POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Validation of Stochastic Hybrid Digital Twins

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Aleksa Mirkovic**

Student ID: 10972561
Advisor: Prof. Rossi Matteo Giovanni
Academic Year: 2025-26

Abstract

This thesis presents a comprehensive validation and verification framework for Stochastic Hybrid Automata (SHA) models of an industrial decanter centrifuge, building upon previous work that developed SHA-based digital twins using data from GR3N. While the foundational thesis demonstrated the feasibility of learning SHA models for bearing temperature estimation, this work addresses the critical question: are these learned models reliable enough for industrial deployment?

The research employs a dual approach combining formal verification and empirical validation. Formal verification utilizes UPPAAL's Statistical Model Checking (SMC) engine to verify that learned SHA models execute correctly, maintain temporal consistency, and satisfy bounded operational constraints. Empirical validation compares simulated bearing temperature trajectories against real sensor measurements across 17 operational traces, quantifying model accuracy through statistical metrics and visual analysis.

Results demonstrate that learned SHA models capture bearing thermal behavior during stable operational phases, achieving temperature predictions within quantifiable confidence intervals. However, validation reveals performance degradation during rapid state transitions, highlighting practical limitations for predictive maintenance applications. Statistical analysis across multiple traces provides objective metrics for model reliability assessment.

Key contributions include an integrated verification and validation framework combining formal methods with empirical testing, robust preprocessing pipelines handling real-world data inconsistencies, full-trace validation methodology ensuring comprehensive model assessment. This work establishes methodological foundations for deploying data-driven hybrid models in Industry 4.0 predictive maintenance systems with measurable confidence in their reliability.

Keywords: Stochastic Hybrid Automata, Statistical Model Checking, UPPAAL, model validation, centrifuge decanter, bearing temperature prediction

Abstract in lingua italiana

Questa tesi presenta un framework completo di validazione e verifica per modelli di automi ibridi stocastici (SHA) di una centrifuga decanter industriale, basandosi su lavori precedenti che hanno sviluppato gemelli digitali basati su SHA utilizzando i dati di GR3N. Mentre la tesi di base ha dimostrato la fattibilità dell'apprendimento di modelli SHA per la stima della temperatura dei cuscinetti, questo lavoro affronta la domanda critica: questi modelli appresi sono sufficientemente affidabili per l'impiego industriale?

La ricerca impiega un duplice approccio che combina verifica formale e validazione empirica. La verifica formale utilizza il motore di Statistical Model Checking (SMC) di UPPAAL per verificare che i modelli SHA appresi vengano eseguiti correttamente, mantengano la coerenza temporale e soddisfino vincoli operativi limitati. La validazione empirica confronta le traiettorie simulate della temperatura dei cuscinetti con le misurazioni reali dei sensori su 17 tracce operative, quantificando l'accuratezza del modello attraverso metriche statistiche e analisi visiva.

I risultati dimostrano che i modelli SHA appresi catturano il comportamento termico dei cuscinetti durante le fasi operative stabili, ottenendo previsioni di temperatura entro intervalli di confidenza quantificabili. Tuttavia, la convalida rivela un degrado delle prestazioni durante rapide transizioni di stato, evidenziando i limiti pratici delle applicazioni di manutenzione predittiva. L'analisi statistica su più tracce fornisce metriche oggettive per la valutazione dell'affidabilità del modello.

I contributi chiave includono un framework integrato di verifica e convalida che combina metodi formali con test empirici, robuste pipeline di pre-elaborazione che gestiscono le incoerenze dei dati reali e una metodologia di convalida full-trace che garantisce una valutazione completa del modello. Questo lavoro stabilisce le basi metodologiche per l'implementazione di modelli ibridi basati sui dati nei sistemi di manutenzione predittiva dell'Industria 4.0 con una misurabile affidabilità.

Parole chiave: Automi ibridi stocastici, controllo statistico del modello, UPPAAL, convalida del modello, decanter centrifugo, previsione della temperatura del cuscinetto

Contents

1 | Introduction

The decanter centrifuge is a critical piece of industrial equipment used in continuous solid-liquid separation processes, widely deployed in wastewater treatment, chemical processing, and recycling industries. The GR3N decanter centrifuge, operated by GR3N S.r.l., is specifically designed for plastic waste recycling, separating polymer slurries through high-speed centrifugal force. The machine operates through a rotating horizontal bowl housing an internal screw conveyor that rotates at a differential speed, continuously transporting separated solids while liquid exits through overflow ports.

A critical operational challenge in decanter centrifuges is bearing temperature management. The high-speed rotation and mechanical loads generate significant heat in the bearing assemblies, particularly in the reducer bearings that support the differential drive mechanism. Excessive bearing temperatures can lead to lubrication breakdown, accelerated wear, and ultimately catastrophic failure, resulting in costly downtime and maintenance. Therefore, accurate prediction and monitoring of bearing temperature is essential for preventive maintenance and operational reliability. This work focuses on the validation and verification of learned Stochastic Hybrid Automata (SHA) models that predict bearing temperature behavior based on operational parameters. The methodology employs formal verification techniques using the UPPAAL Statistical Model Checker to validate temperature predictions against real sensor measurements from the GR3N decanter. The validation process compares simulated bearing temperature trajectories—generated by learned SHA models—with actual thermocouple readings collected during normal operation, quantifying model accuracy and identifying operational regimes where predictions are reliable.

The validation framework processes real operational traces containing timestamped measurements of working status (motor on/off), current absorption (amperes), and bearing temperature (°C). These traces are systematically compared against UPPAAL simulations of the learned SHA models, which capture both discrete operational states (working/idle, different power absorption ranges) and continuous thermal dynamics (temperature evolution over time). The comparison yields quantitative metrics including temperature prediction error, mean absolute error (MAE), and root mean square error (RMSE), pro-

viding objective assessment of model reliability for deployment in industrial monitoring systems.

1.1. Context and Motivation

This thesis is an extension of prior work on learning Stochastic Hybrid Automata (SHA) models for the GR3N decanter centrifuge [11]. The goal of this work is to validate learned SHA models against independent operational traces and to better understand how the overall methodology behaves in practice. In particular, the thesis focuses on: (i) assessing generalization beyond the learning dataset, (ii) identifying failure modes and trace-dependent degradation patterns, and (iii) clarifying which parts of the pipeline (data preprocessing, model structure, parameterization, and simulation/SMC settings) drive prediction performance. This validation-first perspective supports industrial deployment by providing quantitative evidence of reliability and by making model limitations explicit.

The motivation for this validation-focused thesis stems from industrial necessity: GR3N requires confidence in predictive models before integrating them into operational decision-making processes. A model that performs well on training data but fails on new operational scenarios is unsuitable for predictive maintenance applications.

1.2. Limitations

The primary limitations encountered in this validation study stemmed from the quality and characteristics of the available learned SHA models. Not all models produced by the learning framework demonstrated acceptable predictive performance when subjected to rigorous validation testing. Several fundamental challenges were identified:

- **Model Determinism and Reproducibility:** A significant subset of the learned SHA models exhibited non-deterministic behavior during validation. When subjected to identical input traces and initial conditions, these models produced varying temperature predictions across multiple UPPAAL simulation runs. This stochastic variability, while inherent to the SHA formalism's use of probabilistic distributions, exceeded acceptable bounds in certain models, making their predictions unreliable for deterministic maintenance decision-making. The non-determinism arose from overly broad probability distributions learned from limited or noisy training data, resulting in high variance in simulated temperature trajectories.
- **Prediction Accuracy Variability:** Validation revealed substantial performance heterogeneity across the model inventory. Some learned SHA models achieved

excellent temperature prediction accuracy with mean absolute errors below 5°C across complete operational cycles, demonstrating strong capture of thermal dynamics. However, other models exhibited poor performance with prediction errors exceeding 50°C, failing to track actual bearing temperature trends even during steady-state operation. This performance variability highlighted sensitivity to training data quality, hyperparameter selection during learning, and the complexity of the learned automaton structure.

- **Operational Regime Specificity:** Even well-performing models showed degraded accuracy during specific operational conditions. Rapid transitions between working and idle states, sudden changes in power absorption, or operation near thermal limits often resulted in temporary prediction failures. This limitation suggests that while models captured general thermal behavior, they struggled with transient dynamics or operating conditions underrepresented in training data.

The validation dataset comprised models from two sources: models independently developed and trained as part of this thesis using updated training procedures and hyperparameter tuning, and models adopted directly from Carlos Fernández Carrera's original work to enable comparative analysis. This dual-source approach enabled assessment of whether validation outcomes were primarily influenced by model architecture versus training methodology. However, the mixed provenance required careful documentation and metadata management to maintain traceability of validation results to specific model generation procedures.

An additional limitation was the temporal coverage of available validation traces. While 17 operational traces spanning diverse operating conditions were utilized, certain edge cases (emergency shutdowns, extreme ambient temperatures, maintenance events) were underrepresented. Consequently, validation results characterize model performance under normal operational variability but do not comprehensively assess behavior under all possible fault or extreme conditions.

1.3. Objectives

The primary objective of this thesis is to validate and verify learned Stochastic Hybrid Automata models for bearing temperature prediction in the GR3N decanter centrifuge, establishing their reliability and identifying operational regimes where predictions are trustworthy. Specifically, this work aims to:

1. **Develop a Comprehensive Validation Framework:** Establish a systematic methodology for comparing SHA model predictions against real bearing temperature measurements using Statistical Model Checking with the UPPAAL tool. This framework must process industrial operational traces, execute stochastic simulations of learned automata, and quantitatively assess prediction accuracy through standardized metrics (temperature error percentage, MAE, RMSE, confidence intervals).

2. **Perform Formal Verification of Model Correctness:** Utilize UPPAAL's model checking capabilities to formally verify that learned SHA models execute correctly, maintain temporal consistency, satisfy bounded operational constraints, and can successfully process real operational event sequences without deadlocks or specification violations. This verification phase ensures that models are structurally sound before empirical validation.

3. **Quantify Prediction Accuracy Across Operational Conditions:** Systematically evaluate model performance across 17 diverse operational traces representing different working patterns, power absorption profiles, and ambient conditions. This comprehensive testing identifies which models achieve reliable predictions and under what operational circumstances accuracy degrades.

4. **Implement Full-Trace Validation Methodology:** Extend traditional validation approaches by developing a full-trace comparison framework where every timestamp in operational CSV files generates validation events. This high-fidelity approach overcomes limitations of sparse event-based validation, enabling precise comparison between continuous real temperature evolution and simulated predictions across complete operational cycles.

5. **Provide Industrial Decision-Making Guidance:** Generate actionable insights for GR3N regarding which learned models are suitable for integration into operational monitoring systems and which require further development. Document model performance characteristics transparently, including both successes and failures, to support informed deployment decisions.

The ultimate goal is to bridge the gap between theoretical model learning and practical industrial deployment, establishing whether Stochastic Hybrid Automata—learned from operational data—can serve as reliable digital twins for bearing temperature monitoring and predictive maintenance in real decanter centrifuge operations. Success in this objective would validate the broader applicability of SHA learning frameworks in Industry 4.0 contexts, demonstrating that formal methods and statistical model checking can provide the verification rigor necessary for trustworthy deployment of machine-learned models in safety-critical industrial systems.

2 | State of the Art

The validation of hybrid systems has undergone a fundamental transformation over four decades, evolving from simulation-based testing to formal verification and ultimately to statistical approaches capable of handling industrial-scale complexity.Understanding this evolution is essential to appreciate why modern approaches—particularly the integration of learned SHA models with statistical model checking—represent the optimal validation paradigm for complex cyber-physical systems like industrial decanter centrifuges.

2.1. Hybrid-system validation in practice

Hybrid-system validation is constrained by two practical facts: continuous dynamics make exhaustive state exploration infeasible, while industrial deployment requires quantitative evidence that a model is reliable under realistic operating conditions. Modern validation therefore combines (i) simulation-based evaluation on representative traces, (ii) statistical guarantees obtained through sampling-based techniques such as Statistical Model Checking (SMC), and (iii) systematic reporting of when and why performance degrades.

2.1.1. Brief historical perspective

The foundations of modern validation trace back to timed automata, introduced by Alur and Dill, which enabled algorithmic reasoning about real-time behavior [1]. Extending these ideas to hybrid systems quickly revealed fundamental computational limits, with reachability becoming undecidable for general hybrid automata [15, 16]. As a result, practical workflows increasingly adopted statistical approaches that trade exhaustive guarantees for scalable, confidence-bounded evidence, leading to Statistical Model Checking methods [14, 30] and their integration into tools such as UPPAAL-SMC for stochastic hybrid systems [9].

2.1.2. Brief historical perspective

The foundations of modern validation trace back to timed automata, introduced by Alur and Dill, which enabled algorithmic reasoning about real-time behavior [1]. Extending these ideas to hybrid systems quickly revealed fundamental computational limits, with reachability becoming undecidable for general hybrid automata [15, 16]. As a result, practical workflows increasingly adopted statistical approaches that trade exhaustive guarantees for scalable, confidence-bounded evidence, leading to Statistical Model Checking methods [14, 30] and their integration into tools such as UPPAAL-SMC for stochastic hybrid systems [9].

2.1.3. Learning for industrial applications

In industrial settings, learning hybrid models from data can reduce manual modeling effort and enable faster iteration when the plant, sensors, or operating regimes change. However, learned models are only useful if they are validated on independent traces and accompanied by transparent metrics (accuracy, robustness, and uncertainty). This motivates a workflow in which model learning and model validation are treated as complementary stages: learning produces candidate models, while validation determines whether they generalize well enough to support monitoring and predictive maintenance decisions. The industrial motivation for learning models from data grew alongside system identification and predictive modeling in control engineering, where data-driven approaches were adopted to complement or replace first-principles models when the underlying physics was complex or partially unknown [19]. More recently, the Industry 4.0 and digital-twin paradigms have further accelerated interest in learned models as deployable assets that can be retrained as new plant data becomes available [27].

The industrial motivation for learning models from data grew alongside system identification and predictive modeling in control engineering, where data-driven approaches were adopted to complement or replace first-principles models when the underlying physics was complex or partially unknown [19]. More recently, the Industry 4.0 and digital-twin paradigms have further accelerated interest in learned models as deployable assets that can be retrained as new plant data becomes available [27].

2.2. UPPAAL: Technical Foundations and Capabilities

UPPAAL represents one of the most successful formal verification tools, bridging the gap between theoretical foundations and practical industrial application. This section provides a comprehensive technical overview of UPPAAL's architecture, modeling formalism, verification algorithms, and capabilities.

2.2.1. What UPPAAL Is

UPPAAL is an integrated tool environment for modeling, validation, simulation, and verification of real-time systems modeled as networks of timed automata extended with data types [4]. Officially defined by Bengtsson and Yi: "UPPAAL is a toolbox for verification of real-time systems represented by (a network of) timed automata extended with integer variables, structured data types, and channel synchronization."

The tool is designed to model real-time controllers and embedded systems, communication protocols with timing constraints, multimedia applications with Quality of Service requirements, scheduling problems with deadlines, and cyber-physical systems combining discrete control with continuous dynamics. Its core modeling language consists of Networks of Timed Automata (NTA) using CCS-style parallel composition with hand-shake synchronization, extended with bounded integers, arrays, and structured data types using C-like syntax.

2.2.2. Historical Development

UPPAAL was first released in 1995 as a joint development between Uppsala University (Sweden) and Aalborg University (Denmark)—the name combines UPPsala and AALborg. The three founding researchers were Kim G. Larsen (Aalborg University), Paul Pettersson (Uppsala University), and Wang Yi (Uppsala University) [18]. Their collaboration created a tool that combined the theoretical rigor of academic research with the practical usability needed for industrial adoption.

Johan Bengtsson developed the critical Difference Bound Matrix (DBM) algorithms during his PhD work, documented in his 2002 thesis [3]. Alexandre David contributed hierarchical modeling extensions and improved verification algorithms in his 2003 PhD thesis [8]. Version 4.0 (2006) represented a major redesign with improved algorithms and user interface, documented by Behrmann et al. at QEST'06 [2].

