 | 0.457299 | 0.661725 | 0.220692 | 0.220692 |
| **7** | 40 | 2016 | 0.458333 | 0.672108 | 0.222324 | 0.222324 |
| **8** | 45 | 2017 | 0.469013 | 0.661809 | 0.229453 | 0.229453 |
| **9** | 50 | 2018 | 0.463330 | 0.676546 | 0.230839 | 0.230839 |
| **10** | 55 | 2018 | 0.572349 | 0.603579 | 0.223360 | 0.223360 |
| **11** | 60 | 2026 | 0.580454 | 0.638263 | 0.217427 | 0.217427 |
| **12** | 65 | 2026 | 0.591313 | 0.636814 | 0.219941 | 0.219941 |
| **13** | 70 | 2026 | 0.584896 | 0.629105 | 0.218879 | 0.218879 |
| **14** | 75 | 2026 | 0.588351 | 0.632870 | 0.217519 | 0.217519 |
| **15** | 80 | 2041 | 0.394904 | 0.659302 | 0.127253 | 0.127253 |
| **16** | 85 | 2167 | 0.399631 | 0.558204 | 0.128920 | 0.128920 |
| **17** | 90 | 2172 | 0.417587 | 0.609669 | 0.136355 | 0.136355 |
| **18** | 95 | 2172 | 0.434162 | 0.601150 | 0.138461 | 0.138461 |
| **19** | 100 | 2172 | 0.447974 | 0.600033 | 0.139992 | 0.139992 |
| **20** | 105 | 1811 | 0.067366 | 0.572271 | 0.048447 | 0.048447 |
| **21** | 110 | 1811 | 0.061292 | 0.571271 | 0.047228 | 0.047228 |
| **22** | 115 | 1811 | 0.065157 | 0.544245 | 0.047111 | 0.047111 |
| **23** | 120 | 1811 | 0.062396 | 0.560596 | 0.046317 | 0.046317 |
| **24** | 125 | 1811 | 0.059636 | 0.574244 | 0.046702 | 0.046702 |
| **25** | 130 | 1914 | 0.019854 | 0.627248 | 0.019809 | 0.019809 |
| **26** | 135 | 1914 | 0.019331 | 0.659693 | 0.018405 | 0.018405 |
| **27** | 140 | 1914 | 0.018809 | 0.624749 | 0.018297 | 0.018297 |
| **28** | 145 | 1914 | 0.018809 | 0.631878 | 0.018328 | 0.018328 |
| **29** | 150 | 1914 | 0.018809 | 0.709457 | 0.018732 | 0.018732 |
| **30** | 155 | 2041 | 0.492896 | 0.718736 | 0.354843 | 0.354843 |
| **31** | 160 | 2062 | 0.519884 | 0.719249 | 0.383130 | 0.383130 |
| **32** | 165 | 2063 | 0.517693 | 0.719436 | 0.382314 | 0.382314 |

| | Run | N | Acc | AUC_survive | Triggered_rate | Mean_p_fail |
|---|---|---|---|---|---|---|
| **33** | 170 | 2063 | 0.520116 | 0.718100 | 0.383695 | 0.383695 |
| **34** | 175 | 2063 | 0.518662 | 0.722323 | 0.383638 | 0.383638 |
| **35** | 180 | 2041 | 0.543851 | 0.657491 | 0.211748 | 0.211748 |
| **36** | 185 | 2170 | 0.543779 | 0.646698 | 0.212446 | 0.212446 |
| **37** | 190 | 2204 | 0.587114 | 0.639146 | 0.231963 | 0.231963 |
| **38** | 195 | 2207 | 0.590847 | 0.644433 | 0.230942 | 0.230942 |
| **39** | 200 | 2207 | 0.593113 | 0.626055 | 0.233568 | 0.233568 |
| **40** | 205 | 1975 | 0.508354 | 0.646499 | 0.218407 | 0.218407 |
| **41** | 210 | 1967 | 0.512456 | 0.622814 | 0.217188 | 0.217188 |
| **42** | 215 | 1988 | 0.518612 | 0.650333 | 0.221963 | 0.221963 |
| **43** | 220 | 2042 | 0.598923 | 0.608989 | 0.216426 | 0.216426 |
| **44** | 225 | 2057 | 0.604278 | 0.594501 | 0.214280 | 0.214280 |
| **45** | 230 | 2064 | 0.614826 | 0.619715 | 0.215396 | 0.215396 |
| **46** | 235 | 2138 | 0.388681 | 0.630041 | 0.132909 | 0.132909 |
| **47** | 240 | 2154 | 0.374652 | 0.644335 | 0.128067 | 0.128067 |
| **48** | 245 | 2156 | 0.410946 | 0.600122 | 0.133071 | 0.133071 |
| **49** | 250 | 2170 | 0.326267 | 0.601908 | 0.113926 | 0.113926 |
| **50** | 255 | 2170 | 0.329032 | 0.575974 | 0.115563 | 0.115563 |
| **51** | 260 | 2170 | 0.262673 | 0.617638 | 0.103920 | 0.103920 |

```
# --- Survival-BNN calibration (binary): reliability diagram + optional ECE ---
# Requires:
#   p_test_mean : (N,) posterior mean p(survive) on TEST
#   y_test      : (N,) true labels in {0,1} where 1=survive, 0=fail

import numpy as np
import matplotlib.pyplot as plt

# --- inputs ---
p = np.asarray(p_test_mean, dtype=float).reshape(-1)
y = np.asarray(y_test, dtype=int).reshape(-1)
assert p.shape[0] == y.shape[0], "p_test_mean and y_test must have the same length"
assert np.all((p >= 0) & (p <= 1)), "Probabilities must be in [0,1]"
assert set(np.unique(y)).issubset({0, 1}), "y_test must be binary {0,1}"

# --- binning ---
n_bins = 10
edges = np.linspace(0.0, 1.0, n_bins + 1)
bin_id = np.digitize(p, edges[1:-1], right=True)  # 0..n_bins-1

p_bin = np.full(n_bins, np.nan, dtype=float)   # mean predicted prob per bin
y_bin = np.full(n_bins, np.nan, dtype=float)   # observed frequency per bin
n_bin = np.zeros(n_bins, dtype=int)            # sample count per bin
```

```
for k in range(n_bins):
    mask = (bin_id == k)
    n_bin[k] = int(mask.sum())
    if n_bin[k] > 0:
        p_bin[k] = float(p[mask].mean())
        y_bin[k] = float(y[mask].mean())

# --- optional: Expected Calibration Error (ECE) ---
ece = 0.0
N = len(y)
for k in range(n_bins):
    if n_bin[k] > 0 and np.isfinite(p_bin[k]) and np.isfinite(y_bin[k]):
        ece += (n_bin[k] / N) * abs(y_bin[k] - p_bin[k])

# --- plot ---
plt.figure(figsize=(6.2, 6.2))
plt.plot([0, 1], [0, 1], linestyle="--", linewidth=2, label="Ideal (y=x)")

mask = np.isfinite(p_bin) & np.isfinite(y_bin)
plt.scatter(p_bin[mask], y_bin[mask])

# annotate bins with sample count (as in your script)
for k in range(n_bins):
    if np.isfinite(p_bin[k]):
        plt.text(p_bin[k], y_bin[k], str(n_bin[k]), fontsize=9)

plt.xlabel(r"Predicted $p(\mathrm{survive}\mid x)$")
plt.ylabel("Observed survival frequency")
plt.title(f"Survival-BNN calibration (test set), bins={n_bins}, ECE={ece:.3f}")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

print(f"ECE (test, n_bins={n_bins}): {ece:.6f}")

# Optional: save for thesis inclusion (vector preferred)
# plt.savefig("survival_bnn_calibration_test.pdf", bbox_inches="tight")  # PDF vector
# plt.savefig("survival_bnn_calibration_test.png", bbox_inches="tight", dpi=300)  # high-
res raster
```




```
ECE (test, n_bins=10): 0.011176
```

# D Policy-BNN Notebook

# Load tables and join

```
import sqlite3
import pandas as pd

DB_PATH = "normalized_Exp_DB_892.db"

def read_table(name):
    with sqlite3.connect(DB_PATH) as con:
        return pd.read_sql_query(f"SELECT * FROM {name};", con)

df_train = read_table("BNN_Train")
df_val   = read_table("BNN_Val")
df_test  = read_table("BNN_Test")

df_run = read_table("Run")          # RunID, ExperimentID, RunNo
df_exp = read_table("Experiment")   # ExperimentID, ExpNr, ...

print(df_train.shape, df_val.shape, df_test.shape)
print(df_run.columns)
print(df_exp.columns)
```

```
(319442, 18) (105912, 18) (105355, 18)
Index(['RunID', 'ExperimentID', 'RunNo'], dtype='str')
Index(['ExperimentID', 'ExpNr', 'Availability', 'Distribution - Duration',
       'MTTR', 'Sigma', 'Distribution - Interval', 'Mu', 'Beta', 'Eta',
       'Interval Time', 'Repair Time', 'Brooker active', 'Nominal CT',
       'Customer CT', 'Max CT', 'Rho'],
      dtype='str')
```

# Add ExpNr into each BNN table

```
def attach_experiment_label(df_bnn, df_run, df_exp):
    df = df_bnn.merge(df_run[["RunID", "ExperimentID"]], left_on="Run", right_on="RunID",
how="left")
    df = df.merge(df_exp[["ExperimentID", "ExpNr"]], on="ExperimentID", how="left")

    miss = df["ExpNr"].isna().mean()
    print(f"Missing ExpNr after join: {miss:.3%}")

    return df

df_train = attach_experiment_label(df_train, df_run, df_exp)
df_val   = attach_experiment_label(df_val,   df_run, df_exp)
df_test  = attach_experiment_label(df_test,  df_run, df_exp)

df_train[["Run","RunID","ExperimentID","ExpNr"]].head()
```

```
Missing ExpNr after join: 0.000%
Missing ExpNr after join: 0.000%
Missing ExpNr after join: 0.000%
```

| | Run | RunID | ExperimentID | ExpNr |
|---|---|---|---|---|
| **0** | 1 | 1 | 1 | Exp_A_79_DC |
| **1** | 1 | 1 | 1 | Exp_A_79_DC |
| **2** | 1 | 1 | 1 | Exp_A_79_DC |
| **3** | 1 | 1 | 1 | Exp_A_79_DC |
| **4** | 1 | 1 | 1 | Exp_A_79_DC |