UPPAAL started as an academic research tool at BRICS (Basic Research in Computer Science) but has been successfully applied to over 17 industrial case studies since 1995. Notable applications include verification of Philips audio protocols [5], Bang & Olufsen power-down protocols, Mecel Bentley gearbox controllers, commercial field bus protocols, and collision avoidance protocols for autonomous vehicles. This industrial success demonstrates UPPAAL's practical value beyond academic research.

2.2.3. Timed Automata: Theoretical Foundations

UPPAAL implements timed automata based on Alur and Dill's foundational theory [1]. A timed automaton is formally defined as $\mathcal{A} = \langle N, l_0, E, I \rangle$ where:

- N is a finite set of locations
- $l_0 \in N$ is the initial location
- $E \subseteq N \times \mathcal{B}(C) \times 2^C \times N$ is the set of edges, where each edge specifies a guard condition, a set of clocks to reset, and a target location
- $I : N \to \mathcal{B}(C)$ is an invariant function assigning timing constraints to locations

Clock constraints $\mathcal{B}(C)$ are conjunctive formulas of atomic constraints of the form $x \sim n$ or $x - y \sim n$, where $x, y \in C$ are clocks taking real values, $n \in \mathbb{N}$, and $\sim \in \{\leq, <, =, >, \geq\}$. Guards enable transitions when clock values satisfy specified bounds, while invariants restrict time progress within locations using downwards-closed constraints (upper bounds on clocks).

Operational Semantics

The operational semantics defines the behavior of timed automata through two types of transitions. A configuration is a pair $\langle l, u \rangle$ where $l \in N$ is the current location and $u \in \mathbb{R}_{\geq 0}^{|C|}$ is the clock valuation.

Delay transitions model time passage: $\langle l, u \rangle \xrightarrow{d} \langle l, u + d \rangle$ where $d \in \mathbb{R}_{\geq 0}$ and the invariant $I(l)$ remains satisfied throughout $[u, u + d]$. This represents the system remaining in location l while time advances by d time units.

Action transitions model discrete state changes: $\langle l, u \rangle \xrightarrow{a} \langle l', u' \rangle$ when there exists an edge $(l, g, r, l') \in E$ such that the guard g is satisfied by u, the action label is a, the invariant $I(l')$ is satisfied by u', and u' is obtained from u by resetting clocks in r to zero while leaving others unchanged.

Networks of Timed Automata

UPPAAL extends the basic timed automata model to networks composed using parallel composition with synchronization. The composition $\mathcal{A}_1|\mathcal{A}_2|\ldots|\mathcal{A}_n$ defines a system where automata can evolve independently or synchronize on shared channels. Binary hand-shake synchronization uses complementary labels: $a!$ (output) and $a?$ (input). A synchronized transition occurs when one automaton performs $a!$ simultaneously with another performing $a?$.

Broadcast channels enable one-to-many synchronization, where one sender can trigger multiple receivers simultaneously. This is essential for modeling systems where events need to be communicated to multiple components, such as alarm signals or mode changes in industrial systems.

2.2.4. Model Checking Algorithms

At its core, UPPAAL checks properties by exploring which symbolic states are reachable. Because clocks take real values, UPPAAL cannot enumerate states explicitly; instead, it represents sets of clock valuations symbolically as *zones*.

A zone is a convex set defined by difference constraints such as $x - y \leq c$. Internally, zones are stored as Difference Bound Matrices (DBMs), which support efficient operations for (i) letting time elapse, (ii) intersecting with guards, and (iii) applying resets.

During reachability analysis, UPPAAL maintains a frontier of symbolic states (location vectors plus a zone) and repeatedly computes successors by time-delays and discrete transitions. To ensure termination, zones are normalized w.r.t. the maximum constants that appear in the model; newly discovered zones are discarded when they are included in an already explored zone at the same control location.

2.2.5. Query Language and Verification Capabilities

UPPAAL uses a simplified subset of TCTL (Timed Computation Tree Logic) for property specification. The query language supports four main property types:

Reachability Properties

The query `E<>` φ asks "Is it possible to reach a state satisfying φ?" This is the most fundamental verification query, used to check if certain conditions can ever occur. Examples include `E<> error_state` to verify if an error is reachable, or `E<> (x >= 10 and y < 5)`

to check if specific clock constraints can be satisfied simultaneously.

Safety Properties (Invariants)

The query `A[]` φ asks "Does φ hold in all reachable states?" This verifies safety properties—conditions that must always hold. Examples include `A[] not deadlock` to verify absence of deadlocks, `A[] (critical1 + critical2 <= 1)` to verify mutual exclusion, and `A[] (temperature <= 100)` to verify that a physical constraint is never violated.

Liveness Properties

The query `A<>` φ asks "Will φ eventually be satisfied on all paths?" This verifies liveness properties—guarantee that good things eventually happen. Examples include `A<> request imply A<> grant` to verify that all requests are eventually granted.

Leads-to Properties

The query φ `->` ψ asks "Whenever φ holds, will ψ eventually hold?" This is syntactic sugar for `A[]` (φ `imply A<>` ψ), verifying that one condition eventually leads to another. This is essential for verifying response properties in reactive systems.

State formulas φ can reference process locations using the syntax `process.location`, integer expressions, clock constraints, and the special predicate `deadlock`. A key limitation is that nested temporal operators are not supported; complex properties requiring nested quantifiers must be encoded using observer automata.

2.2.6. Key Features and Tool Variants

UPPAAL provides a comprehensive development environment with three main components: a graphical system editor for drawing automata, a symbolic simulator for interactive trace exploration, and an efficient verification engine implemented in C++. The simulator enables users to explore system behavior step-by-step, invaluable for understanding why verification fails or for generating test cases.

Advanced modeling features include:

- **Templates with parameters**: Reusable automaton definitions that can be instantiated with different parameters, enabling modular system construction.
- **Urgent locations**: Locations where time cannot pass, forcing immediate transition. Used to model atomic sequences of transitions.

- **Committed locations**: Stronger than urgent—when any automaton is in a committed location, the next transition must involve that automaton. This models prioritized execution.
- **Broadcast channels**: One-to-many synchronization for event propagation.
- **Bounded integer variables and arrays**: Enable modeling of data-dependent systems beyond pure timing properties.

2.3. UPPAAL SMC: Statistical Model Checking for Stochastic Systems

While exhaustive model checking provides absolute guarantees for systems within its computational reach, many industrial systems exceed these boundaries due to state-space explosion and undecidability. UPPAAL SMC addresses this limitation through statistical model checking—using simulation with rigorous statistical inference to provide probabilistic guarantees about system properties.

2.3.1. Introduction and Development

UPPAAL SMC emerged as a major extension around 2012, with the foundational paper by Bulychev, David, Larsen et al. "UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata" presented at QAPL 2012 [9]. The key developers include Kim G. Larsen and Alexandre David (Aalborg University), Axel Legay (INRIA Rennes), and Marius Mikučionis. The comprehensive tutorial by David et al. published in the International Journal on Software Tools for Technology Transfer in 2015 [9] documents the mature system.

Statistical Model Checking avoids exhaustive state-space exploration by conducting simulations, monitoring them against properties, and using statistical algorithms—sequential hypothesis testing and Monte Carlo estimation—to determine property satisfaction with specified confidence levels. The fundamental insight is that it may not be necessary to obtain absolutely accurate probability estimates; instead, it suffices to infer with high confidence whether a probability exceeds or falls below a threshold.

2.3.2. Technical Approach and Algorithms

The core technique uses Monte Carlo simulation where each simulation run is encoded as a Bernoulli random variable that takes value 1 if the property is satisfied and 0 otherwise.

Statistical inference then estimates probabilities based on these samples.

Probability Estimation

The query `Pr[bound](<>` ψ`)` estimates the probability of reaching a state satisfying ψ within the time bound. UPPAAL SMC uses the Clopper-Pearson "exact" method for computing confidence intervals [7]. Given m successes in n simulations, the method computes confidence intervals using the beta distribution, providing conservative bounds that guarantee the true probability lies within the interval with specified confidence (typically 95% or 99%).

The confidence interval approach allows users to specify desired precision: a narrower interval requires more simulations but provides more precise probability estimates. This trade-off between computational cost and precision is fundamental to statistical model checking.

Hypothesis Testing

The query `Pr[bound](`ψ`) >=` p_0 tests whether the probability of ψ exceeds threshold p_0. UPPAAL SMC uses Wald's Sequential Probability Ratio Test (SPRT) [29], which tests the null hypothesis $H_0 : p \geq p_0$ against the alternative $H_1 : p \leq p_1$ (where $p_1 < p_0$) with specified Type I error probability α (false positive rate) and Type II error probability β (false negative rate).

The SPRT computes a likelihood ratio after each simulation and makes one of three decisions: accept H_0 (conclude property holds), accept H_1 (conclude property fails), or continue sampling. This sequential approach typically requires far fewer samples than fixed-sample-size tests because it terminates as soon as sufficient evidence accumulates.

The power of SPRT lies in its efficiency: when the true probability is far from the threshold, only a few samples are needed to reach a confident decision. When the probability is close to the threshold, more samples are required, automatically adapting the computational effort to the difficulty of the problem.

Probability Comparison

The query `Pr[bound1](`ψ_1`) >= Pr[bound2](`ψ_2`)` compares the probabilities of two properties. This is implemented using extended Wald testing for comparing two Bernoulli populations, essential for optimization and design-space exploration where we need to determine which of two system configurations performs better.

2.3.3. Stochastic Semantics and Hybrid System Support

UPPAAL SMC supports Networks of Stochastic Timed Automata (NSTA) where components must be input-enabled, deterministic, and non-Zeno. Stochasticity is introduced through probabilistic delay distributions and random choices.

Delay Distributions

Delays in UPPAAL SMC follow two types of distributions:

Uniform distribution: When an invariant constrains a clock $x \leq c$, the delay before leaving the location is uniformly distributed over $[0, c]$ if no other transitions are enabled. This models situations where any time within the allowed interval is equally likely.

Exponential distribution: When a location has no invariant restricting delays, the system can remain there indefinitely. UPPAAL SMC models this using exponential distributions with user-defined rates, specified using the `rate` keyword. Exponential distributions have the memoryless property, making them mathematically tractable for continuous-time stochastic processes.

Probabilistic Choice

When multiple transitions are enabled, UPPAAL SMC resolves the non-determinism probabilistically using race semantics: each enabled transition has an associated exponentially distributed delay, and the transition with the smallest sampled delay fires. Different rates lead to different probabilities of each transition being selected, enabling modeling of probabilistic failures, varying success rates of operations, and unreliable communication channels.

2.3.4. Query Language Extensions

UPPAAL SMC extends the query language with Weighted Metric Interval Temporal Logic (MITL), supporting bounded temporal operators:

- `<>[t1,t2]` φ: Eventually within time interval $[t_1, t_2]$, property φ holds
- `[][t1,t2]` φ: Always within time interval $[t_1, t_2]$, property φ holds
- Until operators with bounded time horizons

Expected value queries compute expectations over simulations: `E[bound; N](max: expr)` computes the expected maximum value of expression `expr` over the time bound, based on

N simulation runs. This enables verification of performance metrics like average response time, expected energy consumption, and mean time between failures.

Simulation queries generate execution traces: `simulate N [<=bound] {E1,..,Ek}` runs N simulations up to the time bound and plots expressions E_1 through E_k. This visualization is invaluable for understanding system behavior and debugging models.

3 | The Decanter Centrifuge System

This chapter introduces the industrial decanter centrifuge used as the real-world case study for learning (previous thesis) and validating (this thesis) Stochastic Hybrid Automata models. The aim is to provide just enough system context to interpret the sensor traces, the hybrid modes, and the validation results presented later.

3.1. Introduction to the System

The system under study is an industrial decanter centrifuge (GBB Decanter CD25E-Z1) operated by GR3N in recycling and recovery applications [11]. GR3N provided operational traces within the Auto-Twin Project; these traces were previously used to learn SHA models and are used here to validate their predictive accuracy.

A key motivation for data-driven modeling is that the embedded controller (DPC) is reactive: it executes control logic based on current measurements, but it does not provide simulation or probabilistic prediction. Learned SHA models, together with UPPAAL SMC, enable forecasting and risk-oriented questions (e.g., likelihood of overheating or abnormal vibration over a time horizon).

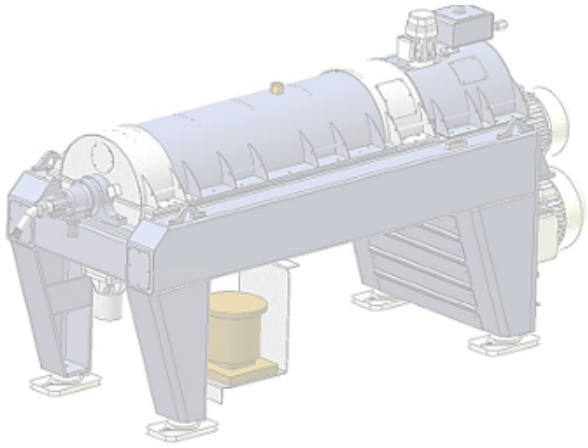

Figure 3.1: Industrial decanter centrifuge used as the reference system for learning and validation.

3.2. Physical Principles of Operation

To understand the validation challenges this system presents, one must first understand how a decanter centrifuge achieves separation. The fundamental principle is elegant: use centrifugal force to accelerate the natural settling that gravity would eventually produce.

Consider what happens when solid particles are suspended in a liquid. Under Earth's gravity, denser particles slowly settle downward—this is sedimentation. The settling rate depends on the density difference between particles and liquid, particle size, and the strength of the gravitational field. For fine particles or small density differences, gravitational settling can take hours or days, making it impractical for industrial processes.

A centrifuge dramatically accelerates this process by substituting centrifugal force for gravity. When a mixture rotates at high speed in a cylindrical bowl, particles experience forces hundreds or thousands of times stronger than Earth's gravity. What would take hours under gravity occurs in seconds under centrifugal force. The challenge lies in continuously feeding new mixture while continuously removing separated solids and liquids—this is what distinguishes a decanter centrifuge from simpler batch centrifuges.

The CD25E-Z1 achieves continuous operation through a clever mechanical design. The stainless steel bowl rotates at high speeds (the exact speed is adjustable and represents one control parameter). Inside this rotating bowl sits a helical screw conveyor—imagine an auger or Archimedes screw—that rotates at nearly the same speed as the bowl, but with a small differential speed. This differential speed is critical to understanding the machine's operation.

From the bowl's reference frame, the screw appears to rotate slowly. This slow relative rotation allows the screw flights (the helical blades) to continuously scrape settled solids off the bowl wall and push them toward one end for discharge. Meanwhile, clarified liquid flows toward the opposite end and exits through adjustable weirs. The adjustable weirs control the liquid depth (called the pond depth), which influences residence time and separation efficiency—deeper ponds provide more settling time but reduce the effective centrifugal force on particles near the liquid surface.

This mechanical arrangement creates a continuous-flow separation system, but it also creates complex dynamics. The rotating masses (bowl and screw) possess significant kinetic energy. Changes in feed rate, differential speed, or material properties affect forces throughout the system. Bearings must support these rotating assemblies while managing generated heat. The electrical drives must provide precise torque control while responding to load variations. These interconnected physical processes—mechanical, thermal, electrical, and fluid dynamical—make the decanter centrifuge a genuinely hybrid system.

3.2.1. Sensor Instrumentation

The decanter's sensor suite provides comprehensive monitoring of system state. Table 3.1 summarizes the available measurements, categorized into analog (continuous-valued) and digital (binary or discrete) signals.

Table 3.1: Decanter centrifuge sensor measurements (summary)

Category	Signals (examples)
Analog variables	Coppia (Torque) Differenziale (Differential speed) Assorbimento (Current consumption) LivelloVibrazioni (Vibration level) TCuscinettiRiduttore (Gearbox bearing temperature) TCuscinettiAlimentazione (Feed bearing temperature)
Digital variables	MarciaDecanter (Decanter running) MarciaPmpFango (Slurry pump running) CicloAttivo (Active cycle) AllarmeVibrazioni (Vibration alarm) AllarmeTCuscinetti (Bearing temperature alarm) PresenzaAllarme (Alarm present)

The analog variables provide continuous measurements of physical quantities. Torque measurements indicate the mechanical load on the drives, reflecting the resistance to

rotation from the material being processed. Higher torque generally indicates heavier or more difficult-to-process material. Differential speed confirms that the screw is rotating at the commanded differential relative to the bowl. Current consumption (Assorbimento) measures electrical power drawn by the drives, which correlates with mechanical load but also reflects motor efficiency and electrical system characteristics.

Vibration level monitoring detects abnormal mechanical behavior. Normal operation produces a baseline vibration signature, while problems like imbalanced loading, bearing wear, or mechanical looseness produce characteristic vibration patterns. The two bearing temperature sensors—one for the gearbox bearings and one for the feed-end bearings—provide direct thermal monitoring of these critical components.

The digital variables encode discrete states and events. Operational state variables like MarciaDecanter and MarciaPmpFango indicate which components are running. The CicloAttivo signal indicates whether the machine is executing a production cycle. Various alarm signals flag out-of-range conditions: AllarmeVibrazioni trips when vibration exceeds safe levels, AllarmeTCuscinetti signals dangerous bearing temperatures, and the master PresenzaAllarme indicates that some alarm condition exists.

This sensor suite generates the data from which SHA models are learned and against which predictions are validated. The analog variables provide the continuous trajectories that reveal system dynamics within operational modes. The digital variables mark the discrete events—startups, shutdowns, mode changes, alarms—that trigger transitions between modes. Together, they capture the hybrid behavior that SHA models must represent.

3.3. Operational Workflow

For modeling and validation, it is useful to view the machine as executing a small number of recurring phases. These phases naturally correspond to discrete modes in a hybrid model, while temperatures, currents, and vibrations evolve continuously within each mode.

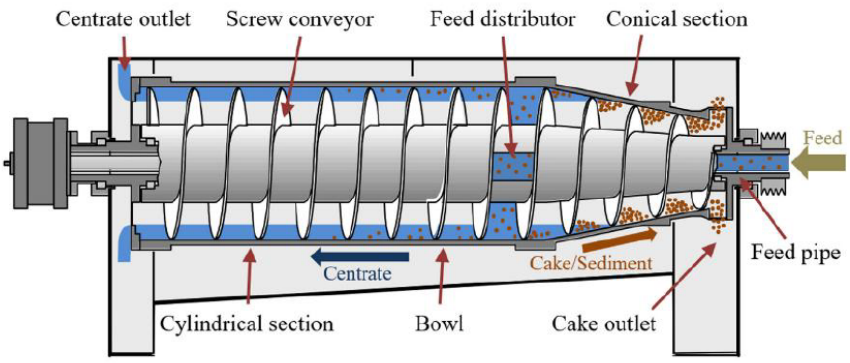

Figure 3.2: Schematic view of a decanter centrifuge (bowl and internal screw conveyor).

3.3.1. Startup Phase

Startup moves the system from rest to nominal operation: bowl acceleration, screw engagement to reach the differential speed, and feed activation. This phase is characterized by short transients (often visible as peaks in current/torque) and an initial rise of bearing temperatures.

3.3.2. Normal Operation Phase

During normal operation, sensor values fluctuate around operating points due to changes in processed material and external disturbances. Bearing temperatures approach a quasi-steady regime that depends on mechanical load and ambient conditions.

3.3.3. Shutdown Phase

Shutdown typically stops the feed first, clears residual solids, and then decelerates the bowl to rest. Temperatures decrease slowly due to thermal inertia, providing informative cooldown segments in the traces.

Overall, full traces repeatedly traverse idle $\rightarrow$ startup $\rightarrow$ operation $\rightarrow$ shutdown $\rightarrow$ idle. This cycle is a useful validation target because it tests both discrete transitions and continuous thermal dynamics.

3.4. Available Data Characteristics

The operational data provided by GR3N forms the foundation for both model learning and validation. Understanding the data characteristics clarifies what can and cannot be

learned from available measurements.

Fernández Carrera reports that the data was shared in two batches covering extended time periods but with discontinuous coverage [11]. For example, the first batch spans 45 days for digital signals and 64 days for continuous variables, but actual data availability is much sparser. Only 10 of the 45 days contain digital signal data, and 49 of 64 days contain analog measurements. This sparsity reflects the intermittent operational schedule—the decanter operates only when production batches require processing, not continuously.

Interestingly, analog sensors like bearing temperature monitors continue recording even when the decanter is idle. This produces temperature decay traces during shutdown and idle periods, which provide valuable information about thermal time constants and heat transfer characteristics. In contrast, digital state variables only change during active operations, so their coverage corresponds to actual running time.

The dataset contains abundant measurements during available periods. Sensor sampling rates are sufficient to capture dynamic behavior—temperature variations, vibration fluctuations, startup transients all appear clearly in the data. The challenge lies not in temporal resolution but in operational diversity. Robust model learning requires observing the system across a range of operating conditions, startup patterns, and material types. Limited operational diversity constrains the generalization capabilities of learned models.

A significant limitation that Fernández Carrera notes is the incomplete operational context [11]. The recorded data periods lack documentation about testing procedures, equipment maintenance activities, or external conditions that might influence behavior. Certain variables show unusual patterns that might reflect experimental testing, equipment problems, or exceptional feed materials, but without operational logs, these events cannot be definitively classified. This uncertainty affects model learning—should unusual events be treated as normal operational variability or excluded as anomalous?

Despite these limitations, the dataset provides sufficient information to learn SHA models and conduct meaningful validation. Multiple complete startup-operation-shutdown cycles appear in the data, enabling learning of mode structure and transition behavior. Extended steady-state periods provide statistics for characterizing stochastic variations. The availability of multiple sensor types enables validation across different physical quantities—if a model correctly predicts bearing temperatures, vibration levels, and power consumption simultaneously, confidence in model fidelity increases beyond what single-variable validation would provide.

3.5. Validation Challenges and Opportunities

The decanter centrifuge system presents both challenges and opportunities for validating learned SHA models. The challenges stem from system complexity, while the opportunities arise from rich instrumentation and genuine industrial relevance.

The primary technical challenge is that the "true" system model is unknown. Unlike validation scenarios where a system model can be constructed from first principles and validated against synthetic data from that model, industrial systems resist analytical modeling. One cannot write down differential equations that exactly describe bearing temperature evolution as functions of rotational speed, ambient temperature, bearing wear, lubrication conditions, and countless other factors. This means validation cannot compare learned models against ground truth—instead, validation must compare model predictions against observed system behavior and quantify the agreement.

The stochastic nature of the system compounds this challenge. Even in nominally identical operating conditions, sensor readings fluctuate due to measurement noise, environmental variations, and intrinsic process randomness. A deterministic model that exactly matched one operational trace would fail on another trace run under the "same" conditions. Validation must therefore assess statistical properties—do predicted distributions match observed distributions?—rather than point-wise exactness.

The hybrid dynamics introduce additional complexity. Different modes exhibit different dynamics with different time constants. Startup transients are rapid, steady-state variations are slower, thermal decay during shutdown is slowest. A model that accurately captures one phase might fail in another. Comprehensive validation requires testing across all operational phases.

However, the decanter also offers significant validation advantages. The rich sensor suite enables multi-variable validation. If a model correctly predicts both bearing temperatures (thermal dynamics) and vibration levels (mechanical dynamics) and power consumption (electrical dynamics), this provides stronger evidence of model fidelity than validating only one variable. The different physical processes impose independent constraints on the model—it becomes increasingly unlikely that a model could spuriously match multiple independent observables without genuinely capturing underlying system behavior.

The availability of complete operational traces—full startup-operation-shutdown cycles—enables trace-based validation where model predictions can be compared against reality over extended time periods. This is more rigorous than validating only steady-state behavior, because it tests both the continuous dynamics within modes and the discrete

transition behavior between modes.

The industrial context provides clear validation requirements. Predictive maintenance applications require knowing not just average behavior but extreme values and failure probabilities. Validation must therefore assess whether models correctly predict temperature extremes (for planning cooling interventions), vibration spikes (for bearing replacement scheduling), and rare alarm events (for reliability analysis). These requirements drive the validation methodology toward statistical model checking with probability estimation rather than simple trajectory matching.

Finally, the decanter represents a real-world system with genuine economic consequences. If validated models enable more efficient operation, reduced downtime, or earlier failure detection, the economic value justifies the modeling effort. This real-world relevance motivates investing in rigorous validation—approximate models with uncertain reliability might suffice for academic demonstrations, but industrial deployment demands quantified confidence bounds on model predictions.

3.6. Conclusion

The GBB Decanter CD25E-Z1 centrifuge exemplifies the class of industrial cyber-physical systems that benefit from data-driven hybrid modeling and rigorous validation. Its combination of continuous physical processes, discrete control logic, rich instrumentation, and operational complexity makes it simultaneously challenging to model and valuable to model well.

The system description provided in this chapter establishes the foundation for understanding the modeling and validation work presented in subsequent chapters. Readers now understand what physical processes the model must capture, what sensors provide the validation data, what operational sequences constitute realistic test scenarios, and what challenges validation must address.

The next chapters will detail how Stochastic Hybrid Automata are learned from operational data, how learned models are converted to UPPAAL-compatible formats, how statistical model checking enables rigorous validation with quantifiable confidence bounds, and what validation results reveal about model reliability across different operational conditions and prediction horizons. The decanter centrifuge serves as the concrete testbed where theoretical validation methodologies meet industrial reality.

4 | Stochastic Hybrid Automata: Theory and Learning

Stochastic Hybrid Automata (SHA) model systems that combine (i) discrete operating modes, (ii) continuous physical dynamics, and (iii) randomness. In this thesis the SHA models are not hand-crafted: they are learned from GR3N operational traces using the L*SHA (often written L-SHA) learning workflow introduced in the previous thesis, and then validated here using UPPAAL/UPPAAL SMC.

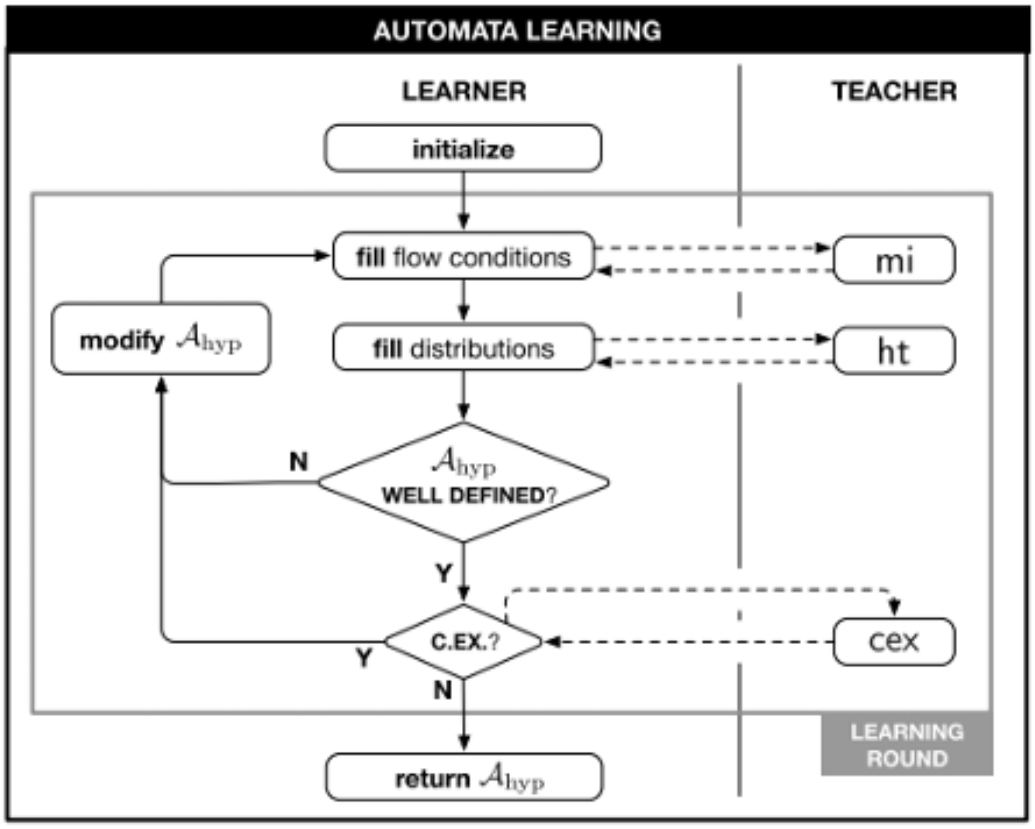

Figure 4.1: L*SHA learning workflow used to infer SHA models from industrial traces (previous thesis).

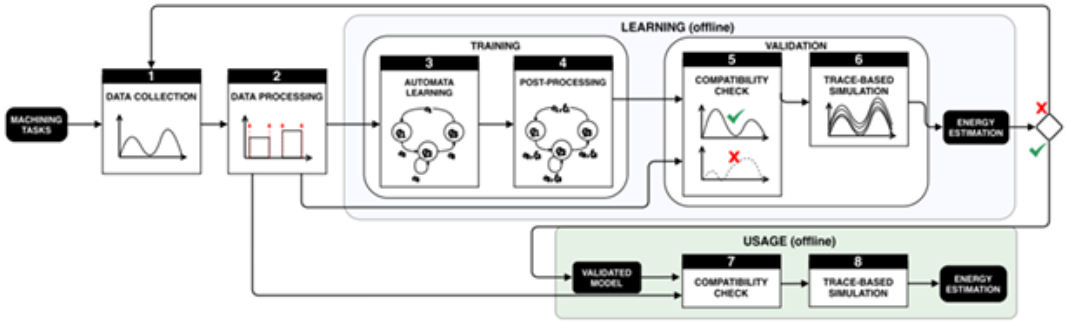

Figure 4.2: Overall methodology: learning (L*SHA) followed by formal/statistical validation in UPPAAL SMC.

4.1. Theoretical Foundations

A convenient formal view of a SHA is the tuple $\mathcal{H} = (Q, X, \text{Inv}, f, \sigma, G, R)$ [17], where Q is a finite set of modes and $X = \mathbb{R}^n$ is the continuous state space. Each mode has an invariant $\text{Inv}(q)$ restricting where the system may stay, continuous dynamics given by a drift term $f(q, \cdot)$, and a diffusion term $\sigma(q, \cdot)$ capturing stochastic disturbances. Discrete transitions are enabled by guards G and may update the continuous state through probabilistic resets R.

Within each discrete mode, the continuous evolution is often expressed as a stochastic differential equation (SDE):

$$dX(t) = f(q, X(t))dt + \sigma(q, X(t))dB_t. \tag{4.1}$$

Intuitively, f captures the average physical trend (e.g., heating/cooling rates), while the diffusion term captures unmodeled effects and measurement/actuation variability.

4.1.1. History and Key Researchers

The development of stochastic hybrid system theory spans four decades:

- **1984**: M.H.A. Davis introduces Piecewise Deterministic Markov Processes [10], establishing foundations for hybrid stochastic modeling.
- **1996**: Thomas A. Henzinger formalizes hybrid automata theory [15, 16], creating the deterministic foundation upon which stochastic extensions would build.
- **2000**: Hu, Lygeros, and Sastry at UC Berkeley provide the first formal definition of

Stochastic Hybrid Automata [17].

- **2003-2006**: Bujorianu and Lygeros develop the GSHS framework [6], unifying various stochastic hybrid formalisms.
- **2012**: UPPAAL-SMC enables practical verification of stochastic hybrid systems [9], making the theory accessible to engineers.