# Map ExpNr to policy class (C2 / B / C1)

```
import numpy as np

def expnr_to_policy_class(expnr: str):
    s = str(expnr).upper()
    if "_C2_" in s or "C2" in s:
        return 0
    if "_B_"  in s or "EXP_B_" in s:
        return 1
    if "_C1_" in s or "C1" in s:
        return 2
    return np.nan

for df in (df_train, df_val, df_test):
    df["policy_class"] = df["ExpNr"].apply(expnr_to_policy_class)

print("Unmapped rows (train/val/test):",
      df_train["policy_class"].isna().sum(),
      df_val["policy_class"].isna().sum(),
      df_test["policy_class"].isna().sum())

df_train = df_train.dropna(subset=["policy_class"]).copy()
df_val   = df_val.dropna(subset=["policy_class"]).copy()
df_test  = df_test.dropna(subset=["policy_class"]).copy()

df_train["policy_class"] = df_train["policy_class"].astype(int)
df_val["policy_class"]   = df_val["policy_class"].astype(int)
df_test["policy_class"]  = df_test["policy_class"].astype(int)

print(df_train["policy_class"].value_counts())
```

```
Unmapped rows (train/val/test): 137022 45325 44763
policy_class
0    63697
1    62593
2    56130
Name: count, dtype: int64
```

# Build features

```
TARGET_COL = "policy_class"
DROP_COLS = {"y", "Run", "RunID", "ExperimentID", "ExpNr", TARGET_COL}
```

```
def prepare_features(df):
    df = df.copy()

    # Ensure numeric columns are numeric
    if "TimeSinceMaintEnd" in df.columns:
        df["TimeSinceMaintEnd"] = pd.to_numeric(df["TimeSinceMaintEnd"], errors="coerce")

    feature_cols = []
    for c in df.columns:
        if c in DROP_COLS:
            continue
        if pd.api.types.is_numeric_dtype(df[c]):
            feature_cols.append(c)

    for c in feature_cols:
        if df[c].isna().any():
            df[c] = df[c].fillna(df[c].median())

    return df, feature_cols

df_train, FEATURE_COLS = prepare_features(df_train)
df_val, _              = prepare_features(df_val)
df_test, _             = prepare_features(df_test)

print("Features:", FEATURE_COLS)
print("Train class balance:\n", df_train[TARGET_COL].value_counts(normalize=True))
```

```
Features: ['In_EndTimeStamp', 'Dt_ToNextMaint', 'DeltaT', 'CT_Median_In', 'CT_IQR_In',
'CT_P95_In', 'CT_Median_Out', 'CT_IQR_Out', 'CT_P95_Out', 'DowntimeWin', 'FailCountWin',
'MTTRWin', 'TimeSinceFailEnd', 'HasFailureHistory', 'TimeSinceMaintEnd', 'BufferLevel']
Train class balance:
 policy_class
0    0.349178
1    0.343126
2    0.307697
Name: proportion, dtype: float64
```

# Scale + tensors

```
from sklearn.preprocessing import StandardScaler
import torch

scaler = StandardScaler()
X_train = scaler.fit_transform(df_train[FEATURE_COLS].values.astype(np.float32))
X_val   = scaler.transform(df_val[FEATURE_COLS].values.astype(np.float32))
X_test  = scaler.transform(df_test[FEATURE_COLS].values.astype(np.float32))

y_train = df_train[TARGET_COL].values.astype(np.int64)
y_val   = df_val[TARGET_COL].values.astype(np.int64)
y_test  = df_test[TARGET_COL].values.astype(np.int64)

X_train_t = torch.tensor(X_train, dtype=torch.float32)
X_val_t   = torch.tensor(X_val, dtype=torch.float32)
X_test_t  = torch.tensor(X_test, dtype=torch.float32)

y_train_t = torch.tensor(y_train, dtype=torch.long)
```

```
y_val_t   = torch.tensor(y_val, dtype=torch.long)
y_test_t  = torch.tensor(y_test, dtype=torch.long)
```

# 3-class Bayesian model

```
import pyro
import pyro.distributions as dist
from pyro.nn import PyroModule, PyroSample
import torch.nn as nn

pyro.clear_param_store()
pyro.set_rng_seed(42)
torch.manual_seed(42)

class BayesianPolicyClassifier(PyroModule):
    def __init__(self, in_dim, hidden_dim=32, num_classes=3):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(in_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
        )
        self.out = PyroModule[nn.Linear](hidden_dim, num_classes)
        self.out.weight = PyroSample(dist.Normal(0., 1.).expand([num_classes,
hidden_dim]).to_event(2))
        self.out.bias   = PyroSample(dist.Normal(0.,
1.).expand([num_classes]).to_event(1))

    def forward(self, x, y=None):
        z = self.encoder(x)
        logits = self.out(z)  # [N,3]
        with pyro.plate("data", x.shape[0]):
            pyro.sample("obs", dist.Categorical(logits=logits), obs=y)
        return logits
```

# Train with SVI

```
from pyro.infer import SVI, Trace_ELBO
from pyro.infer.autoguide import AutoDiagonalNormal
import pyro
import numpy as np

model = BayesianPolicyClassifier(in_dim=X_train_t.shape[1], hidden_dim=32, num_classes=3)
guide = AutoDiagonalNormal(model)

optimizer = pyro.optim.Adam({"lr": 1e-3})
svi = SVI(model, guide, optimizer, loss=Trace_ELBO())

def train_svi(num_steps=3000, print_every=200):
    """
    Trains the Pyro SVI model and returns a list of per-step ELBO losses.
    Also prints a moving average over the last `print_every` steps.
    """
    loss_history = []
```

```
    for step in range(1, num_steps + 1):
        loss = svi.step(X_train_t, y_train_t)  # uses your existing training tensors
        loss_history.append(float(loss))

        if (step % print_every) == 0:
            window = loss_history[-print_every:]  # last print_every losses
            print(f"step {step:5d} | loss {float(np.mean(window)):.2f}")

    return loss_history

loss_history = train_svi(num_steps=3000, print_every=200)
```

```
step   200 | loss 112782.59
step   400 | loss 41977.63
step   600 | loss 32256.24
step   800 | loss 26300.84
step  1000 | loss 22435.85
step  1200 | loss 19581.21
step  1400 | loss 17542.74
step  1600 | loss 15565.30
step  1800 | loss 14122.37
step  2000 | loss 12478.33
step  2200 | loss 11249.76
step  2400 | loss 10266.74
step  2600 | loss 9288.33
step  2800 | loss 8529.55
step  3000 | loss 7699.46
```

# Posterior predictive class probabilities + evaluation

```
from pyro.infer import Predictive
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

@torch.no_grad()
def posterior_class_probs(model, guide, x_tensor, num_samples=300):
    pred = Predictive(model, guide=guide, num_samples=num_samples, return_sites=("obs",))
    y_s = pred(x_tensor, y=None)["obs"].cpu().numpy()  # [S,N]
    N = y_s.shape[1]
    K = 3
    probs = np.zeros((N, K), dtype=np.float32)
    for k in range(K):
        probs[:, k] = (y_s == k).mean(axis=0)
    return probs

def eval_multiclass(name, y_true, probs):
    y_pred = probs.argmax(axis=1)
    print(f"\n=== {name} ===")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Confusion matrix:\n", confusion_matrix(y_true, y_pred))
    print("\nReport:\n", classification_report(y_true, y_pred, digits=4))
    return y_pred

probs_val  = posterior_class_probs(model, guide, X_val_t,  num_samples=300)
probs_test = posterior_class_probs(model, guide, X_test_t, num_samples=300)
```

```
_ = eval_multiclass("VAL",  y_val,  probs_val)
_ = eval_multiclass("TEST", y_test, probs_test)
```

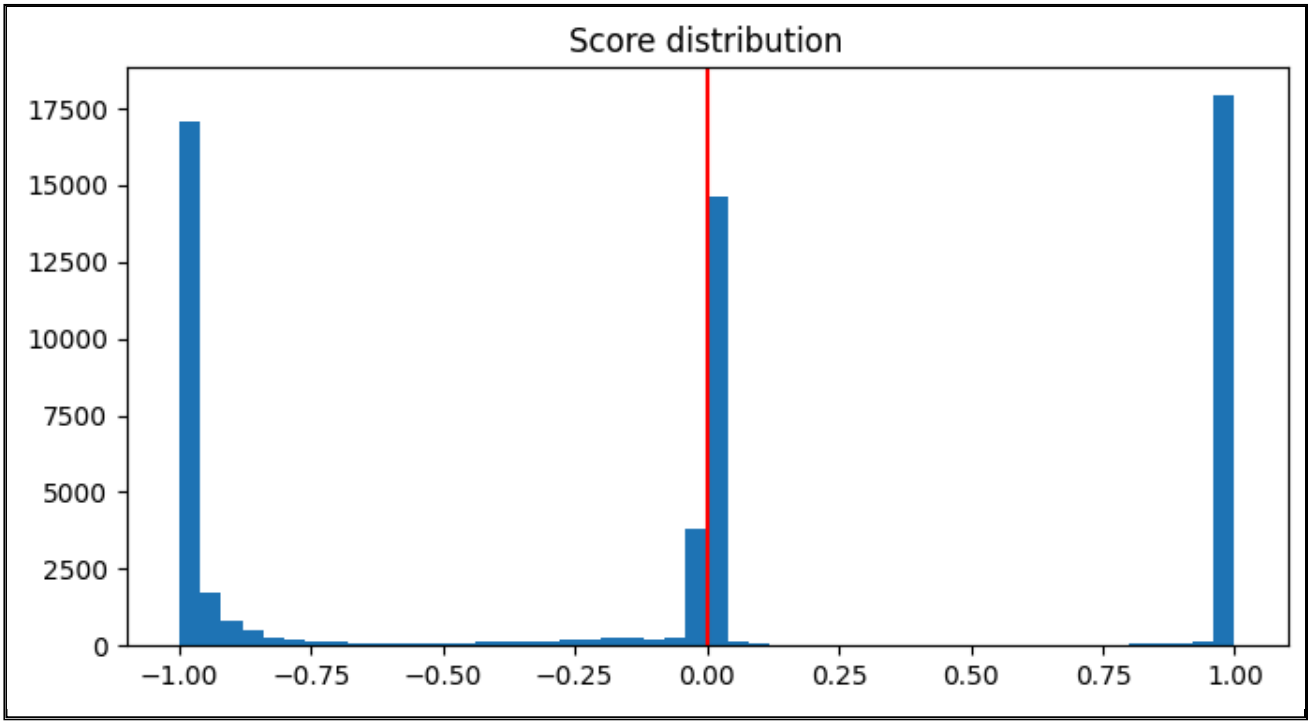
```
=== VAL ===
Accuracy: 0.9901959166157757
Confusion matrix:
 [[20772   223    27]
 [  221 20501    39]
 [    6    78 18720]]

Report:
               precision    recall  f1-score   support

           0     0.9892    0.9881    0.9886     21022
           1     0.9855    0.9875    0.9865     20761
           2     0.9965    0.9955    0.9960     18804

    accuracy                         0.9902     60587
   macro avg     0.9904    0.9904    0.9904     60587
weighted avg     0.9902    0.9902    0.9902     60587


=== TEST ===
Accuracy: 0.9853941114338527
Confusion matrix:
 [[20838   267    16]
 [  439 20359    48]
 [    6   109 18510]]

Report:
               precision    recall  f1-score   support

           0     0.9791    0.9866    0.9828     21121
           1     0.9819    0.9766    0.9792     20846
           2     0.9966    0.9938    0.9952     18625

    accuracy                         0.9854     60592
   macro avg     0.9858    0.9857    0.9858     60592
weighted avg     0.9854    0.9854    0.9854     60592
```

```
score = probs_test[:,2] - probs_test[:,0]
```

```
df_out = df_test.copy()
df_out["score"] = score

df_out.groupby("ExpNr")["score"].mean().sort_values()
```

```
ExpNr
Exp_C2_89_DC   -0.991634
Exp_C2_94_DC   -0.991110
Exp_C2_99_DC   -0.990646
Exp_C2_84_DC   -0.987958
Exp_C2_99_CC   -0.973320
Exp_C2_94_CC   -0.965537
Exp_C2_79_DC   -0.962099
Exp_C2_89_CC   -0.957199
Exp_C2_79_CC   -0.917389
Exp_C2_84_CC   -0.901935
Exp_B_79_DC    -0.181627
Exp_B_84_DC    -0.043203
Exp_B_79_CC    -0.028439
Exp_B_89_DC    -0.024218
Exp_B_94_DC    -0.019039
Exp_B_99_DC    -0.014335
Exp_B_84_CC    -0.008694
Exp_B_99_CC    -0.000094
Exp_B_89_CC     0.000522
Exp_B_94_CC     0.000525
Exp_C1_79_DC    0.973527
Exp_C1_79_CC    0.976088
Exp_C1_84_DC    0.985820
Exp_C1_84_CC    0.990230
Exp_C1_89_DC    0.990503
Exp_C1_94_CC    0.990927
Exp_C1_94_DC    0.991419
Exp_C1_89_CC    0.991977
Exp_C1_99_DC    0.992700
Exp_C1_99_CC    0.992983
Name: score, dtype: float32
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))
plt.hist(score, bins=50)
plt.axvline(0, color="red")
plt.title("Score distribution")
plt.show()
```




```
df_plot = df_test.copy()
df_plot["score"] = score
df_plot = df_plot.merge(df_exp[["ExperimentID","Interval Time"]], on="ExperimentID")

df_plot.groupby("Interval Time")["score"].mean().sort_index()
```

```
Interval Time
2400.0     0.988441
3600.0     0.986794
4800.0    -0.006976
7200.0    -0.514966
10800.0   -0.984737
Name: score, dtype: float32
```

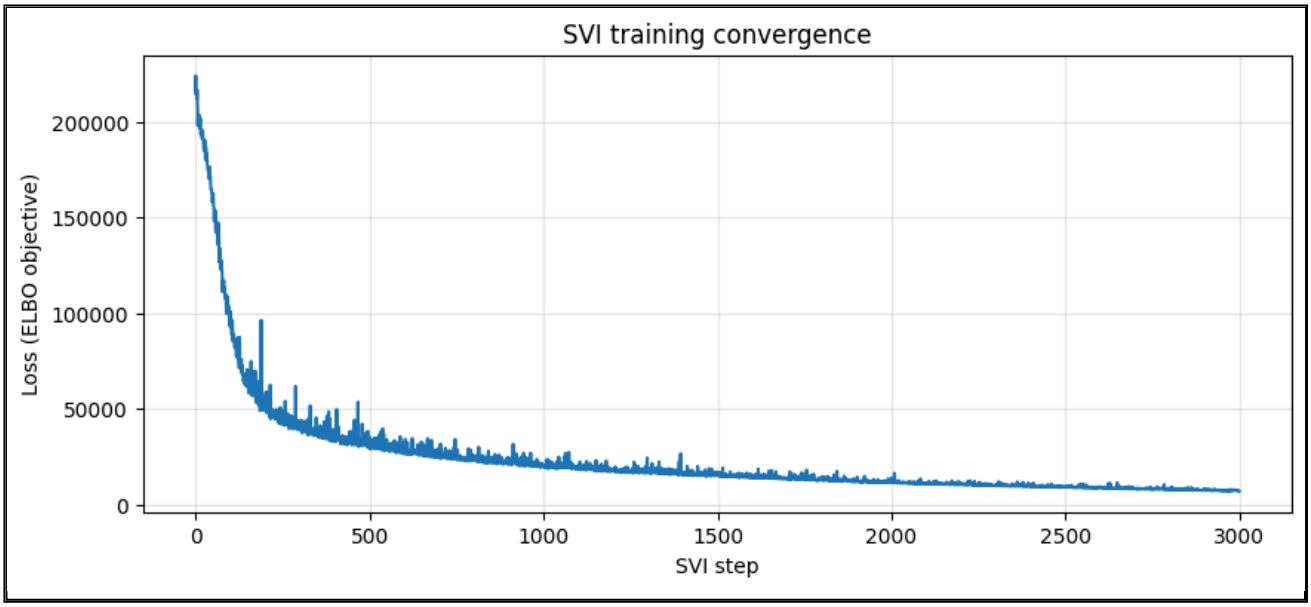
# utilities

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def _to_numpy(x):
    """Robust conversion: torch tensor / list / numpy / pandas -> numpy array."""
    try:
        import torch
        if isinstance(x, torch.Tensor):
            return x.detach().cpu().numpy()
    except Exception:
        pass
    if isinstance(x, (pd.Series, pd.DataFrame)):
        return x.to_numpy()
    return np.asarray(x)

def _ensure_1d_int(y):
    y = _to_numpy(y).reshape(-1)
    return y.astype(int)
```

```
def _ensure_2d_float(P):
    P = _to_numpy(P)
    if P.ndim == 1:
        P = P.reshape(-1, 1)
    return P.astype(float)

def predictive_entropy(P, eps=1e-12):
    """Row-wise entropy for probability matrix P (N x K)."""
    P = np.clip(P, eps, 1.0)
    return -np.sum(P * np.log(P), axis=1)

def one_hot(y, K):
    y = _ensure_1d_int(y)
    oh = np.zeros((y.size, K), dtype=float)
    oh[np.arange(y.size), y] = 1.0
    return oh
```

# Training convergence

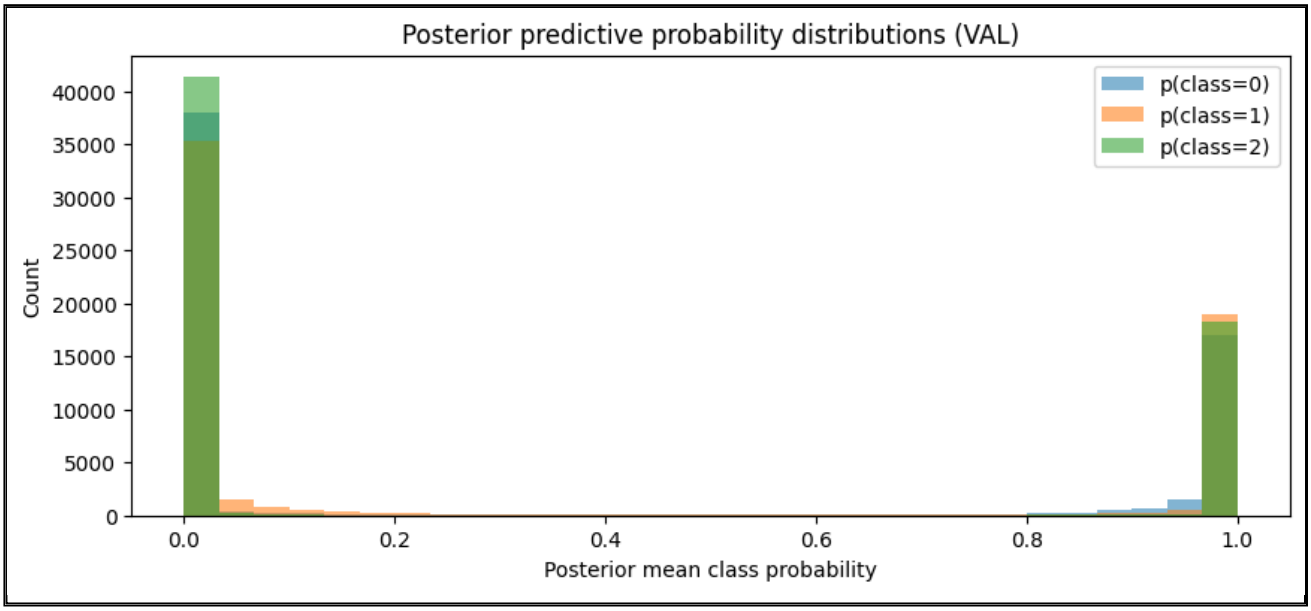
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(loss_history)
ax.set_xlabel("SVI step")
ax.set_ylabel("Loss (ELBO objective)")
ax.set_title("SVI training convergence")
ax.grid(True, alpha=0.3)
plt.show()
```



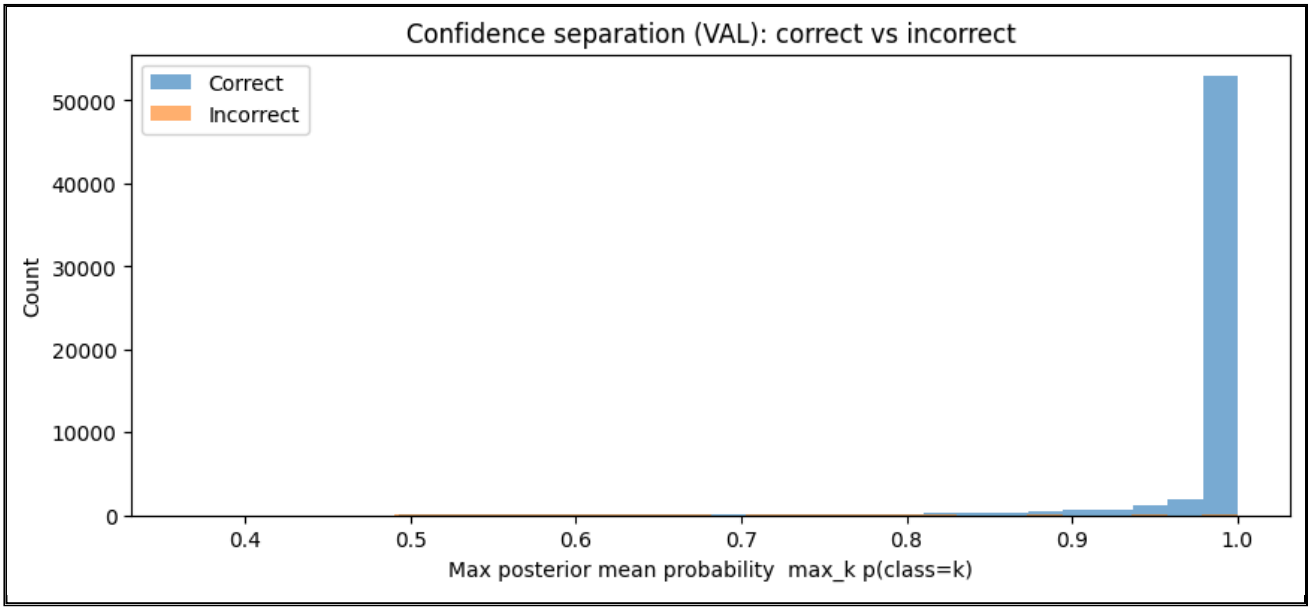

# Posterior predictive probability distributions

```
# Choose split
P = globals().get("probs_val", None)
y = globals().get("y_val", None)

if P is None or y is None:
```

```
    print("Missing probs_val/y_val. Switch to probs_test/y_test or run Predictive to
create them.")
else:
    P = _ensure_2d_float(P)
    y = _ensure_1d_int(y)
    K = P.shape[1]

    fig, ax = plt.subplots(figsize=(10, 4))
    for k in range(K):
        ax.hist(P[:, k], bins=30, alpha=0.55, label=f"p(class={k})")
    ax.set_xlabel("Posterior mean class probability")
    ax.set_ylabel("Count")
    ax.set_title("Posterior predictive probability distributions (VAL)")
    ax.legend()
    plt.show()
```