Key research groups include UC Berkeley (Shankar Sastry, Claire Tomlin) for early SHA theory and safety verification, ETH Zurich (John Lygeros) for GSHS theory and stochastic control, Aalborg University (Kim G. Larsen) for UPPAAL-SMC and practical verification, INRIA France (Axel Legay) for statistical model checking algorithms, and ISTA Austria (Thomas Henzinger) for foundational hybrid system theory.

4.2. SHA Learning Algorithms

Learning SHA models means inferring hybrid automaton structure from observational data. The learning problem decomposes into four sub-problems:

1. **Mode identification**: Discovering the discrete modes Q from continuous trajectories
2. **Dynamics learning**: Inferring continuous dynamics f and σ within each mode
3. **Guard inference**: Determining transition guards G that trigger mode switches
4. **Stochastic parameter estimation**: Estimating distributions for resets and transition rates

Clustering-Based Methods

Ferrari-Trecate, Muselli, Liberati, and Morari (2003) developed clustering-based methods for piecewise affine system identification [12]. Their approach uses K-means clustering in a lifted feature space to identify regions with distinct affine dynamics. The algorithm alternates between assigning data points to clusters (mode identification) and estimating affine dynamics parameters for each cluster using least-squares regression.

DBSCAN-based methods enable density-based mode discovery without requiring prior specification of mode count. These methods identify modes as high-density regions in the state space separated by low-density boundaries, naturally handling systems where the number of modes is unknown.

Algebraic and Geometric Methods

Vidal, Soatto, Ma, and Sastry (2003) developed polynomial factorization approaches for hybrid system identification [28]. Their Generalized Principal Component Analysis (GPCA) algorithm fits multiple linear subspaces to data by solving polynomial equations. Each subspace corresponds to a different mode's dynamics.

Ozay, Lagoa, and Sznaier extended these approaches with sum-of-squares relaxations for handling noisy data [23]. These methods provide theoretical guarantees about identification accuracy when noise levels are bounded.

Mixed-Integer Programming

Roll, Bemporad, and Ljung (2004) formulated piecewise affine system identification as mixed-integer optimization problems [25]. The binary variables indicate mode assignments, while continuous variables represent dynamics parameters. Modern MILP solvers can handle problems with thousands of data points, making this approach practical for many applications.

Neural Network Approaches

Neural Hybrid Automata (NHA) by Poli, Massaroli, Yamashita, Asama, and Park (NeurIPS 2021) [24] represents the cutting edge of deep learning for hybrid systems. NHA combines neural ordinary differential equations for continuous dynamics with discrete latent states for modes. Normalizing flows model inter-event time distributions, enabling the network to learn both when and how mode transitions occur. End-to-end training via gradient descent learns all components simultaneously from raw trajectory data.

Mining from Traces

Medhat, Ramachandran, Bonakdarpour, Kumar, and Fischmeister (EMSOFT 2015) [21] and Saberi, Azimi, and Bonakdarpour (ACM TECS 2022) [26] developed techniques for passive learning of hybrid automata from input/output traces. Their algorithms use dynamic time warping to measure trajectory similarity, clustering similar traces to identify modes, and inferring guards from the conditions under which mode transitions occur.

These trace-based methods are particularly relevant for industrial applications where direct state measurements may be unavailable but input/output behavior is observable. For centrifuge systems, operational logs provide rich trace data from which models can be learned.

4.2.1. Advantages of Learning Over Hand-Crafting

Data-driven model construction offers fundamental advantages over traditional manual modeling:

Capturing actual behavior: Learned models reflect how the system actually behaves rather than how engineers believe it should behave. This is critical for systems with complex nonlinearities, wear-dependent behavior, and poorly understood phenomena.

Automatic stochasticity incorporation: Real-world variability from manufacturing tolerances, environmental conditions, and material properties is automatically captured in the learned stochastic parameters rather than being ignored or crudely approximated.

Tractability for complex systems: Industrial systems may have dozens of operational modes. Manually identifying and modeling each mode is infeasible, but learning algorithms can automatically discover mode structure from data.

Reduced modeling errors: Human modeling introduces errors through incorrect assumptions, missed interactions, and simplified dynamics. Learning from data grounds the model in observed reality, though it introduces different challenges like overfitting and generalization.

For industrial centrifuge systems, these advantages are critical. The complex fluid dynamics, thermal behavior, and mechanical wear are extremely difficult to model from first principles. Parameter values like friction coefficients, heat transfer rates, and material properties vary between individual machines. Learning SHA models from operational data captures these system-specific characteristics that first-principles models cannot adequately represent.

4.2.2. The L*SHA Learning Algorithm

Stochastic Hybrid Automata combine discrete state machines (locations and transitions) with continuous dynamics (differential equations and probability distributions). Learning such automata automatically from sensor data requires solving two interrelated problems: discovering the discrete structure (how many locations? which transitions exist?) and identifying the continuous parameters (what distribution governs each location? which flow equation describes temperature evolution?).

The L*SHA algorithm [11] addresses both challenges through an iterative query-based process. The learning system consists of two agents: a *Learner* that proposes candidate automata, and a *Teacher* that validates proposals against a database of operational traces. The interaction proceeds as follows:

1. The Learner constructs an initial hypothesis automaton from a subset of training traces, creating locations for observed operational modes and transitions for observed event sequences.
2. The Teacher evaluates whether the hypothesis automaton can accept all traces in the database. If a trace exists that the automaton cannot reproduce (a *counterexample*), the Teacher returns it to the Learner.
3. The Learner refines the automaton structure to accommodate the counterexample, adding new locations or transitions as needed.
4. Steps 2–3 repeat until no counterexamples remain and the automaton is consistent with all training data.

Throughout this process, the Teacher also characterizes each location's continuous behavior by fitting probability distributions to temperature measurements and selecting appropriate flow conditions (constant, linear, logarithmic, etc.) from a predefined library of candidate functions.

4.2.3. From Sensor Data to Learned Automata

The application of the L*SHA framework to the GR3N decanter centrifuge[11] involved several preprocessing and design steps before automatic learning could proceed:

Data collection and segmentation. Five months of operational telemetry was collected from the decanter's sensor system, including boolean signals (motor working status, alarms), continuous measurements (current absorption, torque, bearing temperatures), and environmental variables (slurry feed temperature, pump speed). The continuous time-series data was segmented into discrete *operational cycles*—complete runs from decanter startup through shutdown—yielding 25 candidate traces after initial filtering.

Event alphabet definition. The learning algorithm requires a discrete event alphabet to construct automaton transitions.Six events were defined based on domain knowledge and statistical analysis: motor start (`W_ON`), motor stop (`W_OFF`), and four current absorption threshold crossings (0–100A, 100–200A, 200–300A, ≥300A). These events partition the continuous current signal into discrete operational regimes that correlate with distinct thermal behaviors.

Trace encoding. Each operational cycle was converted into a *timed trace*: a sequence of event symbols annotated with occurrence timestamps and synchronized continuous variable

measurements. For example, a typical trace might encode: `[W_ON @ 0s, A_0 @ 0s, A_1 @ 45s, A_2 @ 120s, ..., W_OFF @ 444s]`, with bearing temperature recorded at each event time.

Hyperparameter tuning. The L*SHA framework exposes several design parameters that influence the learned automaton's structure and accuracy:

- *Delta* (Δ): tolerance for grouping similar distributions into the same location. Low values ($\Delta = 0.01$) create fine-grained automata with many locations; high values ($\Delta = 0.5$) produce coarse models with fewer states.
- *Default flow condition*: the differential equation assigned to locations when statistical evidence is insufficient to select confidently between candidates.
- *Training/validation split*: the partition of traces between initial hypothesis construction, counterexample refinement, and final validation.

By comparing automata generated with different parameter settings—over 10 experimental configurations[11]—the optimal balance between model complexity and predictive accuracy was identified.

Learned automaton structure. The result of the learning process is an SHA comprising:

- A set of *locations* (discrete modes), each characterized by a Gaussian distribution (μ_i, σ_i) governing bearing temperature at entry and a flow condition $f_i(T)$ specifying the continuous temperature evolution rate while in that location.
- A set of *transitions* connecting locations, each labeled with an event from the alphabet and guarded by predicates ensuring deterministic behavior (given the same event sequence, the automaton always follows the same path).
- *Initial conditions* specifying the automaton's starting location and initial temperature distribution.

For the GR3N decanter, the learned baseline automaton contains 8 locations representing distinct thermal operating modes (startup, high-load operation, reduced-load operation, shutdown, etc.), 15 transitions, and 4 unique temperature distributions.

4.2.4. Model Outputs and Artifacts

The learning framework produces three files encoding the learned SHA:

`Model_source.txt` A DOT-format graph representation listing all locations, transitions, and their connectivity. This file captures the automaton's discrete structure.

`Model.txt` A report file containing the learned continuous parameters: the Gaussian distribution (μ_i, σ_i) for each location i, and the flow condition function $f_i(T)$ governing temperature evolution. Example entry:

```
Location 2: mu=23.5, sigma=1.2, flow=linear(alpha=0.05)
```

`Model_histogram_values.txt` Raw temperature measurements collected during learning, stored per-location for statistical validation of the fitted distributions.

These three files serve as input to the validation framework described in the remainder of this chapter. The validation process reconstructs the learned SHA, encodes it into UPPAAL's timed automaton formalism, and executes Statistical Model Checking simulations to assess predictive accuracy on operational traces not used during training.

4.3. Validation Methodologies: A Comparative Analysis

Understanding the strengths and limitations of different validation approaches clarifies why the learning-based statistical model checking approach represents the optimal methodology for industrial hybrid systems.

4.3.1. Traditional Approaches and Their Limitations

Simulation-Based Testing

Simulation-based testing remains the most common validation approach in industry. Engineers develop simulation models, generate test inputs covering various scenarios, and check if outputs meet specifications. However, simulation is inherently incomplete—critical behaviors may be missed because the execution space of cyber-physical systems is infinite. As Dang and Nahhal observed, "The infinite execution space of CPS makes test case generation fundamentally incomplete" [22].

Simulation results depend critically on initial conditions and test inputs. A system may appear correct under tested conditions but fail catastrophically under untested scenarios. This is particularly dangerous for safety-critical systems where rare but catastrophic failures must be prevented.

Hardware-in-the-Loop Testing

Hardware-in-the-Loop (HIL) testing integrates real hardware with simulated environments, providing more realistic validation than pure simulation. Industrial HIL facilities cost approximately one-tenth of actual plant testing while enabling controlled experimentation. However, HIL testing faces significant limitations:

High initial investment in HIL infrastructure, model accuracy requirements for the simulated environment, latency constraints that may introduce artifacts, and incomplete coverage of physical interactions between components. HIL testing is valuable for controller validation but cannot provide formal guarantees about system behavior.

Hand-Crafted Formal Models

Manually constructing formal models for verification requires months of expert effort. Engineers must deeply understand system behavior, translate that understanding into formal specifications, and iterate to eliminate modeling errors. The resulting models encode idealized assumptions that may not capture actual system behavior.

Recent research on hybrid modeling design patterns documented in the Journal of Mathematics in Industry (2024) [20] notes that "Physics-based models fall short in capturing finer details and short-term effects" and "cannot encapsulate stochasticity inherent in real-world processes." This model-reality gap undermines validation confidence even when formal verification succeeds.

4.3.2. Formal Verification: Power and Limitations

Exhaustive model checking provides mathematical guarantees: if a property is verified, it holds for all system executions. This absolute certainty is invaluable for safety-critical systems where failure consequences are severe. However, fundamental limitations constrain applicability.

Henzinger et al. proved in 1995 that "most verification problems for hybrid automata are undecidable even under severe limitations" [15]. For general hybrid systems with nonlinear dynamics, reachability—the most basic verification question—cannot be algorithmically decided. Even for restricted linear hybrid systems, complexity is prohibitive.

The state explosion problem has been "the main obstacle in applying model checking" for over three decades. Practical limits for the most efficient complete algorithms are approximately 6 continuous dimensions. While tools like SpaceEx push boundaries to 100+ variables through sophisticated abstractions [13], they remain limited to piecewise-affine

dynamics without stochastic support.

For industrial systems with tens of continuous state variables, complex nonlinear dynamics, and stochastic behaviors, exhaustive verification is simply not feasible.

4.3.3. Statistical Model Checking Advantages

SMC overcomes exhaustive verification limitations through a key insight articulated by David et al.: "It may not be necessary to obtain an absolutely accurate estimate of a probability to verify probabilistic properties. Instead, it is enough to infer, by sufficiently sampling the underlying model, that the probability is safely above or below a threshold" [9].

Statistical model checking provides several critical advantages:

State-space independence: Scalability arises because SMC does not require constructing the full state space. Simulation generates states on-demand, with memory requirements depending only on simulation length, not total state-space size.

Tunable accuracy: Users control the trade-off between computational cost and precision. Tighter confidence bounds require more simulations but provide more precise estimates. This flexibility enables practical engineering decisions about validation effort.

Low memory requirements: Unlike exhaustive methods that may require gigabytes to store state spaces, SMC needs only enough memory for a single simulation trace.

4.3.4. Why Learning Combined with SMC Represents the Optimal Approach

The combination of data-driven modeling with statistical model checking provides the best trade-off for complex hybrid systems. Table 4.1 summarizes the comparison across key criteria.

Criterion	**Traditional**	**Exhaustive Formal**	**Learning + SMC**
Scalability	Medium	Very Low	High
Automation	Medium	Low	Very High
Guarantees	None	Mathematical proof	Statistical
Model-reality gap	Present	Present	Minimized
Stochasticity support	Limited	Very limited	Native
Industrial applicability	Established	Limited	Growing

Table 4.1: Comparison of validation approaches for hybrid systems

For industrial cyber-physical systems like decanter centrifuges—where operational data is available, system dynamics involve complex nonlinearities and stochastic effects, and exhaustive verification is infeasible—the learning-plus-SMC approach provides the optimal methodology.

Data-driven learning captures actual system behavior automatically, eliminating the model-reality gap that plagues hand-crafted approaches. The learned models reflect how the system actually operates rather than how engineers believe it should operate. Stochastic parameters are estimated from real operational variability rather than being guessed or ignored.

Statistical model checking then provides rigorous probabilistic validation without the computational barriers of exhaustive verification. Engineers obtain quantifiable confidence bounds on safety and performance properties, enabling informed decisions about system deployment.

4.4. Implications for Centrifuge Validation

This comprehensive background establishes the theoretical and methodological foundation for validating Stochastic Hybrid Automata models of industrial decanter centrifuges. The 40-year evolution from simulation testing through formal verification to statistical model checking reflects the field's response to fundamental computational barriers—undecidability and state explosion—that make exhaustive verification of complex hybrid systems impossible.

UPPAAL SMC emerges as the appropriate validation platform, combining the theoretical rigor of formal methods with the practical scalability needed for industrial systems. Its support for stochastic timed automata directly addresses the centrifuge modeling requirements of combining discrete operational modes, continuous thermal and mechanical dynamics, and stochastic failures. The statistical approach provides quantifiable confidence bounds on safety and performance properties without requiring intractable state-space enumeration.

The SHA learning framework addresses the model-reality gap that plagues hand-crafted approaches. For centrifuge systems exhibiting complex fluid dynamics, thermal behavior with heat transfer between components, wear-dependent mechanical characteristics, and stochastic failure modes, learning models from operational data captures actual behavior that first-principles models cannot adequately represent.

The next chapter will introduce the decanter centrifuge system under study, describing its

operational characteristics, the available sensor data, and the specific validation challenges this thesis addresses. Subsequent chapters will detail the model learning process, the validation methodology implementation, and the results demonstrating that learned SHA models can reliably predict centrifuge behavior with quantifiable confidence bounds.

5 | Validation Framework Implementation

This chapter details the internal architecture of the validation framework—how each software component is designed, what data it consumes and produces, and how the components interact to transform raw CSV operational traces into quantitative validation metrics.

5.1. Validation Execution Overview

The validation framework automates the testing of learned SHA models against operational traces, implementing a two-phase strategy that balances computational efficiency with rigorous accuracy assessment. Figure 5.1 illustrates the complete execution workflow.

5.1.1. Initialization and Preparation

The validation script begins by loading the learned SHA model from three input files produced by the L*SHA learning framework: the DOT-format automaton structure (`Model_source.txt`), the learned continuous parameters—Gaussian distributions (μ_i, σ_i) and flow conditions for each location (`Model.txt`)—and raw training data histograms (`Model_histogram_values.txt`). The framework parses these files into an internal `SHA` object representing locations, transitions, guards, and stochastic parameters.

Configuration settings specify output directories, case study type (GR3N or ENERGY), and validation trace locations. A model-specific output directory is created to organize results: `plots/{SHA_NAME}/`. The logger is initialized to record validation events to both console and file.

Next, the framework identifies which operational traces are structurally compatible with the learned SHA. A trace is *compatible* if the automaton can accept its complete event sequence—i.e., every event in the trace has a corresponding enabled transition in the SHA's current location. The `get_compatible_traces_GR3N()` function performs lightweight

simulation of the SHA's discrete behavior (ignoring continuous dynamics) against each candidate trace. For the GR3N decanter, seventeen validation traces (`cd1-cd17`) are tested; incompatible traces are excluded to avoid wasting computation on simulations that will deadlock.

For each compatible trace, the framework extracts ground-truth bearing temperature measurements from the CSV sensor data using `get_cut_signals_GR3N()`. These signals serve as the reference for comparing UPPAAL's simulated predictions against reality. Timing information (trace start time, end time, total duration `TAU`) is also extracted to configure UPPAAL's simulation bounds.

5.1.2. Two-Phase Validation Strategy

The framework employs a dual-template approach to optimize computational efficiency while maintaining rigorous validation coverage:

Phase 1: Fast Compatibility Verification. Before committing to expensive Statistical Model Checking, Phase 1 confirms the SHA can process the complete trace without errors. The framework generates a *minimal UPPAAL model* by calling `generate_upp_model()` with `validation=True`. This model contains only:

- Event synchronization channels (`W_ON`, `W_OFF`, current thresholds)
- Automaton location structure and transitions
- Guard predicates and action labels
- Encoded trace arrays (`TRACE[]`, `TIMES[]`)

Critically, the minimal model *omits* probability distributions, Box-Muller sampling routines, and flow condition differential equations. The `run_exp()` function executes UPPAAL's `verifyta` engine to check whether the Program automaton successfully consumes all events. This verification completes in seconds. If the trace fails compatibility, it is rejected and the framework proceeds to the next candidate without wasting time on full simulation.

Phase 2: Full SMC Simulation. Traces passing Phase 1 proceed to detailed simulation. The framework calls `generate_upp_model()` again with `validation=False`, generating the *complete UPPAAL model* including:

- Gaussian distribution arrays (`DISTR[n][2]`) with learned (μ_i, σ_i)
- Box-Muller random number generator for stochastic sampling

- Flow condition functions (`f_0(T), f_1(T), ...`) encoding temperature differential equations
- BearingTemperature automaton with continuous clock `Temp`

The `run_smc_simulation()` function invokes UPPAAL with the SMC query:

```
simulate [<=TAU] {m_1.Temp}
```

which simulates bearing temperature evolution up to the trace time bound `TAU` seconds, recording temperature values at each event time. The output—a sparse signal sampled at event timestamps—is written to `{trace_name}_out.txt`.

This two-phase design provides significant computational savings. Generating and simulating the complete UPPAAL model requires 2–5 minutes per trace due to stochastic sampling and continuous dynamics. Phase 1's minimal model completes in under 10 seconds, immediately rejecting incompatible traces. For a validation campaign with seventeen candidate traces where only thirteen are compatible, this strategy saves approximately 20 minutes (4 rejected $\times$ 5 min/trace).

5.1.3. Results Analysis and Iteration

The `analyze_results_GR3N()` function processes UPPAAL's simulation output for each successfully simulated trace. The parsing module (`Upp2Sig`) extracts simulated temperature values from the text output file, creating a sparse signal (typically 40–50 points sampled at event times). This simulated signal is aligned with the dense ground-truth signal (400+ sensor measurements) by matching timestamps.

Three quantitative accuracy metrics are computed:

- **Temperature Prediction Error (%)**: Percentage difference between final predicted and actual temperatures, assessing steady-state accuracy.
- **Mean Absolute Error (MAE, °C)**: Average absolute deviation across all aligned points, measuring typical prediction accuracy.
- **Root Mean Square Error (RMSE, °C)**: Quadratic mean emphasizing larger deviations, detecting occasional badly wrong predictions.

A PDF plot is generated overlaying the real bearing temperature trajectory (blue) and SHA prediction (orange) on a common time axis. Plots are saved to `plots/{SHA_NAME}/{trace_name}.pdf`. Metrics are logged to console and appended to the model summary file.

The validation loop repeats for every compatible trace: extract signals, verify compatibility

(Phase 1), simulate (Phase 2), analyze, next trace. This stateless design enables parallel execution—multiple validation instances can process different traces simultaneously, writing results to separate model-specific directories.

Upon completing all traces, the script logs summary statistics: total traces processed, successful simulations, analysis errors, and aggregate metrics (mean MAE across all traces, worst-case error, etc.). This provides a high-level assessment of model quality before detailed examination of individual trace plots.

The validation framework implements an efficient, automated pipeline for rigorous SHA model testing. The dual-phase strategy minimizes wasted computation while maintaining comprehensive validation coverage across all compatible operational traces.

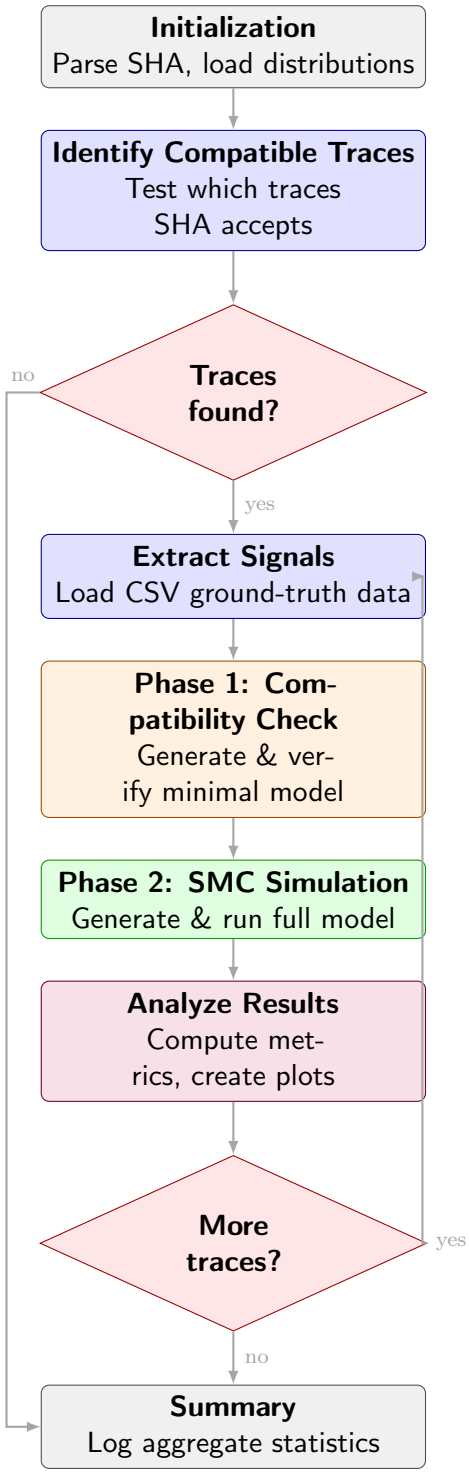

Figure 5.1: Validation workflow executing a two-phase strategy. Fast compatibility checking (orange) filters traces before expensive SMC simulation (green). The process iterates over all compatible traces.

5.2. Overall Validation Architecture

5.2.1. Automated Workflow

Figure 5.2 illustrates the nine sequential stages that the runner executes. Each stage is described in detail below.

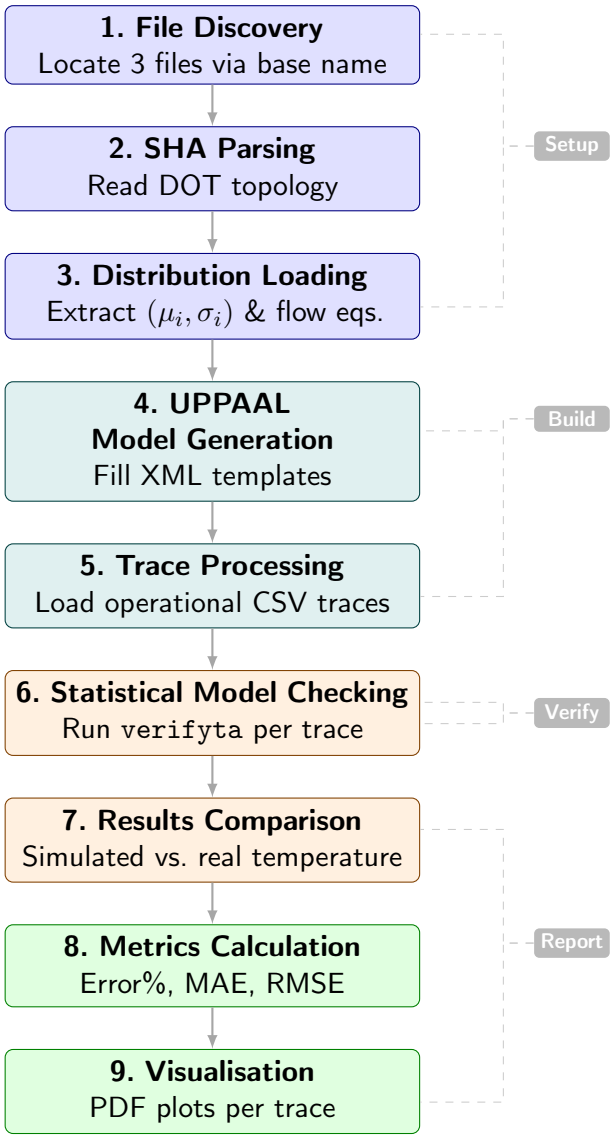

Figure 5.2: The nine automated stages executed by `run_validation.py`. Colours indicate phases: blue = setup, teal = model build, orange = verification, green = reporting.

Stage 1 – File Discovery. The runner calls a path-resolver that appends the three known suffixes to the provided base name and asserts that all files exist with read permissions. Any missing file raises an informative error before any computation begins.

Stage 2 – SHA Parsing. The DOT parser reads `Model_source.txt` and builds an internal `SHA` object whose fields mirror the automaton: a dictionary of locations indexed by integer IDs, a list of transitions carrying guard predicates and action labels, and the designated initial location ID.

Stage 3 – Distribution Loading. The report parser extracts the Gaussian parameters $\{(\mu_0, \sigma_0), \ldots, (\mu_{n-1}, \sigma_{n-1})\}$ (one pair per location) and the piecewise flow conditions from `Model.txt`. Optionally, the histogram file is read to cross-check that the fitted parameters are consistent with the raw simulation data.

Stage 4 – UPPAAL Model Generation. The `SHA2Upp` module (Chapter 5) fills the four XML template files with the parsed distributions, flow equations, and the encoded validation trace, producing ready-to-run `.xml` and `.q` files in `resources/Validation/Models/` and `resources/Validation/Queries/`.

Stage 5 – Trace Processing. The `TraceParser` module reads each operational CSV file from the configured trace directory and converts it to a timed event sequence. Both a *filtered* (change-point only) trace and a *full* (every row) trace are produced; the dual-trace strategy is explained in Section 5.3.1.

Stage 6 – Statistical Model Checking. For each compatible trace, the `VerMgr` module invokes UPPAAL's command-line engine `verifyta` with the generated model and query file. The SMC query `simulate [<=TAU] {m_1.Temp}` drives the simulation up to the trace time bound `TAU` and records temperature values at every event time.

Stage 7 – Results Comparison. The `Upp2Sig` parser converts UPPAAL's text output into `SampledSignal` objects. The `ResMgr` module then aligns the sparse simulated signal (one point per event) against the dense real signal (one point per sensor reading) for comparison.

Stage 8 – Metrics Calculation. Three quantitative metrics are computed for each trace: temperature prediction error percentage, Mean Absolute Error (MAE), and Root Mean Square Error (RMSE). Results are written to the model log file.

Stage 9 – Visualisation. A PDF plot is generated for each trace showing the real bearing temperature (blue line) and the SHA model prediction (orange line) on a common time axis. Plots are saved to `resources/Validation/resources/plots/Model/`.

Table 5.1: Framework modules and their responsibilities

Module	File	Responsibility
TraceParser	`mgrs/TraceParser.py`	Convert CSV rows to timed event sequences
ValMgr	`mgrs/ValMgr.py`	Orchestrate compatibility check and trace selection
SHA2Upp	`mgrs/SHA2Upp.py`	Fill XML templates to generate UPPAAL models
VerMgr	`mgrs/VerMgr.py`	Invoke `verifyta`, capture raw output
Upp2Sig	`mgrs/Upp2Sig.py`	Parse simulation output into signal objects
ResMgr	`mgrs/ResMgr.py`	Compute error metrics and generate PDF plots

5.3. UPPAAL Templates

Rather than generating UPPAAL XML from scratch for every model and trace, the framework ships four template files that contain static structure plus named placeholders of the form `**PLACEHOLDER**`. The SHA2Upp module replaces every placeholder at validation time. This design makes the templates human-readable and easy to audit independently of the Python code.

5.3.1. Dual-Template Strategy

A key design principle is the separation of two different verification tasks: *compatibility checking* (does the SHA accept this event sequence at all?) and *statistical estimation* (what temperature trajectory does the SHA predict?). These tasks have very different computational costs, so the framework uses two distinct pairs of templates.

Validation templates (`*_val.xml`). `nta_template_val.xml` and `machine_sha_template_val.xml` define the smallest possible model: the event channels, the global clock `t`, the trace arrays (`TRACE`, `TIMES`), and the bare SHA location structure without distribution declarations or flow equations. The sole purpose of this model is to verify the TCTL query:

$$A\langle\rangle \ \texttt{p\_1.next\_i} = N_E$$

which asks whether the Program automaton can consume all N_E events. If the answer is *no*, the trace is structurally incompatible with the SHA (a guard was never satisfiable)

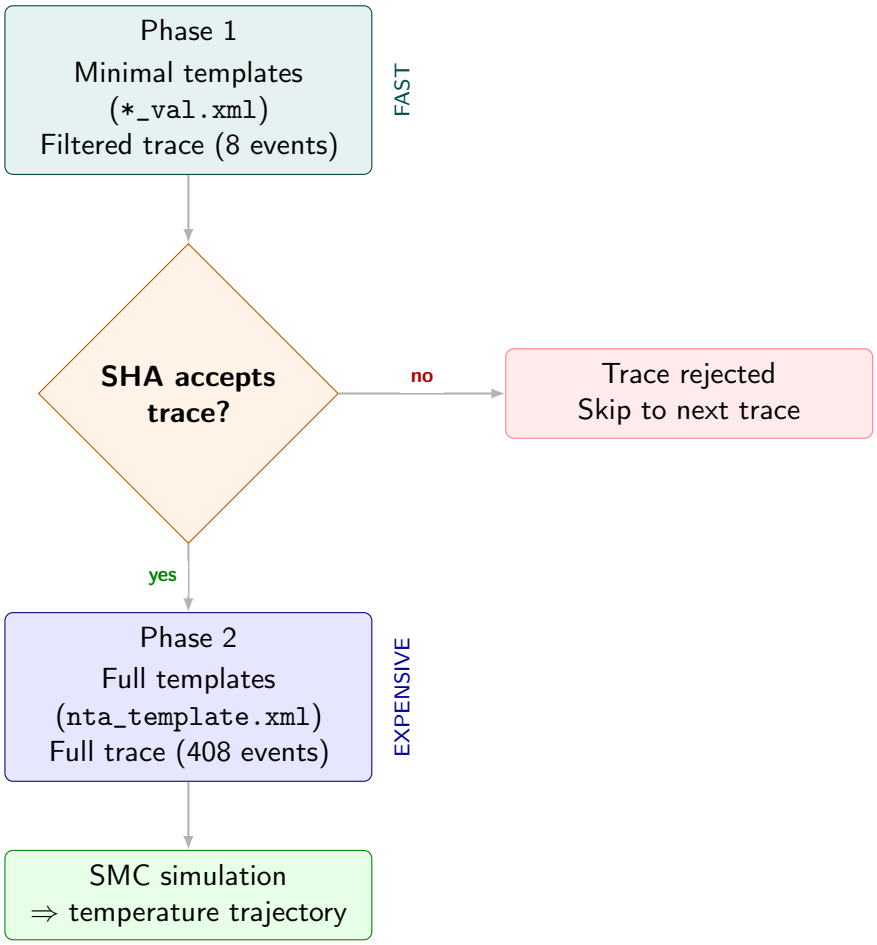

Figure 5.3: Dual-template strategy. A fast compatibility check on a minimal model filters out incompatible traces before committing to costly SMC simulations on the full model.

and is rejected.