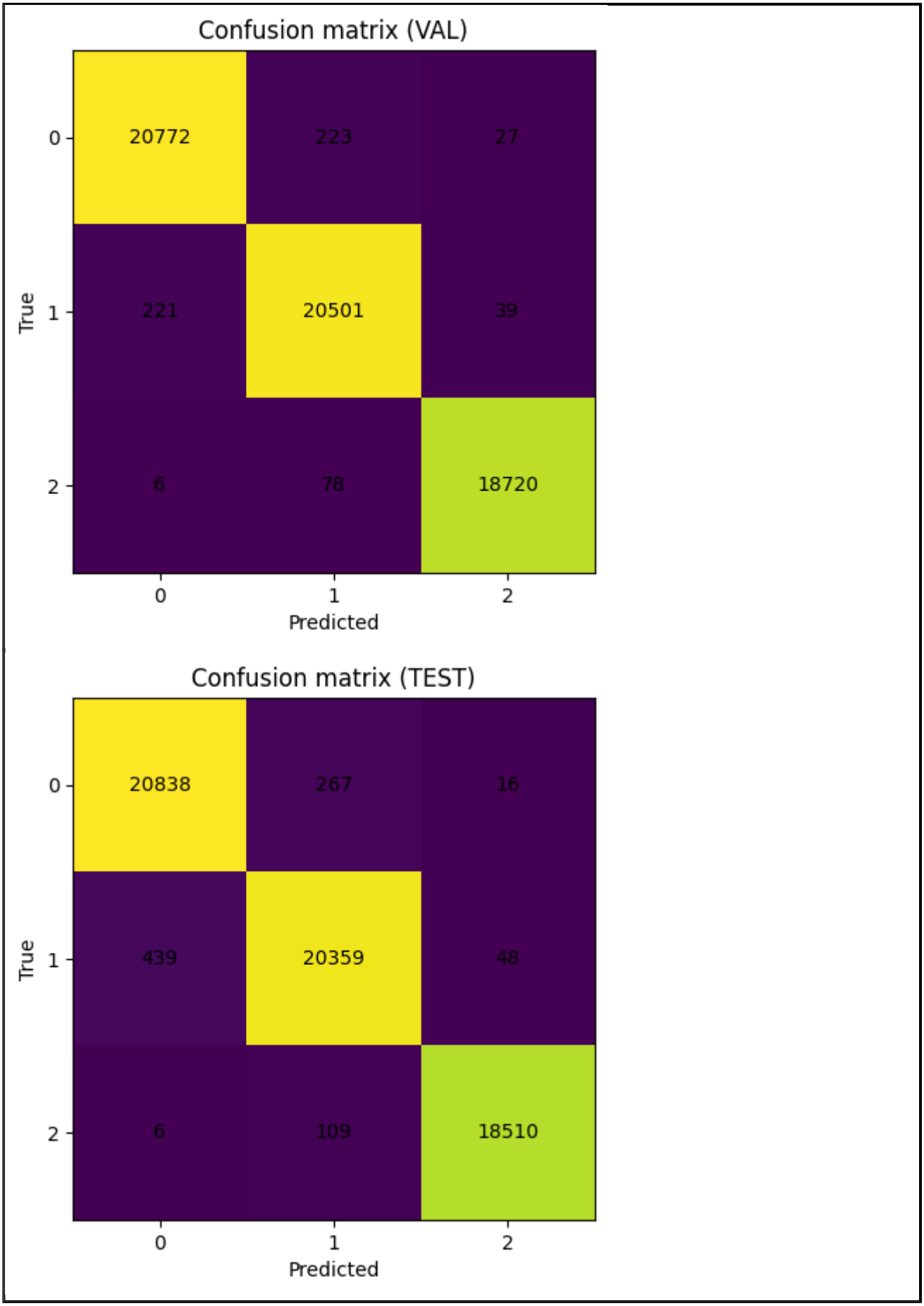



```
P = globals().get("probs_val", None)
y = globals().get("y_val", None)

if P is None or y is None:
    print("Missing probs_val/y_val.")
else:
    P = _ensure_2d_float(P)
    y = _ensure_1d_int(y)
    y_pred = np.argmax(P, axis=1)

    p_max = np.max(P, axis=1)
    correct = (y_pred == y)

    fig, ax = plt.subplots(figsize=(10, 4))
    ax.hist(p_max[correct], bins=30, alpha=0.6, label="Correct")
    ax.hist(p_max[~correct], bins=30, alpha=0.6, label="Incorrect")
    ax.set_xlabel("Max posterior mean probability  max_k p(class=k)")
    ax.set_ylabel("Count")
    ax.set_title("Confidence separation (VAL): correct vs incorrect")
    ax.legend()
    plt.show()
```



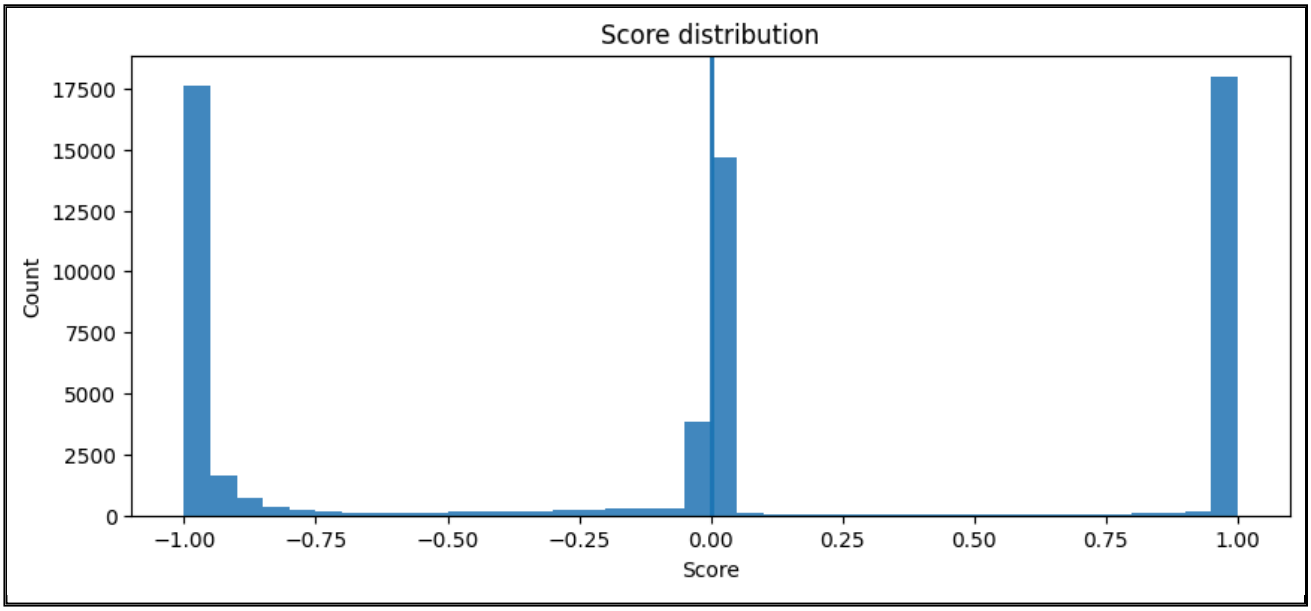

# Confusion matrix heatmaps

```
from sklearn.metrics import confusion_matrix

def plot_cm(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots(figsize=(5.6, 4.8))
    im = ax.imshow(cm, interpolation="nearest")
    ax.set_title(title)
    ax.set_xlabel("Predicted")
    ax.set_ylabel("True")
    ax.set_xticks(range(cm.shape[1]))
    ax.set_yticks(range(cm.shape[0]))
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, str(cm[i, j]), ha="center", va="center")
    plt.show()
    return cm

# VAL
if "probs_val" in globals() and "y_val" in globals():
    P_val = _ensure_2d_float(probs_val)
    y_val_np = _ensure_1d_int(y_val)
    y_pred_val = np.argmax(P_val, axis=1)
    cm_val = plot_cm(y_val_np, y_pred_val, "Confusion matrix (VAL)")

# TEST
if "probs_test" in globals() and "y_test" in globals():
    P_test = _ensure_2d_float(probs_test)
    y_test_np = _ensure_1d_int(y_test)
    y_pred_test = np.argmax(P_test, axis=1)
    cm_test = plot_cm(y_test_np, y_pred_test, "Confusion matrix (TEST)")
```




“Score distribution”