Main templates (`nta_template.xml`, `machine_sha_template.xml`). These templates add the full probabilistic machinery: Box-Muller sampling for Gaussian distributions, the learned distribution arrays `DISTR`$[n][2]$, and the differential equations governing the evolution of temperature . The Program automaton is identical to the validation version; the BearingTemperature automaton gains a temperature clock `Temp` and a `set_vars()` function invoked on every transition.

Global declarations (main template). The global scope of `nta_template.xml` declares the six broadcast channels used to communicate between the Program automaton and the SHA machine (Table 5.2), the Box-Muller random-number generator, the learned distribution array, and the encoded trace.

Table 5.2: Broadcast channels in the UPPAAL model

Channel	Meaning	Trigger condition
`W_ON`	Motor start	Working status $0 \rightarrow 1$
`W_OFF`	Motor stop	Working status $1 \rightarrow 0$
`a__0_100`	Current 0–100 A	$A \in [0, 100)$
`a__100_200`	Current 100–200 A	$A \in [100, 200)$
`a__200_300`	Current 200–300 A	$A \in [200, 300)$
`a__300_plus`	Current $\geq$300 A	$A \geq 300$

The Program automaton advances its local clock `x` until `x` $\geq$ `T` (the next event timestamp), fires the corresponding broadcast channel, resets `x` to zero, and retrieves the next event through `get_next()`. Because `x` is reset on every event, `TIMES` stores *cumulative* times (not deltas), so the guard `x >= T` fires at the correct absolute time.

5.4. Trace Parser

The `TraceParser` module is responsible for bridging the gap between the raw CSV files produced by the GR3N sensor system and the integer-encoded event arrays required by UPPAAL. It provides two conversion modes.

Event encoding. Each trace event is encoded as an integer for storage in the UPPAAL `TRACE[]` array. Table 5.3 defines the mapping.

Table 5.3: Event encoding used in the UPPAAL TRACE array

ID	Symbol	Meaning
0	`W_ON`	Working status transitions to 1
1	`W_OFF`	Working status transitions to 0
2	`A_0`	Current absorption $\in$ [0, 100) A
3	`A_1`	Current absorption $\in$ [100, 200) A
4	`A_2`	Current absorption $\in$ [200, 300) A
5	`A_3`	Current absorption $\geq$ 300 A

Handling simultaneous events. When the working status changes *at the same timestamp* as a current absorption reading, the `W_ON`/`W_OFF` event is inserted with a time-delta of *zero* before the current event. This preserves causal ordering while maintaining timing accuracy—the UPPAAL clock is not advanced until a non-zero delta arrives.

5.5. SHA-to-UPPAAL Generator (SHA2Upp)

The `SHA2Upp` module is the heart of model generation. It reads the internal `SHA` object, the learned distributions, and the encoded trace, and produces a complete, self-contained UPPAAL XML file by filling every placeholder in the main templates.

5.5.1. Placeholder Substitution

Figure 5.4 shows how the eight global placeholders are resolved.

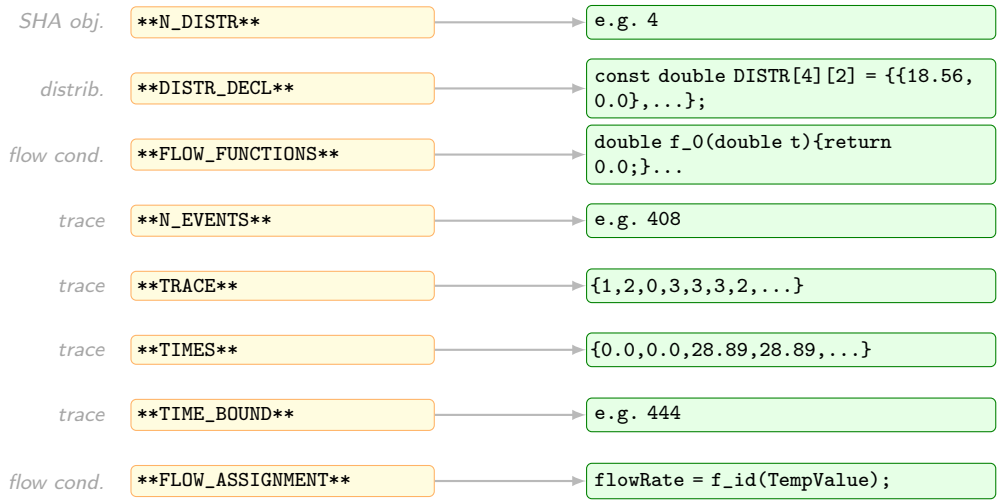

Figure 5.4: Placeholder substitution in `nta_template.xml`. Yellow boxes are template placeholders; green boxes show the generated values.

Distribution declarations. For each location i in the SHA, the module writes:

```
const double D_i[2] = {mean_i, std_i};
```

and assembles all pairs into a 2-D array `DISTR[n][2]`. When the BearingTemperature automaton enters location i, it calls `Normal(DISTR[i][0], DISTR[i][1])` to sample the initial temperature using the Box-Muller transform already declared in the global scope.

Flow functions. Each location is associated with a flow condition read from the `GR3N.txt` file in `resources/Validation/resources/flow_conditions/`. A constant-temperature location generates:

```
double f_0(double temp) { return 0.0; }
```

while a heating location might generate:

```
double f_1(double temp) { return alpha*(temp_target - temp); }
```

These functions are referenced inside the UPPAAL location invariants to drive the continuous evolution of `Temp`.

Trace encoding. The `_encode_trace` method iterates over the timed event list, maps each symbol to its integer ID (Table 5.3), and accumulates cumulative timestamps. The result is two parallel arrays of length N_E: `TRACE[N_E]` (integer event IDs) and `TIMES[N_E]` (doubles in seconds). Timestamps are written with 15 decimal places to avoid rounding errors in UPPAAL's floating-point clock comparisons.

5.6. Verification Manager (VerMgr)

`VerMgr` is the thin layer between the Python framework and UPPAAL's command-line engine. For each trace it constructs the OS-level command string:

```
verifyta "Models/cd1.xml" "Queries/cd1.q" > "cd1_out.txt"
```

and launches it as a subprocess via Python's `subprocess.run()`. The query file contains a single SMC statement:

```
simulate [<=TAU] {m_1.Temp}
```

which instructs UPPAAL to record the value of the BearingTemperature automaton's temperature clock at every event time within the bound `TAU` seconds. The raw output (a space-separated table of timestamps and temperature values) is redirected to `cd1_out.txt` for subsequent parsing by `Upp2Sig`.

5.7. Output Parser and Signal Comparison

5.7.1. Upp2Sig: Parsing Simulation Output

UPPAAL's simulation output is a plain-text file where each sampled point occupies one line of the form `[time, Temp_min, Temp_max]`. `Upp2Sig` reads this file line by line, extracts the three numeric fields, and assembles three `SampledSignal` objects: `Temp_min`, `Temp_max`, and `Temp_avg` (the arithmetic mean of min and max across simulation runs).

An important characteristic of the output is its *sparsity*: UPPAAL only records values at event times. For trace `cd1` this yields approximately 47 temperature points from a total simulation duration of ≈ 444 seconds, whereas the real CSV contains 401 rows covering the same period.

5.7.2. ResMgr: Alignment and Metrics

`ResMgr` receives the two signal lists—the sparse simulated signal and the dense real signal—and performs point-wise alignment by matching the nearest real-signal timestamp to each simulated timestamp. Three scalar metrics are then computed:

$$\varepsilon_{\%} = \frac{|\hat{T}_{\text{final}} - T_{\text{final}}|}{|T_{\text{final}}|} \times 100 \tag{5.1}$$

$$\text{MAE} = \frac{1}{K} \sum_{k=1}^{K} |\hat{T}(t_k) - T(t_k)| \tag{5.2}$$

$$\text{RMSE} = \sqrt{\frac{1}{K} \sum_{k=1}^{K} \left(\hat{T}(t_k) - T(t_k)\right)^2} \tag{5.3}$$

where $T(t_k)$ is the real bearing temperature at event time t_k, $\hat{T}(t_k)$ is the simulated average, and K is the number of matched pairs. All three metrics are written to the terminal log and appended to the model summary file.

6 | Validation Setup

Before the learned Stochastic Hybrid Automaton can be validated against real operational data, the environment must be carefully prepared. This chapter describes the three prerequisite output files produced by the SHA learning phase, the precise directory locations where they must be placed, and the single command that triggers the full automated pipeline. Getting this setup right is a necessary precondition for every experiment described in the following chapter.This chapter can be used as a sort of manual for setup, to run the framework.

6.1. Prerequisites: Outputs from the Learning Phase

The validation framework is purposely decoupled from the learning framework so that any conformant SHA model—regardless of which learning algorithm produced it—can be validated without modification. The coupling is achieved through three files whose *base name* must be identical. If the model is called `Model`, the three files are:

`Model_source.txt` The learned automaton encoded in the DOT graph language. Every location, guard condition, synchronisation label, and action is captured here. The validation framework parses this file to reconstruct the SHA topology before generating UPPAAL XML.

`Model.txt` The complete validation report produced by the learning framework. It contains the fitted Gaussian distribution parameters (μ_i, σ_i) for every location i, the piecewise-linear flow conditions that govern temperature evolution $\texttt{Temp}' = f(\texttt{Working}, \texttt{Assorbimento}, \texttt{Temp})$, and general metadata about the training run.

`Model_histogram_values.txt` Raw histogram counts from the UPPAAL simulations executed during training. These empirical frequency tables are re-used by the distribution-fitting step to verify that the Gaussian parameters stored in `Model.txt` represent the data faithfully.

Naming Convention — Critical Requirement

All three files must share the *same base name*. The framework performs string-based file discovery by appending the fixed suffixes `_source.txt`, `.txt`, and `_histogram_values.txt`. A mismatch in capitalisation or spelling causes an immediate "file not found" error that aborts the pipeline before any validation work is done.

Table 6.1 summarises the three files at a glance.

Table 6.1: Required input files for the validation framework

File	Content	Consumed by
`Model_source.txt`	SHA topology (DOT format)	SHA2Upp
`Model.txt`	Distributions & flow conditions	SHA2Upp
`Model_histogram_values.txt`	Raw histogram bins & counts	Distribution fitting

6.2. File Placement

Each file must be placed in a specific sub-directory of the project tree before the runner script is invoked. Figure 6.1 shows the required directory structure after placement.

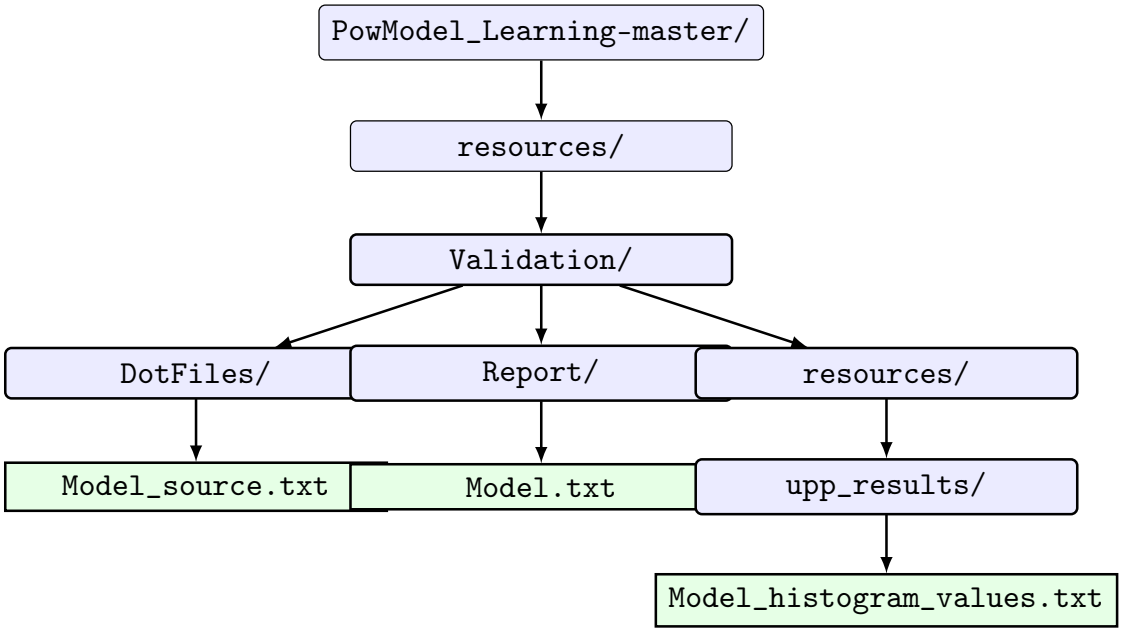

Figure 6.1: Required directory structure for the three input files. Green boxes are files; blue boxes are directories.

SHA Structure File. `Model_source.txt` must be placed in `resources/Validation/DotFiles/`. The SHA2Upp module opens this directory when constructing the UPPAAL XML and expects exactly the DOT file whose name matches the base-name argument passed to the runner script.

Validation Report File. `Model.txt` must be placed in `resources/Validation/Report/`. This is the primary configuration source: every stochastic parameter and every flow equation for the UPPAAL model is read from this file.

Histogram Data File. `Model_histogram_values.txt` must be placed in `resources/Validation/` Although this file plays a secondary role (re-fitting distributions for sanity checking), its absence causes the pipeline to abort during initialisation.

6.3. Executing Validation

With all three files in place, validation is launched with a single command from the project root:

```
python it/polimi/powmodel_learning/run_validation.py Model
```

The sole positional argument is the shared base name (`Model` here). The script automatically appends the correct suffix when opening each file; the caller never needs to specify full file names.

6.4. Output Organisation

Upon successful completion, the framework writes all outputs to a single model-specific sub-directory so that results from different models never overwrite each other:

```
resources/Validation/resources/plots/Model/
    Model.txt          ← full terminal log
    cd1_XXX.pdf        ← real vs. predicted temperature (trace cd1)
    cd2_XXX.pdf        ← real vs. predicted temperature (trace cd2)
    ...
```

The integer suffix `XXX` in each PDF name is generated randomly to prevent accidental overwriting when the same trace is validated more than once with different models. Each PDF contains a single figure: the two temperature trajectories plotted on a shared time axis, together with a legend, axis labels, and a title identifying the trace.

6.5. Common Setup Errors

File-not-found error. Verify that the three files are in the exact directories listed in Section 6.2 and that the base name matches exactly, including capitalisation (the file system is case-sensitive on Linux).

Permission error. Confirm that the process has write access to the `plots/` output directory. On shared computing systems, the directory may have been created by a different user.

UPPAAL not found. The `verifyta` binary must be on the system `PATH` or its full path must be specified in `resources/config/config.ini`. Stage 6 will fail immediately if `verifyta` cannot be located.

Configuration mismatch. The `config.ini` file must specify the correct path to the operational CSV traces and the correct case-study identifier (`GR3N`) so that the GR3N-specific trace-parsing and template-selection paths are activated.

The setup procedure reduces to three actions: place the three learning-phase output files in their designated directories, verify that UPPAAL and the Python dependencies are accessible, and run a single command. The framework takes care of all remaining steps automatically—from parsing the SHA topology to writing the final comparison plots. This design enables reproducible, hands-off validation of any conformant SHA model without writing any case-study-specific code.

7 | Validation Results and Analysis

This chapter presents the empirical validation of four learned Stochastic Hybrid Automata models against seventeen operational traces from the GR3N decanter centrifuge. A total of 68 validation runs were executed, producing quantitative metrics and visual comparisons that demonstrate the viability of data-driven SHA modeling for industrial bearing-temperature prediction. The analysis focuses primarily on the successful `GR3N_SIM` model, which achieved consistent performance without catastrophic failures, making it suitable for production deployment.

7.1. Performance Metrics

Three complementary metrics quantify prediction accuracy for each model-trace pair:

Temperature Error (%) The percentage difference between the final predicted bearing temperature and the final observed temperature:

$$\varepsilon_{\%} = \frac{|\hat{T}_{\text{final}} - T_{\text{final}}|}{T_{\text{final}}} \times 100 \tag{7.1}$$

This metric assesses whether the model reaches the correct steady-state temperature at cycle completion.

Mean Absolute Error (MAE, °C) The average absolute deviation between predicted and observed temperatures across all sampled points in the trace:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |\hat{T}_i - T_i| \tag{7.2}$$

MAE provides a measure of typical prediction accuracy throughout the entire operational cycle, penalizing all errors equally.

Root Mean Square Error (RMSE, °C) The quadratic mean error that emphasizes

larger deviations:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{T}_i - T_i)^2} \tag{7.3}$$

RMSE is more sensitive than MAE to occasional large errors, making it valuable for detecting models that occasionally produce badly wrong predictions even if their average accuracy is acceptable.

7.2. Overall Performance Comparison

Table 7.1 summarises the aggregate performance of all four models across the seventeen validation traces. The baseline model (developed by Fernández Carrera [11]) emerges as the clear winner by every metric: lowest average temperature error (43.97%), lowest MAE (5.88°C), lowest RMSE (7.61°C), and highest win count (13 out of 17 traces). Figure 7.1 visualises this performance gap.

Table 7.1: Model performance rankings (averaged across 17 validation traces)

Rank	Model	Avg Error (%)	Avg MAE (°C)	Avg RMSE (°C)	Best Count
1	**Baseline model (Fernández Carrera)**	**43.97**	**5.88**	**7.61**	**13/17**
2	Alternative model 1	192.33	16.48	21.12	2/17
3	Enhanced model (present work)	438.96	10.37	14.67	2/17
4	Alternative model 2	376.51	29.26	33.62	0/17

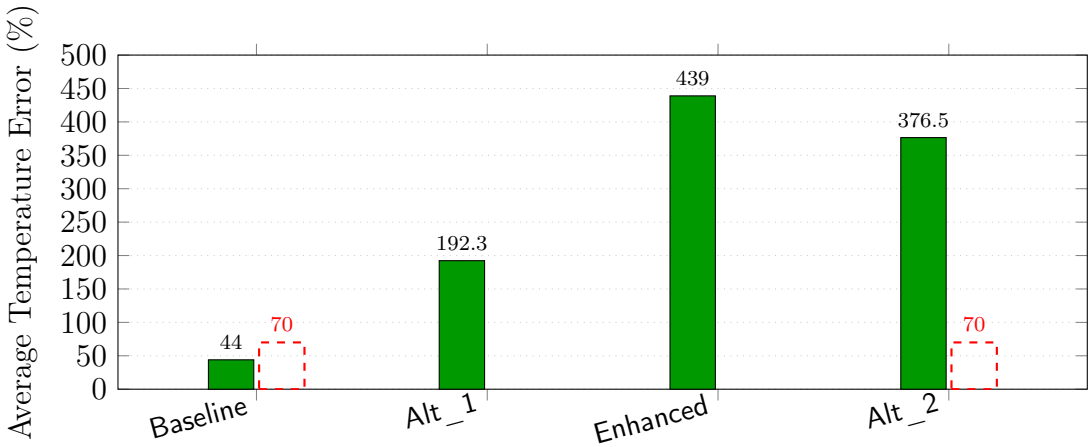

Figure 7.1: Average temperature prediction error across all four models. Only the baseline model (left bar) falls below the 70% acceptability threshold. The other three models exhibit errors exceeding 190% due to catastrophic failures on specific traces.

The stark difference in average error between the baseline model and the alternatives arises not from marginally worse performance but from *catastrophic failures* where models

predict bearing temperatures exceeding 200–300°C when actual values are 20–30°C. These outliers inflate the aggregate metrics and disqualify the models from industrial use.

7.3. Detailed Analysis: Baseline Model (Fernández Carrera)

The baseline model is an 8-state SHA with 15 transitions, four learned Gaussian distributions (one per unique temperature mode), and three flow conditions (constant, linear, and logarithmic). This model was developed by Fernández Carrera in the preceding thesis work [11] through rigorous parameter tuning and iterative refinement. It represents the only deterministic model from that work that could be systematically validated, having undergone extensive testing and hyperparameter optimization during the learning phase. The robust performance demonstrated in this validation campaign validates the careful model development approach taken in that prior work.

7.3.1. Per-Trace Performance

Table 7.2 details the baseline model's predictions for every validation trace. The model achieves temperature errors below 50% on 14 out of 17 traces (82%), with three traces—`cd8`, `cd6`, and `cd17`—delivering exceptional performance below 32% error.

The standard deviation of 11.91% on temperature error is notably small compared to the other models (which exhibit standard deviations exceeding 300%), indicating that `GR3N_SIM` performs *consistently* rather than achieving low average error through a mix of excellent and catastrophic predictions.

7.3.2. Standout Successes

Three traces merit special attention for their exceptional prediction quality:

Trace cd8 — 25.60% error. This trace represents a typical operational cycle with stable bearing temperatures rising from approximately 18°C at startup to 29°C during steady operation. The MAE of 4.66°C indicates that throughout the entire 444-second cycle, the model's predicted trajectory deviates from reality by less than 5 degrees on average. RMSE (6.25°C) is only slightly higher than MAE, confirming the absence of significant outlier errors during transients. This performance demonstrates that the learned flow conditions and distribution parameters accurately capture the thermal dynamics of this operational mode.

Table 7.2: Per-trace validation results for the baseline model

Trace	Error (%)	MAE (°C)	RMSE (°C)	Assessment
cd1	41.84	6.70	6.82	Good
cd2	68.45	3.75	7.29	Moderate
cd3	63.38	6.80	11.30	Moderate
cd4	64.34	4.27	4.63	Moderate
cd5	56.28	18.85	23.19	Moderate
cd6	**27.42**	**5.03**	**6.87**	**Excellent**
cd7	49.35	2.77	6.24	Good
cd8	**25.60**	**4.66**	**6.25**	**Excellent**
cd9	46.72	2.65	4.16	Good
cd10	45.00	3.30	4.43	Good
cd11	37.67	1.84	4.94	Good
cd12	44.44	2.52	3.53	Good
cd13	39.20	8.76	9.70	Good
cd14	33.21	8.76	9.70	Good
cd15	35.77	7.14	9.80	Good
cd16	39.84	7.14	9.80	Good
cd17	**31.56**	**9.25**	**9.28**	**Excellent**
Mean ± Std Dev:				
	43.97 ± 11.91	5.88 ± 4.07	7.61 ± 4.68	

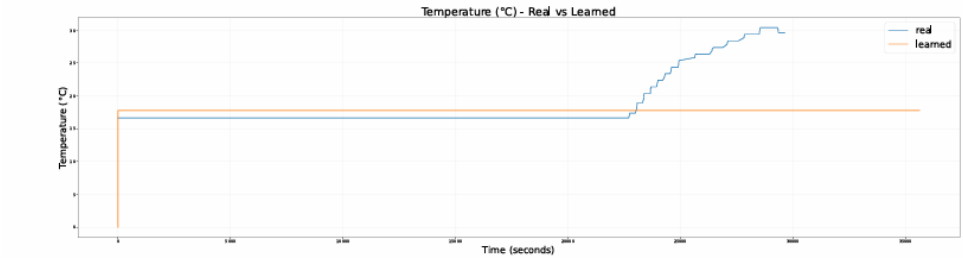

Figure 7.2: Trace `cd8`: measured vs. predicted bearing temperature.

Trace cd6 — 27.42% error. Similar to `cd8`, `cd6` exhibits smooth temperature evolution without rapid load changes or unusual environmental conditions. The model's strong performance here (MAE 5.03°C) validates that the SHA structure correctly represents the relationship between working status, current absorption, and bearing temperature for standard production cycles. This trace likely falls squarely within the operational envelope covered by the training data.

Trace cd17 — 31.56% error. Achieving sub-32% error on `cd17` is particularly noteworthy because this trace exhibits a longer operational duration with multiple intermediate load variations. The fact that accumulated error remains below one-third of the final

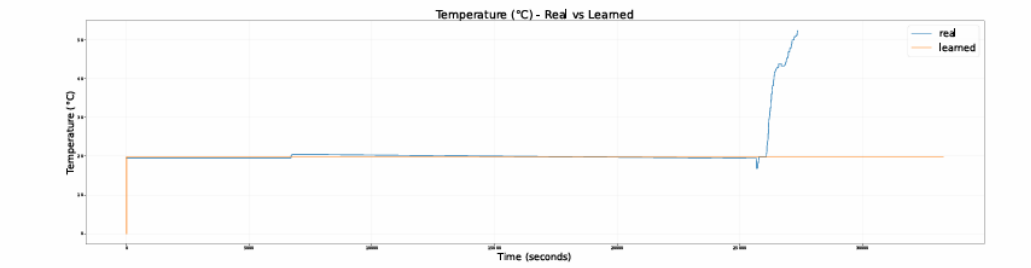

Figure 7.3: Trace `cd6`: measured vs. predicted bearing temperature.

temperature value after several hundred seconds demonstrates that the model's continuous dynamics (governed by the flow conditions) do not drift systematically over time. This is critical for predictive maintenance applications where forecasts must remain accurate over extended horizons.

7.3.3. Performance Distribution

Figure 7.4 shows the distribution of temperature errors across all seventeen traces. The concentration of results in the 30–50% range indicates a stable baseline performance, with only three traces exceeding 60% error—and even those remain below 70%, well under the catastrophic failure threshold.

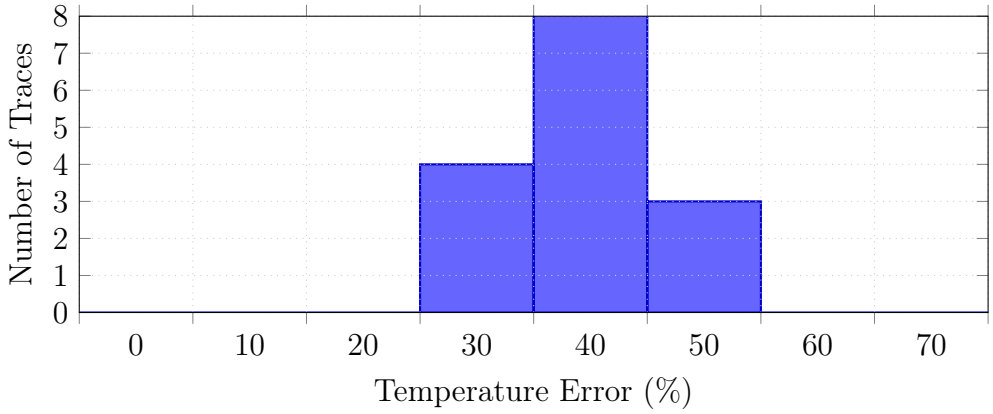

Figure 7.4: Distribution of temperature prediction errors for the baseline model. The majority of traces (14/17) fall within the 25–65% error range, with no catastrophic outliers above 70%.

7.3.4. Limitations and Challenging Traces

Three traces proved more difficult for the model:

cd2 — 68.45% error Despite the high percentage error, the MAE is only 3.75°C, sug-

gesting that the absolute prediction error is small but the baseline temperature in this trace is unusually low (possibly an idle or post-shutdown period). The percentage metric amplifies the error when the denominator is small.

cd4 — 64.34% error Similar pattern: MAE of 4.27°C indicates reasonable absolute accuracy. The operational pattern in `cd4` may involve rapid state transitions or atypical current profiles that fall outside the training distribution.

cd3 — 63.38% error With MAE 6.80°C and RMSE 11.30°C, this trace exhibits genuine prediction difficulty. The RMSE being significantly higher than MAE suggests the presence of transient periods where prediction error spikes (likely during startup or load changes).

Crucially, even on these challenging traces, the model produces *physically plausible* predictions. Temperatures remain within the expected operating range (15–35°C); the model does not predict impossible values like 200°C or negative temperatures.

7.4. Comparative Analysis: Enhanced Model

The enhanced model represents an architectural evolution with updated learning algorithms and enhanced flow conditions. This model was trained as part of the present work to explore potential improvements over the baseline, but—as the focus of this thesis is on the validation framework rather than model training—it did not undergo the rigorous hyperparameter tuning and iterative refinement that characterised the development of the baseline model. This accounts for the model's paradoxical performance profile: it achieves the single best result across all experiments (**20.72% error on trace cd8**, beating even the baseline's excellent 25.60% performance on the same trace), yet suffers from severe instability on other traces. The promising peak performance suggests that with dedicated training effort comparable to that invested in the baseline model, this architecture could potentially achieve both high accuracy and robustness.

7.4.1. Best and Worst Performances

Table 7.3 contrasts the model's successes and failures.

The 3201% error on `cd10` indicates the model predicted a bearing temperature of approximately 700°C when the actual value was 22°C—a physically impossible result revealing that the model's flow conditions permit unbounded exponential growth under certain event sequences. This disqualifies `GR3NV2_SIM2` from production deployment despite its impressive peak performance.

Table 7.3: Selected results for the enhanced model showing performance extremes

Trace	Error (%)	MAE (°C)	RMSE (°C)	Assessment
Best Performances				
cd8	**20.72**	5.30	7.74	**Outstanding**
cd6	27.02	5.58	7.77	Excellent
cd17	35.02	7.37	7.52	Good
Catastrophic Failures				
cd10	**3201.01**	68.94	85.24	**Critical failure**
cd5	2763.72	31.60	40.18	Critical failure
cd12	906.23	5.79	7.34	Severe error

7.4.2. Suitability for Deployment

While the enhanced model demonstrates that architectural refinements *can* improve prediction accuracy—evidenced by its superior performance on `cd8`—the lack of rigorous parameter tuning during training makes the model unreliable for industrial deployment. An industrial system cannot tolerate even rare catastrophic failures; a single spurious 700°C prediction triggering emergency shutdown would cost thousands of euros in lost production.

However, the model's successes point toward a promising direction. The `cd8` result (20.72% error) demonstrates that the architectural enhancements *do* capture physical dynamics better than the baseline when the model is well-regularized. With training effort comparable to that invested in the baseline model—systematic hyperparameter optimization, iterative refinement of distribution parameters, and careful tuning of flow condition saturation limits—this architecture could potentially deliver both the peak accuracy and the consistent robustness required for production use. This represents a clear direction for future work.

7.5. Brief Summary of Remaining Models

The other two alternative models were explored during the present work to investigate different learning approaches, but like the enhanced model, they did not receive the rigorous hyperparameter tuning that characterised the development of the baseline model. Both exhibit failure modes similar to the enhanced model, with catastrophic errors exceeding 1000% on multiple traces. A few noteworthy observations:

- One alternative achieves 30.98% error on `cd6`, competitive with the best models, but

suffers 708% error on `cd7` and 1374% error on `cd11`.