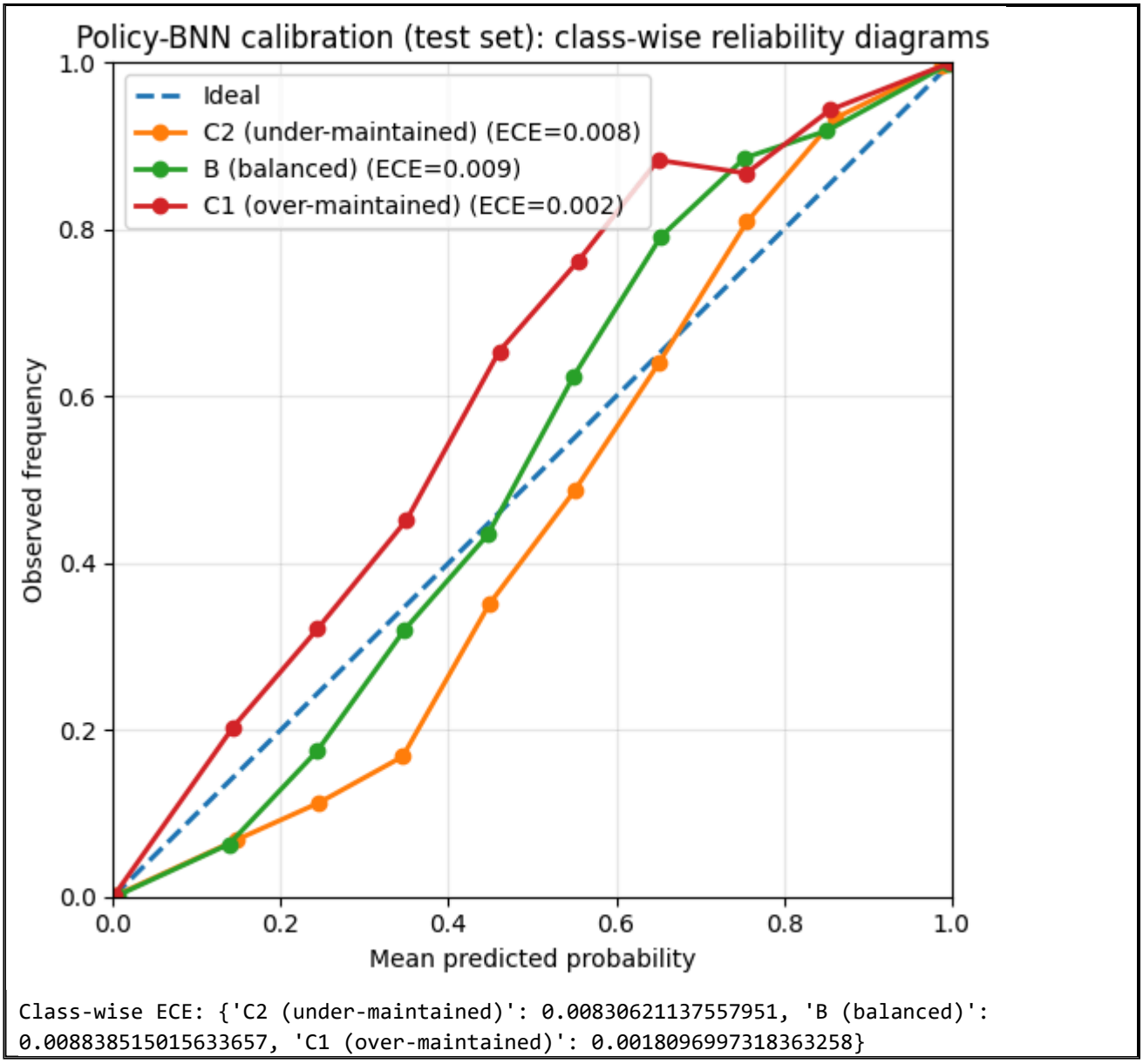
```
# IMPORTANT: define score exactly as in your notebook cell.
# Examples (do NOT use unless it matches your code):
# score = probs_test[:, 2] - probs_test[:, 0]
# score = np.log(probs_test[:, 2] + 1e-12) - np.log(probs_test[:, 0] + 1e-12)

# --- Replace this line with your notebook's definition ---
score = probs_test[:,2] - probs_test[:,0]
# ---------------------------------------------------------

if score is None:
    print("Set `score = ...` exactly as in your HTML cell (copy the expression).")
else:
    score = _to_numpy(score).reshape(-1)
    fig, ax = plt.subplots(figsize=(10, 4))
    ax.hist(score, bins=40, alpha=0.85)
    ax.axvline(0.0, linewidth=2)  # matches your red vertical line concept
    ax.set_xlabel("Score")
    ax.set_ylabel("Count")
    ax.set_title("Score distribution")
    plt.show()
```




```
# --- Multiclass calibration (Policy-BNN): class-wise reliability diagrams ---
# Paste this AFTER you have `probs_test` (N x 3) and `y_test` (N,)
# Works also for probs_val / y_val.

import numpy as np
import matplotlib.pyplot as plt

def reliability_curve_binary(p, y, n_bins=10):
    """
    Compute reliability curve for binary event with predicted prob p and true labels y in
{0,1}.
    Returns: bin_centers, conf, freq, bin_counts
    """
    p = np.asarray(p, dtype=float).reshape(-1)
    y = np.asarray(y, dtype=int).reshape(-1)
    assert p.shape[0] == y.shape[0], "p and y must have same length"

    # Bin edges (uniform in [0,1])
    edges = np.linspace(0.0, 1.0, n_bins + 1)
    bin_ids = np.digitize(p, edges, right=True) - 1
```

```
    bin_ids = np.clip(bin_ids, 0, n_bins - 1)

    conf = np.zeros(n_bins, dtype=float)
    freq = np.zeros(n_bins, dtype=float)
    counts = np.zeros(n_bins, dtype=int)

    for b in range(n_bins):
        mask = (bin_ids == b)
        counts[b] = int(mask.sum())
        if counts[b] > 0:
            conf[b] = float(p[mask].mean())
            freq[b] = float(y[mask].mean())
        else:
            conf[b] = np.nan
            freq[b] = np.nan

    bin_centers = 0.5 * (edges[:-1] + edges[1:])
    return bin_centers, conf, freq, counts

def ece_binary(p, y, n_bins=10):
    """
    Expected Calibration Error for binary probabilities.
    ECE = sum_k (|Bk|/n) * |freq(Bk) - conf(Bk)|
    """
    p = np.asarray(p, dtype=float).reshape(-1)
    y = np.asarray(y, dtype=int).reshape(-1)
    n = y.size

    edges = np.linspace(0.0, 1.0, n_bins + 1)
    bin_ids = np.digitize(p, edges, right=True) - 1
    bin_ids = np.clip(bin_ids, 0, n_bins - 1)

    ece = 0.0
    for b in range(n_bins):
        mask = (bin_ids == b)
        nb = int(mask.sum())
        if nb == 0:
            continue
        conf = float(p[mask].mean())
        freq = float(y[mask].mean())
        ece += (nb / n) * abs(freq - conf)
    return float(ece)

def plot_multiclass_reliability(P, y, class_names=None, n_bins=10, title="Policy-BNN
calibration (class-wise)"):
    """
    P: (N,K) posterior mean class probabilities
    y: (N,) true class labels in {0..K-1}
    """
    P = np.asarray(P, dtype=float)
    y = np.asarray(y, dtype=int).reshape(-1)
    assert P.ndim == 2, "P must be (N,K)"
    N, K = P.shape
    assert y.size == N, "y must have length N"

    if class_names is None:
        class_names = [f"class {k}" for k in range(K)]
    assert len(class_names) == K

    fig, ax = plt.subplots(figsize=(6.2, 6.2))
```

```
    ax.plot([0, 1], [0, 1], linestyle="--", linewidth=2, label="Ideal")

    eces = []
    for k in range(K):
        p_k = P[:, k]
        y_k = (y == k).astype(int)   # one-vs-rest labels
        centers, conf, freq, counts = reliability_curve_binary(p_k, y_k, n_bins=n_bins)
        ece_k = ece_binary(p_k, y_k, n_bins=n_bins)
        eces.append(ece_k)

        # Only plot bins that have samples
        mask = ~np.isnan(conf) & ~np.isnan(freq)
        ax.plot(conf[mask], freq[mask], marker="o", linewidth=2,
                label=f"{class_names[k]} (ECE={ece_k:.3f})")

    ax.set_xlabel("Mean predicted probability")
    ax.set_ylabel("Observed frequency")
    ax.set_title(title)
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.grid(True, alpha=0.3)
    ax.legend()
    plt.show()

    return eces

# ---- choose split here ----
P = probs_test   # or probs_val
y = y_test       # or y_val

class_names = ["C2 (under-maintained)", "B (balanced)", "C1 (over-maintained)"]

eces = plot_multiclass_reliability(
    P, y,
    class_names=class_names,
    n_bins=10,
    title="Policy-BNN calibration (test set): class-wise reliability diagrams"
)

print("Class-wise ECE:", dict(zip(class_names, eces)))

# Optional: save for thesis inclusion
# plt.savefig("policy_bnn_calibration_test.pdf", bbox_inches="tight", dpi=300)
```




```
Class-wise ECE: {'C2 (under-maintained)': 0.00830621137557951, 'B (balanced)':
0.008838515015633657, 'C1 (over-maintained)': 0.0018096997318363258}
```

# E Phase-BNN Notebook

# Phase-BNN (MC Dropout) — Bayesian Phase Mapping

This notebook trains a **Phase-BNN** to classify the maintenance phase **B / C1 / C2** and then projects **D tokens** (xs, vs, s, m, vm, xm) into that phase space using **posterior predictive sampling** (MC Dropout).

Outputs produced:

- Confusion matrix table (TEST)
- KPI table (TEST)
- Bayesian uncertainty summary (MI statistics)
- D-token → phase mapping table (P(B), P(C1), P(C2), MI)
- Plot: phase probabilities by token
- Plot: epistemic uncertainty (MI) by token

```
# Cell 1: Configuration
from pathlib import Path

# --- Set your DB path here ---
DB_PATH = Path(r"normalized_Exp_DB_892.db")

# --- Training hyperparameters ---
RANDOM_SEED = 42
EPOCHS = 12
BATCH_SIZE = 512
LR = 1e-3
WEIGHT_DECAY = 1e-4
DROPOUT_P = 0.25
HIDDEN = 128

# --- Bayesian MC sampling ---
MC_SAMPLES = 80
```

```
# Cell 2: Load data from SQLite and join ExpNr
import sqlite3
import pandas as pd

assert DB_PATH.exists(), f"DB not found: {DB_PATH}"
con = sqlite3.connect(str(DB_PATH))

sql = (
    "SELECT bf.*, r.ExperimentID, e.ExpNr "
    "FROM BNN_Features bf "
    "JOIN Run r ON r.RunID = bf.Run "
    "JOIN Experiment e ON e.ExperimentID = r.ExperimentID"
)

df = pd.read_sql_query(sql, con)

print("Loaded rows/cols:", df.shape)
df[["Run","ExperimentID","ExpNr"]].head(10)
```

```
Loaded rows/cols: (530709, 20)
```

| | Run | ExperimentID | ExpNr |
|---|---|---|---|
| **0** | 1 | 1 | Exp_A_79_DC |
| **1** | 1 | 1 | Exp_A_79_DC |
| **2** | 1 | 1 | Exp_A_79_DC |
| **3** | 1 | 1 | Exp_A_79_DC |
| **4** | 1 | 1 | Exp_A_79_DC |
| **5** | 1 | 1 | Exp_A_79_DC |
| **6** | 1 | 1 | Exp_A_79_DC |
| **7** | 1 | 1 | Exp_A_79_DC |
| **8** | 1 | 1 | Exp_A_79_DC |
| **9** | 1 | 1 | Exp_A_79_DC |

```
# Cell 3: Parse ExpNr into family/level/policy/token
import re
import pandas as pd

def parse_exp(expnr: str) -> pd.Series:
    expnr = str(expnr)

    # Numeric-level experiments: Exp_A_79_DC, Exp_C1_84_CC, Exp_C2_99_DC, ...
    m_num = re.match(r"^Exp_([A-Z]\d?)_(\d+)_(CC|DC)$", expnr)
    if m_num:
        return pd.Series({
            "family": m_num.group(1),
            "level_num": int(m_num.group(2)),
            "level_tok": None,
            "policy": m_num.group(3),
        })

    # Token-level experiments (D): Exp_D_m_DC, Exp_D_xs_CC, ...
```

```
    m_tok = re.match(r"^Exp_(D)_(xs|vs|s|m|vm|xm)_(CC|DC)$", expnr)
    if m_tok:
        return pd.Series({
            "family": m_tok.group(1),
            "level_num": None,
            "level_tok": m_tok.group(2),
            "policy": m_tok.group(3),
        })

    return pd.Series({"family": None, "level_num": None, "level_tok": None, "policy":
None})

parsed = df["ExpNr"].apply(parse_exp)
df = pd.concat([df, parsed], axis=1)

unparsed = float(df["family"].isna().mean())
print("Unparsed share:", unparsed)
df["family"].value_counts(dropna=False)
```

```
Unparsed share: 0.0
family
D     125432
C2    105840
B     104200
A     101678
C1     93559
Name: count, dtype: int64
```

```
# Cell 4: Create phase label y_phase for B / C1 / C2 (D is excluded from training)
import numpy as np

phase_map = {"B": 0, "C1": 1, "C2": 2}
df["y_phase"] = df["family"].map(phase_map)

print("Phase label coverage (non-null share):", float(df["y_phase"].notna().mean()))
df[["ExpNr","family","level_num","level_tok","policy","y_phase"]].head(10)
```

```
Phase label coverage (non-null share): 0.5720630326600831
```

| | ExpNr | family | level_num | level_tok | policy | y_phase |
|---|---|---|---|---|---|---|
| **0** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |
| **1** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |
| **2** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |
| **3** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |
| **4** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |
| **5** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |
| **6** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |
| **7** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |
| **8** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |
| **9** | Exp_A_79_DC | A | 79.0 | NaN | DC | NaN |

```
# Cell 5: Select feature columns (exclude leakage and identifiers)
import pandas as pd

LEAK = {"y", "Dt_ToNextMaint"}  # likely targets / future-looking
EXCLUDE = {"Run", "ExperimentID", "ExpNr", "family", "level_num", "level_tok", "policy",
"y_phase"} | LEAK

feature_cols = [c for c in df.columns if c not in EXCLUDE and
pd.api.types.is_numeric_dtype(df[c])]

print("n_features:", len(feature_cols))
feature_cols
```

```
n_features: 15
['In_EndTimeStamp',
 'DeltaT',
 'CT_Median_In',
 'CT_IQR_In',
 'CT_P95_In',
 'CT_Median_Out',
 'CT_IQR_Out',
 'CT_P95_Out',
 'DowntimeWin',
 'FailCountWin',
 'MTTRWin',
 'TimeSinceFailEnd',
 'HasFailureHistory',
 'TimeSinceMaintEnd',
 'BufferLevel']
```

```
# Cell: Clean non-finite rows BEFORE split
import numpy as np

df = df.copy()
df[feature_cols] = df[feature_cols].replace([np.inf, -np.inf], np.nan)

bad = df[feature_cols].isna().any(axis=1)
print("Rows with NaN in features:", int(bad.sum()), "out of", len(df))

df = df.loc[~bad].copy()
print("After cleaning:", df.shape)
```

```
Rows with NaN in features: 2078 out of 530709
After cleaning: (528631, 25)
```

```
# Cell: Clean D rows (non-finite features)
import numpy as np

# Work on the already cleaned df (after global cleaning)
D = df[df["family"] == "D"].copy()

mask_D = np.isfinite(D[feature_cols].values).all(axis=1)
print("Dropping D rows:", int((~mask_D).sum()), "out of", len(D))

D = D.loc[mask_D].copy()

# rebuild X_D and tokens
X_D = D[feature_cols].astype(float).values
tok_D = D["level_tok"].astype(str).values
```

```
print("Remaining D rows:", len(D))
```

```
Dropping D rows: 0 out of 125006
Remaining D rows: 125006
```

```
# Cell 6: Build Train/Val/Test split by Run (prevents window leakage across splits)
import numpy as np
from sklearn.model_selection import train_test_split

phase_df = df[df["y_phase"].notna()].copy()

runs = phase_df["Run"].unique()
r_tr, r_tmp = train_test_split(runs, test_size=0.30, random_state=RANDOM_SEED)
r_va, r_te  = train_test_split(r_tmp, test_size=0.50, random_state=RANDOM_SEED)

train = phase_df[phase_df["Run"].isin(r_tr)].copy()
val   = phase_df[phase_df["Run"].isin(r_va)].copy()
test  = phase_df[phase_df["Run"].isin(r_te)].copy()

print("rows Train/Val/Test:", train.shape, val.shape, test.shape)
print("Train class counts:", train["family"].value_counts().to_dict())
```

```
rows Train/Val/Test: (211540, 25) (44797, 25) (45984, 25)
Train class counts: {'C2': 77582, 'C1': 69366, 'B': 64592}
```

```
# Cell 7: Standardize features using Train only
from sklearn.preprocessing import StandardScaler
import numpy as np

X_tr = train[feature_cols].astype(float).values
y_tr = train["y_phase"].astype(int).values

X_va = val[feature_cols].astype(float).values
y_va = val["y_phase"].astype(int).values

X_te = test[feature_cols].astype(float).values
y_te = test["y_phase"].astype(int).values

scaler = StandardScaler()
X_tr_s = scaler.fit_transform(X_tr)
X_va_s = scaler.transform(X_va)
X_te_s = scaler.transform(X_te)

print("Shapes:", X_tr_s.shape, X_va_s.shape, X_te_s.shape)
```

```
Shapes: (211540, 15) (44797, 15) (45984, 15)
```

```
# Cell 8: Define MC Dropout Phase-BNN (PyTorch)
import torch
import torch.nn as nn
import torch.nn.functional as F

device = "cuda" if torch.cuda.is_available() else "cpu"
torch.manual_seed(RANDOM_SEED)

class MCDropoutMLP(nn.Module):
    def __init__(self, d_in, d_out=3, p_drop=0.25, h=128):
        super().__init__()
        self.fc1 = nn.Linear(d_in, h)
```

```
        self.fc2 = nn.Linear(h, h)
        self.out = nn.Linear(h, d_out)
        self.drop = nn.Dropout(p_drop)

    def forward(self, x):
        x = F.relu(self.fc1(x)); x = self.drop(x)
        x = F.relu(self.fc2(x)); x = self.drop(x)
        return self.out(x)

model = MCDropoutMLP(d_in=X_tr_s.shape[1], d_out=3, p_drop=DROPOUT_P,
h=HIDDEN).to(device)
model
```

```
MCDropoutMLP(
  (fc1): Linear(in_features=15, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=128, bias=True)
  (out): Linear(in_features=128, out_features=3, bias=True)
  (drop): Dropout(p=0.25, inplace=False)
)
```

```
# Cell: Guard — ensure no NaN/inf in matrices
import numpy as np

def assert_finite(X, name):
    X = np.asarray(X)
    n_bad = int((~np.isfinite(X)).sum())
    assert n_bad == 0, f"{name} contains {n_bad} non-finite values."

assert_finite(X_tr, "X_tr")
assert_finite(X_va, "X_va")
assert_finite(X_te, "X_te")
if "X_D" in globals():
    assert_finite(X_D, "X_D")

print("All matrices are finite.")
```

```
All matrices are finite.
```

```
# Cell 9: Train Phase-BNN with validation accuracy each epoch
from torch.utils.data import TensorDataset, DataLoader
from sklearn.metrics import accuracy_score

Xtr_t = torch.tensor(X_tr_s, dtype=torch.float32)
ytr_t = torch.tensor(y_tr, dtype=torch.long)

dl = DataLoader(TensorDataset(Xtr_t, ytr_t), batch_size=BATCH_SIZE, shuffle=True)

opt = torch.optim.Adam(model.parameters(), lr=LR, weight_decay=WEIGHT_DECAY)
loss_fn = nn.CrossEntropyLoss()

Xva_t = torch.tensor(X_va_s, dtype=torch.float32).to(device)

for epoch in range(EPOCHS):
    model.train()
    total_loss = 0.0
    for xb, yb in dl:
        xb, yb = xb.to(device), yb.to(device)
        opt.zero_grad()
        loss = loss_fn(model(xb), yb)
        loss.backward()
```

```
        opt.step()
        total_loss += float(loss) * len(xb)

    model.eval()
    with torch.no_grad():
        pva = torch.softmax(model(Xva_t), dim=1).cpu().numpy()
    acc_va = accuracy_score(y_va, pva.argmax(axis=1))
    print(f"epoch {epoch+1:02d}  loss={total_loss/len(Xtr_t):.4f}  val_acc={acc_va:.4f}")
```

```
C:\Users\marcp\AppData\Local\Temp\ipykernel_4220\795478443.py:24: UserWarning: Converting
a tensor with requires_grad=True to a scalar may lead to unexpected behavior.
Consider using tensor.detach() first. (Triggered internally at C:\actions-
runner\_work\pytorch\pytorch\pytorch\torch\csrc\autograd\generated\python_variable_method
s.cpp:837.)
  total_loss += float(loss) * len(xb)
epoch 01  loss=0.6473  val_acc=0.5528
epoch 02  loss=0.5662  val_acc=0.5387
epoch 03  loss=0.5480  val_acc=0.5769
epoch 04  loss=0.5381  val_acc=0.5514
epoch 05  loss=0.5306  val_acc=0.5601
epoch 06  loss=0.5254  val_acc=0.5862
epoch 07  loss=0.5202  val_acc=0.6619
epoch 08  loss=0.5162  val_acc=0.6056
epoch 09  loss=0.5132  val_acc=0.6087
epoch 10  loss=0.5101  val_acc=0.5955
epoch 11  loss=0.5086  val_acc=0.6178
epoch 12  loss=0.5060  val_acc=0.5973
```

```
# Cell 10: TEST evaluation — Confusion matrix table
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix

model.eval()
Xte_t = torch.tensor(X_te_s, dtype=torch.float32).to(device)

with torch.no_grad():
    pte = torch.softmax(model(Xte_t), dim=1).cpu().numpy()

yhat = pte.argmax(axis=1)
cm = confusion_matrix(y_te, yhat)

cm_df = pd.DataFrame(cm, index=["true_B","true_C1","true_C2"], columns=
["pred_B","pred_C1","pred_C2"])
cm_df
```