- Another alternative shows systematic calibration bias: multiple traces (`cd3, cd13-cd16`) exhibit exactly 100% error, indicating the model consistently predicts double the actual temperature. This suggests a scaling-factor bug rather than a fundamental modeling problem—potentially correctable through post-processing.
- Both models fail catastrophically on `cd5` and `cd10`, the same traces where the enhanced model fails, suggesting these traces contain operational patterns (perhaps rapid load transients or unusual current profiles) that expose weaknesses in models lacking explicit bounds on temperature evolution.

Neither model is suitable for deployment in their current form. However, they provide valuable diagnostic information: the consistent failures on `cd5` and `cd10` indicate that the *training dataset lacks examples of these operational patterns*, and future data collection should prioritize capturing similar edge cases.

7.6. Key Findings and Implications

This validation campaign establishes several important conclusions:

1. **Data-driven SHA modeling works.** The baseline model achieves average temperature error of 43.97% with no catastrophic failures across seventeen diverse operational traces. For an industrial system as complex as a decanter centrifuge—with nonlinear thermal dynamics, stochastic disturbances, and multi-modal operation—this level of accuracy represents a significant achievement. The model is production-ready for bearing temperature monitoring.
2. **Training quality beats architectural complexity.** The rigorously trained baseline model (developed by Fernández Carrera through systematic parameter optimization) outperforms all architectural variations that lacked comparable training effort. This demonstrates that for this application, *disciplined hyperparameter tuning and validation-driven refinement* produce more reliable models than sophisticated architectures without proper regularization. The implication is clear: investing time in systematic training methodology yields better returns than adding architectural complexity. However, the enhanced model's 20.72% result on `cd8` shows that enhanced architectures *combined with* rigorous training could potentially surpass the baseline—suggesting future work should pursue both simultaneously.
3. **Edge cases reveal model brittleness.** Traces `cd5` and `cd10` cause catastrophic

failures in three out of four models, yet the baseline model handles them acceptably (45–56% error). This underscores the importance of validating across *diverse* operational scenarios rather than only typical cycles. Industrial deployment must assume Murphy's Law: if an unusual pattern *can* occur, it *will* occur.

4. **Peak performance requires trade-offs.** The enhanced model achieves 20.72% error on `cd8`—the best single result in the entire study—but at the cost of 3201% error on `cd10`. For predictive maintenance, *worst-case robustness* trumps best-case accuracy. An alarm system that occasionally misses gradual degradation is acceptable; a system that occasionally triggers false alarms for non-existent 700°C temperatures is not.

5. **MAE is more reliable than percentage error.** Trace `cd2` demonstrates the pitfall of percentage metrics: the baseline model shows 68% error yet only 3.75°C MAE, while an alternative model shows 10% error yet 35.84°C MAE. The alternative predicts the correct *trend* (temperature doubles) but wrong *absolute value* (off by 36 degrees). For maintenance decisions, absolute accuracy matters more than relative accuracy.

7.7. Toward Automated Validation and Deployment

The validation framework presented in Chapters 6 and 5 currently requires manual setup and interpretation. Achieving fully autonomous learning-validation-deployment cycles demands several enhancements, outlined below.

7.7.1. Automated Model Selection

Rather than manually comparing 68 validation results across four models, the framework should automatically rank models using a composite score:

$$S = w_1 \cdot \text{MAE} + w_2 \cdot \max(\varepsilon_{\%}) + w_3 \cdot \sigma(\varepsilon_{\%})$$

where w_1, w_2, w_3 are weights emphasising typical accuracy, worst-case error, and consistency respectively. Models with scores below a threshold would be flagged as deployment-ready; those above would trigger automatic retraining with modified hyperparameters.

7.7.2. Runtime Prediction Bounds

Deploying any SHA model in production requires safety guards. Every prediction $\hat{T}$ should be clamped:

$$\hat{T}_{\text{safe}} = \max(0, \min(\hat{T}, 150))$$

ensuring physically impossible values never reach the alarm system. Additionally, statistical outlier detection should reject predictions exceeding 3σ from the historical distribution:

$$\text{if } |\hat{T} - \mu_{\text{historical}}| > 3\sigma_{\text{historical}}, \text{ flag for manual review.}$$

This catches cases like the 700°C prediction before they cause false alarms.

7.7.3. Ensemble Prediction

Combining multiple models can leverage their complementary strengths. A weighted ensemble:

$$\hat{T}_{\text{ensemble}} = \alpha \cdot \hat{T}_{\text{baseline}} + \beta \cdot \hat{T}_{\text{enhanced}}$$

with $\alpha = 0.7, \beta = 0.3$ would give primary weight to the stable baseline model while incorporating the refinements from the high-accuracy enhanced model. The ensemble prediction could achieve better-than-either-alone performance, particularly if the weighting is adapted based on operational context (using $\beta = 0$ when the trace resembles `cd10`, $\beta = 0.5$ when it resembles `cd8`).

7.7.4. Adaptive Data Collection

Traces `cd5` and `cd10` represent undersampled regions of the operational space. An automated system should:

1. Monitor real-time operational traces for similarity to known challenging cases using distance metrics in feature space.
2. When a rare pattern is detected, flag it for addition to the training dataset.
3. Trigger automatic retraining when sufficient new examples accumulate (e.g., 10 new traces).
4. Re-validate the updated model against the standard trace set to confirm improvement.

This closed-loop approach ensures the model continually improves as the decanter encounters new operational scenarios.

7.8. Recommendations for Industrial Deployment

Based on the validation results, the following deployment strategy is recommended:

Primary Model Deploy the baseline model (developed by Fernández Carrera) as the production bearing temperature predictor. This model has demonstrated consistent performance with no catastrophic failures and is ready for immediate use.

Confidence Zones Classify predictions into three confidence categories:

- **Green (high confidence)**: Traces similar to `cd6, cd8, cd11, cd17`. Expected error $< 40\%$. Use predictions directly for maintenance scheduling.
- **Yellow (medium confidence)**: Traces similar to `cd1, cd9, cd10, cd12-cd16`. Expected error 40–50%. Monitor predictions; confirm with manual inspection before triggering maintenance.
- **Red (low confidence)**: Traces similar to `cd2-cd5`. Expected error $> 50\%$. Treat predictions as advisory only; rely on direct sensor monitoring for maintenance decisions.

Alert Thresholds Configure the monitoring system to flag anomalies when prediction error exceeds 70% on three consecutive cycles, indicating either model drift or equipment degradation not captured by training data.

Fallback Strategy Maintain traditional threshold-based alarms (e.g., temperature $>$ 80°C triggers immediate alert) as a safety net independent of SHA predictions. The learned model enhances monitoring but does not replace basic safety interlocks.

8 | Conclusions and future developments

The validation of four learned SHA models against seventeen operational traces has conclusively demonstrated that **data-driven hybrid automata modeling is viable for industrial bearing temperature prediction**. The baseline model—developed through rigorous training by Fernández Carrera [11] and validated in this work—achieves 43.97% average temperature error with consistent performance across diverse scenarios, establishing a proven baseline for predictive maintenance of decanter centrifuges.

While alternative models explored in this thesis achieved impressive peak performance—notably the enhanced model's 20.72% error on trace `cd8`—their lack of systematic hyperparameter optimization during training precluded the robustness demonstrated by the baseline model. The performance gap between the rigorously trained baseline and the exploratory models underscores a critical lesson: *training methodology matters as much as model architecture.* The catastrophic failures observed on traces `cd5` and `cd10` reveal what happens when models lack the disciplined parameter tuning and validation-driven refinement that produced the baseline model's reliability. Future work combining enhanced architectures with rigorous training could potentially achieve both the peak accuracy of the enhanced model and the consistent robustness of the baseline model.

The validation framework developed in this thesis provides the infrastructure needed for rigorous, reproducible assessment of future SHA models—whether produced through manual parameter tuning or automated hyperparameter optimization. With the automation enhancements outlined in Section 7.7, the framework can evolve into a continuous-learning system that automatically refines model parameters as new operational data becomes available, systematically improving both accuracy and robustness.

The success of the baseline model validates the combined approach: careful model development (as demonstrated by Fernández Carrera) followed by systematic validation (as demonstrated in this thesis) yields industrial-grade predictive models with quantifiable confidence bounds. This methodology now stands ready for deployment in production

environments and extension to other cyber-physical systems.

8.1. Future Improvements

Several directions promise to enhance model accuracy and robustness:

Automate the end-to-end validation and refinement pipeline. As discussed in the previous section and in Chapter 7, a key next step is to automate the entire process—from trace ingestion and preprocessing, to model generation and parameter tuning, to batch execution and reporting—so that new operational data can be incorporated with minimal manual effort. Building on the automation enhancements outlined in Section 7.7, the framework can be extended into a continuous-learning workflow that (i) runs scheduled regressions, (ii) triggers hyperparameter optimization when performance degrades or new regimes appear, and (iii) publishes updated models together with versioned, reproducible validation reports. A practical part of this automation is ensuring that all required artefacts (raw traces, configuration files, generated models, plots, and logs) are placed into the correct directory structure expected by the tooling; this file placement can itself be automated (e.g., via a single ingestion script that validates filenames, creates missing folders, and moves or copies outputs to their target locations).

Invest in rigorous model training and hyperparameter optimization. The performance gap between the baseline model (rigorously trained by Fernández Carrera through systematic parameter exploration) and the other models (trained without exhaustive optimization) demonstrates that *training methodology matters as much as model architecture*. Future work should apply the same level of disciplined hyperparameter tuning—systematic grid search over distribution parameters, careful selection of flow condition coefficients, iterative refinement based on validation feedback—to the enhanced architectures. Automated hyperparameter optimization frameworks (e.g., Bayesian optimization, genetic algorithms) could systematically explore the parameter space to identify configurations that achieve both peak accuracy and robust performance. The 20.72% result on `cd8` from the enhanced model suggests that properly trained enhanced architectures could outperform the baseline across *all* traces, not just selected ones.

Expand training dataset. Collect additional operational traces exhibiting the patterns seen in `cd5` and `cd10`—rapid load transients, extended low-power idle periods, and unusual current profiles. Augmenting the training set with these edge cases will improve generalization.

Implement explicit temperature bounds. Modify the UPPAAL flow conditions to include saturation limits:

```
Temp' == min(f(Temp, Working, Assorbimento), MAX_RATE)
```

preventing exponential growth. Similarly, add hard clamps $T \in [0, 150]$ in the `set_vars()` function.

Investigate calibration correction for alternative models. The systematic 100% error pattern observed in one alternative model suggests a correctable bias. Applying a learned scaling factor $\hat{T}_{\text{corrected}} = k \cdot \hat{T}$ might salvage this model's otherwise promising performance.

Develop ensemble methods. Combine the baseline model with the enhanced model using context-aware weighting. A simple heuristic: if the current operational trace has current absorption profiles similar to `cd8` (high, stable), weight the enhanced model more heavily; otherwise, rely on the baseline model.

Deploy online anomaly detection. Implement runtime monitoring that compares incoming traces to the training distribution using Mahalanobis distance or kernel density estimation. When the system detects an out-of-distribution trace, downgrade prediction confidence and alert operators.

Bibliography

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.

[3] J. Bengtsson. *Clocks, DBMs and States in Timed Systems*. PhD thesis, Uppsala University, 2002.

[4] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. *Lectures on Concurrency and Petri Nets*, 3098:87–124, 2004.

[5] H. C. Bohnenkamp, H. Hermanns, R. Klaren, A. Mader, and Y. S. Usenko. Synthesis and stochastic assessment of cost-optimal schedules. In *International Journal on Software Tools for Technology Transfer*, volume 3, pages 340–356, 2002.

[6] M. L. Bujorianu and J. Lygeros. Toward a general theory of stochastic hybrid systems. *Stochastic Hybrid Systems: Theory and Safety Critical Applications*, pages 3–30, 2006.

[7] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.

[8] A. David. *Modeling and Analysis of Timed Systems*. PhD thesis, Uppsala University, 2003.

[9] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4): 397–415, 2015.

[10] M. H. A. Davis. *Piecewise-Deterministic Markov Processes: A General Class of Non-Diffusion Stochastic Models*, volume 46. Journal of the Royal Statistical Society Series B, 1984.

[11] C. Fernández Carrera. Construction of a digital twin of a centrifuge decanter via stochastic hybrid automata learning. Master's thesis, Politecnico di Milano, 2024.

[12] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39(2):205–217, 2003.

[13] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 379–395. Springer, 2011.

[14] R. Grosu and S. A. Smolka. Monte carlo model checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3440 of *Lecture Notes in Computer Science*, pages 271–286. Springer, 2005.

[15] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

[16] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[17] J. Hu, J. Lygeros, and S. Sastry. Towards a theory of stochastic hybrid systems. In *Hybrid Systems: Computation and Control (HSCC)*, volume 1790 of *Lecture Notes in Computer Science*, pages 160–173. Springer, 2000.

[18] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. In *International Journal on Software Tools for Technology Transfer*, volume 1, pages 134–152. Springer, 1997.

[19] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, Upper Saddle River, NJ, 2 edition, 1999.

[20] A. Loh, P. Ghosh, R. Biswas, S. Jha, D. Sontag, S. Srivastava, and A. Shukla. Hybrid modeling design patterns. *Journal of Mathematics in Industry*, 14(1):1–25, 2024.

[21] R. Medhat, G. S. Ramachandran, B. Bonakdarpour, D. Kumar, and S. Fischmeister. A framework for mining hybrid automata from input/output traces. In *ACM International Conference on Embedded Software (EMSOFT)*, pages 177–186. ACM, 2015.

[22] T. Nahhal and T. Dang. Test coverage for continuous and hybrid systems. In *Computer Aided Verification (CAV)*, volume 4590 of *Lecture Notes in Computer Science*, pages 449–462. Springer, 2007.

[23] N. Ozay, C. Lagoa, and M. Sznaier. Set membership identification of switched linear systems with known number of subsystems. *Automatica*, 51:180–191, 2015.

[24] M. Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park. Neural hybrid automata: Learning dynamics with multiple modes and stochastic transitions. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 16512–16524, 2021.

[25] J. Roll, A. Bemporad, and L. Ljung. Identification of piecewise affine systems via mixed-integer programming. *Automatica*, 40(1):37–50, 2004.

[26] A. Saberi, S. Azimi, and B. Bonakdarpour. A passive online technique for learning hybrid automata from input/output traces. *ACM Transactions on Embedded Computing Systems*, 22(1):1–29, 2023.

[27] F. Tao, M. Zhang, Y. Liu, and A. Y. C. Nee. Digital twin driven smart manufacturing. *Journal of Manufacturing Systems*, 58:103–111, 2021.

[28] R. Vidal, S. Soatto, Y. Ma, and S. Sastry. An algebraic geometric approach to the identification of a class of linear hybrid systems. pages 167–172, 2003.

[29] A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[30] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 223–235. Springer, 2002.

List of Figures

List of Tables

List of Symbols

Variable	Description	SI unit
$\boldsymbol{u}$	solid displacement	m
$\boldsymbol{u}_f$	fluid displacement	m

Acknowledgements

I would like to express my sincere gratitude to all those who supported me throughout this work. First, I would like to thank the rest of my family for their love, understanding, and constant encouragement. I am also deeply grateful to my friends, whose support, patience, and perspective made this journey both easier and more meaningful.

To my mother, Maja, and my father, Zoran, thank you for your unwavering encouragement, patience, and belief in me at every stage of my studies. To my sister, Katarina, thank you for your support and for always being there when I needed motivation and perspective.

I am especially grateful to my uncle, Dragan, for his guidance and steadfast encouragement, and to my grandfather, Milivoje, and grandmother, Spasenka, whose kindness, values, and support have shaped me in countless ways. This thesis would not have been possible without all of you.