| | pred_B | pred_C1 | pred_C2 |
|---|---|---|---|
| **true_B** | 6474 | 1258 | 4660 |
| **true_C1** | 479 | 17782 | 432 |
| **true_C2** | 5190 | 696 | 9013 |

```
# Cell 11: TEST KPI table (accuracy, balanced accuracy, macro F1)
import pandas as pd
from sklearn.metrics import accuracy_score, balanced_accuracy_score, f1_score

acc = accuracy_score(y_te, yhat)
```

```
bacc = balanced_accuracy_score(y_te, yhat)
f1m = f1_score(y_te, yhat, average="macro")

kpi = pd.DataFrame([{
    "n_test": int(len(y_te)),
    "accuracy": float(acc),
    "balanced_accuracy": float(bacc),
    "macro_f1": float(f1m),
}])

kpi
```

| | n_test | accuracy | balanced_accuracy | macro_f1 |
|---|---|---|---|---|
| **0** | 45984 | 0.723491 | 0.69288 | 0.691561 |

```
# Cell 12: Bayesian posterior predictive sampling utilities (MC Dropout)
import numpy as np
import torch

@torch.no_grad()
def mc_predict_proba(model, X_tensor, T=80):
    # Keep dropout ON
    model.train()
    probs = []
    for _ in range(T):
        logits = model(X_tensor)
        probs.append(torch.softmax(logits, dim=1).cpu().numpy())
    return np.stack(probs, axis=0)  # (T,N,K)

def predictive_uncertainty(probs_T):
    eps = 1e-12
    p_mean = probs_T.mean(axis=0)  # (N,K)
    H_pred = -np.sum(p_mean * np.log(p_mean + eps), axis=1)                          # total
    H_exp  = (-np.sum(probs_T * np.log(probs_T + eps), axis=2)).mean(axis=0)      #
expected
    MI     = H_pred - H_exp                                                        #
epistemic
    return p_mean, H_pred, MI
```

```
# Cell 13: Bayesian uncertainty summary on TEST (MI)
import pandas as pd
import numpy as np

probs_T_te = mc_predict_proba(model, Xte_t, T=MC_SAMPLES)
p_mean_te, H_pred_te, MI_te = predictive_uncertainty(probs_T_te)

summary = pd.DataFrame([{
    "MI_mean": float(MI_te.mean()),
    "MI_p50": float(np.quantile(MI_te, 0.50)),
    "MI_p90": float(np.quantile(MI_te, 0.90)),
    "MI_p99": float(np.quantile(MI_te, 0.99)),
    "H_pred_mean": float(H_pred_te.mean())
}])

summary
```

| | MI_mean | MI_p50 | MI_p90 | MI_p99 | H_pred_mean |
|---|---|---|---|---|---|
| **0** | 0.013361 | 0.007194 | 0.033837 | 0.073906 | 0.516252 |

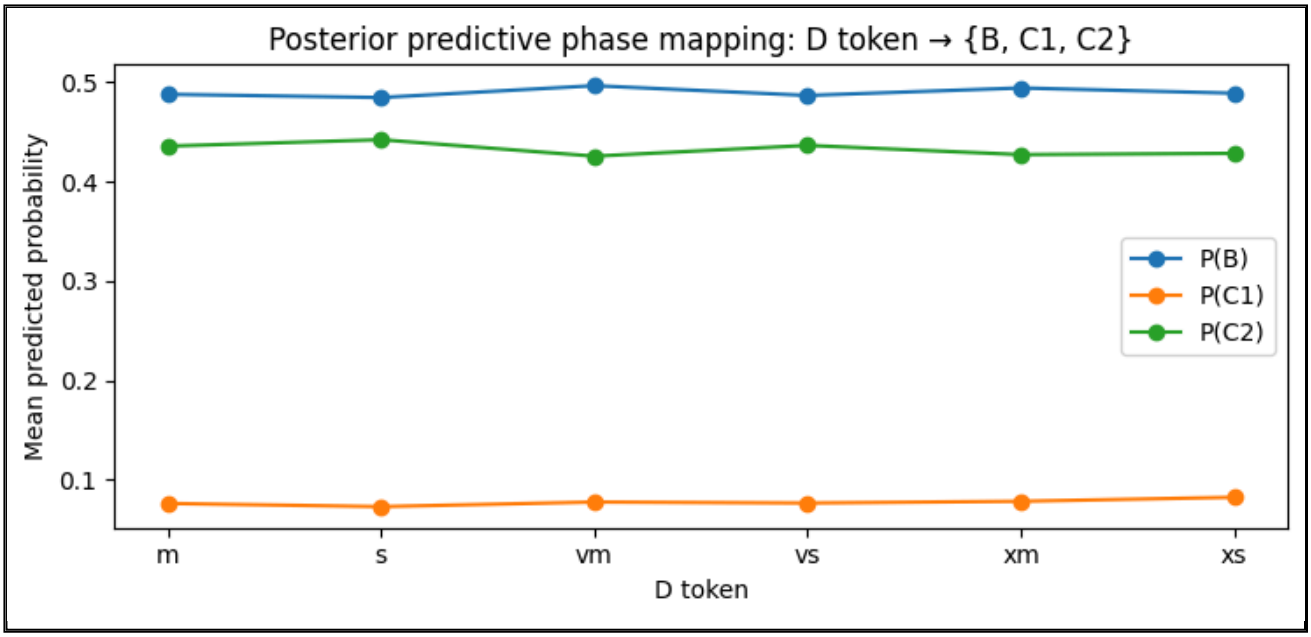
```
# Cell 14: Project D samples into phase space (posterior predictive + MI)
import pandas as pd
import numpy as np
import torch

D = df[df["family"] == "D"].copy()
assert len(D) > 0, "No family D rows found. Check parsing or DB content."

X_D = scaler.transform(D[feature_cols].astype(float).values)
tok_D = D["level_tok"].astype(str).values

XD_t = torch.tensor(X_D, dtype=torch.float32).to(device)

probs_T_D = mc_predict_proba(model, XD_t, T=MC_SAMPLES)
p_mean_D, H_pred_D, MI_D = predictive_uncertainty(probs_T_D)

print("D rows:", len(D), "tokens:", sorted(pd.unique(tok_D)))
```

```
D rows: 125006 tokens: ['m', 's', 'vm', 'vs', 'xm', 'xs']
```

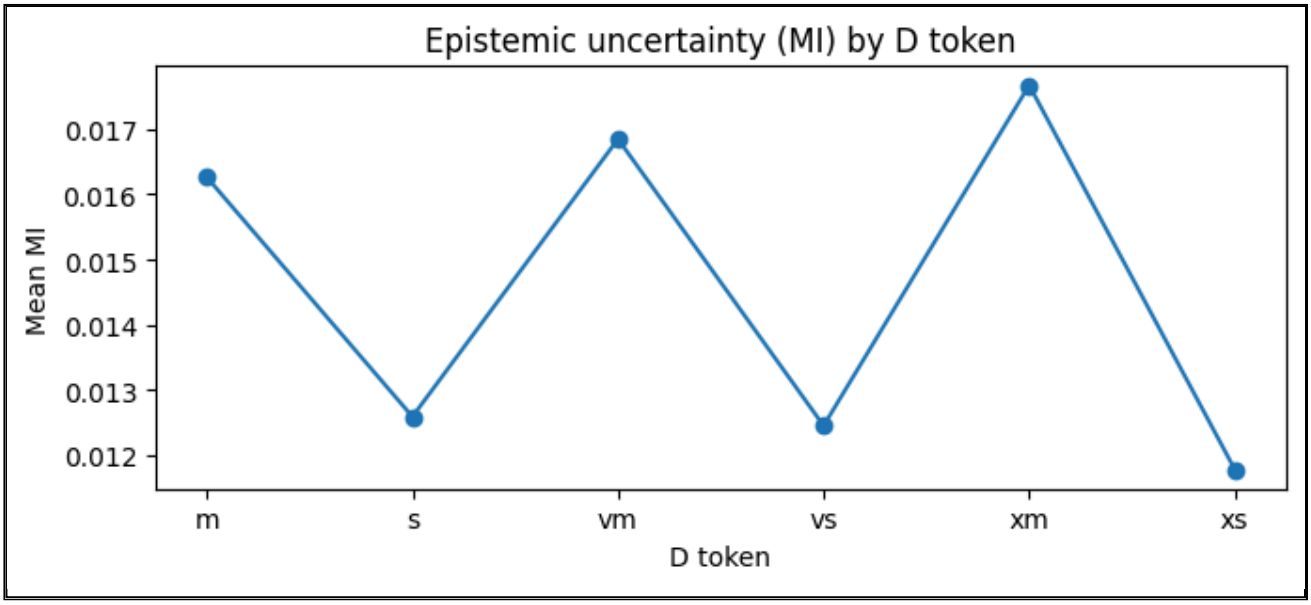
```
# Cell 15: D-token → phase mapping table
import pandas as pd
import numpy as np

id_to_phase = {0:"B", 1:"C1", 2:"C2"}

rows = []
for tok in sorted(pd.unique(tok_D)):
    idx = np.where(tok_D == tok)[0]
    p_tok = p_mean_D[idx].mean(axis=0)
    mi_tok = MI_D[idx].mean()
    ent_tok = float(-np.sum(p_tok * np.log(p_tok + 1e-12)))
    top = int(np.argmax(p_tok))
    rows.append({
        "D_token": tok,
        "n": int(len(idx)),
        "P(B)": float(p_tok[0]),
        "P(C1)": float(p_tok[1]),
        "P(C2)": float(p_tok[2]),
        "mapped_phase": id_to_phase[top],
        "entropy_total": ent_tok,
        "epistemic_MI": float(mi_tok),
    })

map_table = pd.DataFrame(rows).sort_values("D_token")
map_table
```

| | D_token | n | P(B) | P(C1) | P(C2) | mapped_phase | entropy_total | epistemic_MI |
|---|---|---|---|---|---|---|---|---|
| **0** | m | 20979 | 0.487866 | 0.076390 | 0.435743 | B | 0.908589 | 0.016264 |
| **1** | s | 20742 | 0.484613 | 0.073185 | 0.442200 | B | 0.903249 | 0.012588 |
| **2** | vm | 21013 | 0.496535 | 0.077828 | 0.425636 | B | 0.909905 | 0.016850 |
| **3** | vs | 20671 | 0.486706 | 0.076811 | 0.436480 | B | 0.909451 | 0.012449 |
| **4** | xm | 21115 | 0.494236 | 0.078565 | 0.427199 | B | 0.911500 | 0.017676 |
| **5** | xs | 20486 | 0.488964 | 0.082567 | 0.428465 | B | 0.918916 | 0.011778 |

```
# Cell 16: Plot — phase probabilities per D token
import matplotlib.pyplot as plt

toks = map_table["D_token"].tolist()

plt.figure(figsize=(7.5, 3.6))
plt.plot(toks, map_table["P(B)"].values, marker="o", label="P(B)")
plt.plot(toks, map_table["P(C1)"].values, marker="o", label="P(C1)")
plt.plot(toks, map_table["P(C2)"].values, marker="o", label="P(C2)")
plt.title("Posterior predictive phase mapping: D token → {B, C1, C2}")
plt.xlabel("D token")
plt.ylabel("Mean predicted probability")
plt.legend()
plt.tight_layout()
plt.show()
```




```
# Cell 17: Plot — epistemic uncertainty (MI) per D token
import matplotlib.pyplot as plt

plt.figure(figsize=(7.0, 3.2))
plt.plot(map_table["D_token"], map_table["epistemic_MI"], marker="o")
plt.title("Epistemic uncertainty (MI) by D token")
plt.xlabel("D token")
plt.ylabel("Mean MI")
plt.tight_layout()
plt.show()
```




```
# Cell 18: Save outputs (optional)
from pathlib import Path

out_dir = Path(".")
map_table.to_csv(out_dir / "PhaseBNN_D_token_mapping.csv", index=False)
kpi.to_csv(out_dir / "PhaseBNN_test_kpi.csv", index=False)
cm_df.to_csv(out_dir / "PhaseBNN_test_confusion_matrix.csv")
summary.to_csv(out_dir / "PhaseBNN_test_uncertainty_summary.csv", index=False)

print("Saved CSVs to:", out_dir.resolve())
```

```
Saved CSVs to: C:\Learning\University\Thesis\BNN\Project
```

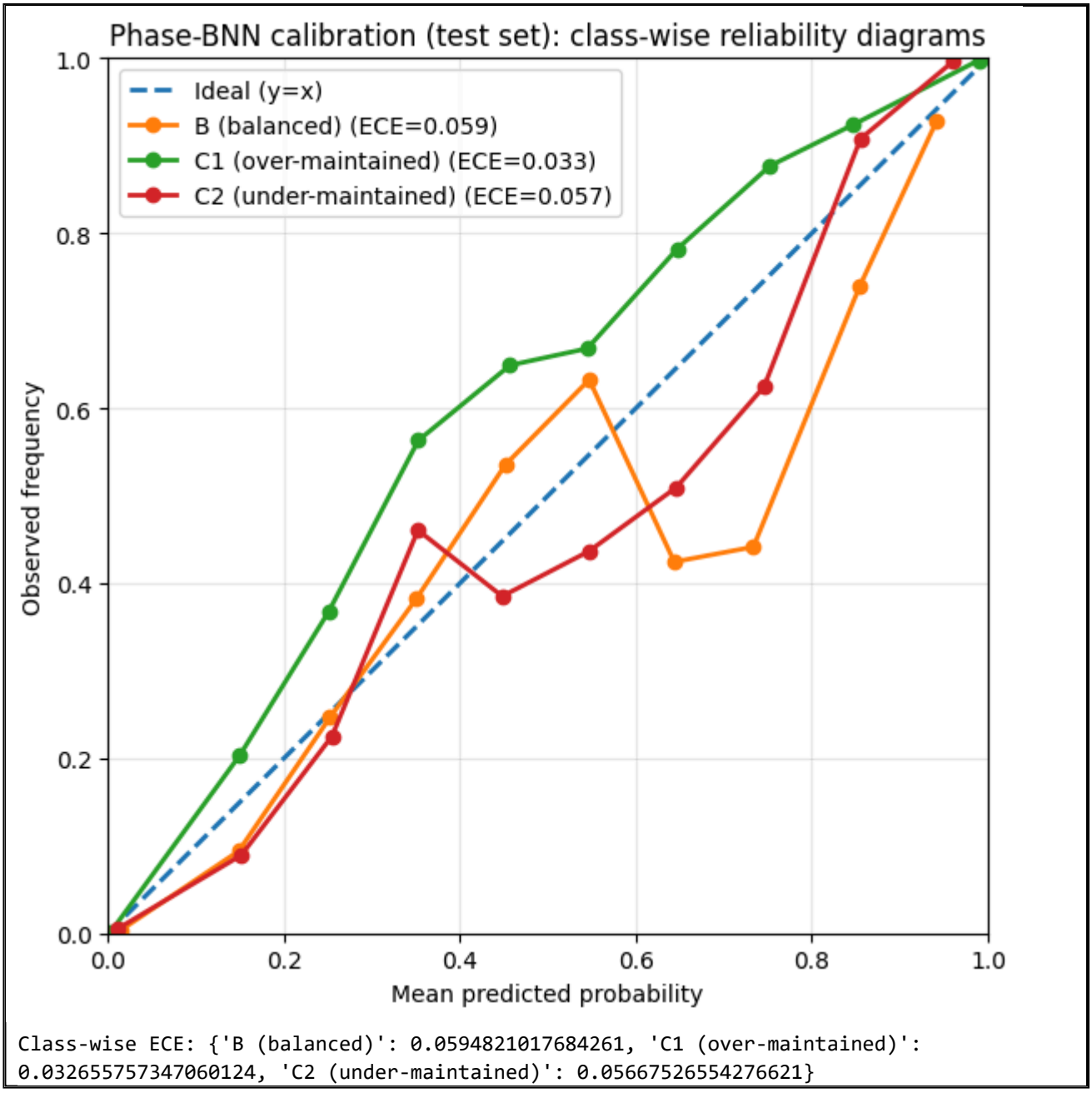
```
map_table["B_minus_C2"] = map_table["P(B)"] - map_table["P(C2)"]
map_table[["D_token", "B_minus_C2"]]
```

| | D_token | B_minus_C2 |
|---|---|---|
| **0** | m | 0.052123 |
| **1** | s | 0.042413 |
| **2** | vm | 0.070900 |
| **3** | vs | 0.050226 |
| **4** | xm | 0.067036 |
| **5** | xs | 0.060499 |

```
# --- Phase-BNN calibration (multiclass): class-wise reliability diagrams + class-wise
ECE ---
# Requires:
#   p_mean_te : (N,3) posterior mean probabilities on TEST (from MC Dropout)
#   y_te      : (N,)  true labels in {0,1,2}

import numpy as np
import matplotlib.pyplot as plt

def reliability_curve_binary(p, y, n_bins=10):
    """
```

```
    Reliability curve for a binary event:
      p: predicted prob in [0,1]
      y: true labels in {0,1}
    Returns: conf, freq, counts (each length n_bins) + bin_centers
    """
    p = np.asarray(p, dtype=float).reshape(-1)
    y = np.asarray(y, dtype=int).reshape(-1)
    assert p.shape[0] == y.shape[0], "p and y must have same length"

    edges = np.linspace(0.0, 1.0, n_bins + 1)
    bin_ids = np.digitize(p, edges, right=True) - 1
    bin_ids = np.clip(bin_ids, 0, n_bins - 1)

    conf = np.full(n_bins, np.nan, dtype=float)
    freq = np.full(n_bins, np.nan, dtype=float)
    counts = np.zeros(n_bins, dtype=int)

    for b in range(n_bins):
        mask = (bin_ids == b)
        counts[b] = int(mask.sum())
        if counts[b] > 0:
            conf[b] = float(p[mask].mean())
            freq[b] = float(y[mask].mean())

    bin_centers = 0.5 * (edges[:-1] + edges[1:])
    return bin_centers, conf, freq, counts

def ece_binary(p, y, n_bins=10):
    """
    Expected Calibration Error for binary event:
      ECE = sum_k (|Bk|/n) * |freq(Bk) - conf(Bk)|
    """
    p = np.asarray(p, dtype=float).reshape(-1)
    y = np.asarray(y, dtype=int).reshape(-1)
    n = y.size

    edges = np.linspace(0.0, 1.0, n_bins + 1)
    bin_ids = np.digitize(p, edges, right=True) - 1
    bin_ids = np.clip(bin_ids, 0, n_bins - 1)

    ece = 0.0
    for b in range(n_bins):
        mask = (bin_ids == b)
        nb = int(mask.sum())
        if nb == 0:
            continue
        conf = float(p[mask].mean())
        freq = float(y[mask].mean())
        ece += (nb / n) * abs(freq - conf)
    return float(ece)

def plot_multiclass_calibration(P, y, class_names=None, n_bins=10,
                                title="Phase-BNN calibration (test set)"):
    """
    Multiclass calibration via one-vs-rest reliability diagrams.
      P: (N,K) predicted class probabilities
      y: (N,) true class labels in {0..K-1}
    """
    P = np.asarray(P, dtype=float)
    y = np.asarray(y, dtype=int).reshape(-1)
```

```
    assert P.ndim == 2, "P must be (N,K)"
    N, K = P.shape
    assert y.size == N, "y must have length N"

    if class_names is None:
        class_names = [f"class {k}" for k in range(K)]
    assert len(class_names) == K

    fig, ax = plt.subplots(figsize=(6.2, 6.2))
    ax.plot([0, 1], [0, 1], linestyle="--", linewidth=2, label="Ideal (y=x)")

    eces = {}
    for k in range(K):
        p_k = P[:, k]
        y_k = (y == k).astype(int)

        _, conf, freq, counts = reliability_curve_binary(p_k, y_k, n_bins=n_bins)
        ece_k = ece_binary(p_k, y_k, n_bins=n_bins)
        eces[class_names[k]] = ece_k

        mask = np.isfinite(conf) & np.isfinite(freq)
        ax.plot(conf[mask], freq[mask], marker="o", linewidth=2,
                label=f"{class_names[k]} (ECE={ece_k:.3f})")

    ax.set_xlabel("Mean predicted probability")
    ax.set_ylabel("Observed frequency")
    ax.set_title(title)
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.grid(True, alpha=0.3)
    ax.legend()
    plt.tight_layout()
    plt.show()

    return eces

# ---- Use TEST split ----
P = p_mean_te   # (N,3) from your MC dropout posterior mean
y = y_te        # (N,) true labels

class_names = ["B (balanced)", "C1 (over-maintained)", "C2 (under-maintained)"]

eces = plot_multiclass_calibration(
    P, y,
    class_names=class_names,
    n_bins=10,
    title="Phase-BNN calibration (test set): class-wise reliability diagrams"
)

print("Class-wise ECE:", eces)

# Optional: save as publication/thesis figure
# plt.savefig("phase_bnn_calibration_test.pdf", bbox_inches="tight", dpi=300)
```




```
Class-wise ECE: {'B (balanced)': 0.0594821017684261, 'C1 (over-maintained)':
0.032655757347060124, 'C2 (under-maintained)': 0.05667526554276621}
```