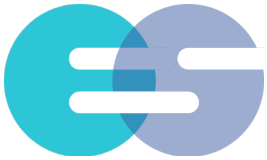
TUHH
Hamburg
University of
Technology



Integration of Hybrid System Identification using Symbolic Regression into a Modeling Framework for Cyber-Physical Systems

Projectwork

by

Harshitha Viswanath

Institute of Embedded Systems
Hamburg University of Technology

September 11, 2025

Examiner: Prof. Dr.-Ing. Görschwin Fey
Supervisor: Swantje Plambeck

Contents

1 Introduction

A Cyber-Physical System (CPS) is an integrated system that combines computation, communication, and physical processes. In such systems, embedded computing elements monitor and control physical state via feedback loops. As defined in (1; 2), a CPS involves deep, ongoing interdependence between computational intelligence and real-world dynamics.

A CPS consists of tightly integrated computational and physical components that interact in real time. Their architecture typically includes a central computational and communication core that continuously senses, processes, and actuates on physical processes. These systems are designed specifically to operate within and directly influence physical environments, enabling real-time decision-making and control.

The role of a CPS has become essential in a wide range of application areas, particularly where safety, precision, and responsiveness are critical. Key domains include:

- **Smart Grids:** The physical domain consists of consumer electrical appliances, while the cyber domain includes centralized independent system operators that compute real-time electricity prices based on demand. Bidirectional communication enables dynamic control and adaptive decision-making
- **Healthcare and Biomedical Devices:** Intelligent medical devices monitor patient physiological data in real time. Cyber components provide automated interventions or augmented support, enhancing patient care and outcomes
- **Automotive and Aerospace Systems:** Cyber-physical systems underpin autonomous vehicles, advanced driver-assistance, and critical aerospace functions like flight control and threat detection, where real-time sensing and control are essential

One of the distinguishing features of CPS is their hybrid nature,the ability to combine discrete, high-performance computation with the continuous, dynamic behaviors of physical systems. In the physical world, time progresses continuously, and systems operate in parallel. Consequently, the cyber components must be designed to interact seamlessly with changing physical variables, often under strict temporal constraints.

As highlighted by (3), the complexity of CPS introduces several engineering challenges. These include the need for precise models to handle environmental noise and uncertainty, strategies to manage asynchronous behavior between digital and physical components, support for multi-scale system dynamics, and a transition from ad hoc design to mathematically verifiable development. Additionally, CPS demands formal analysis over test-heavy methods and fault-tolerant architectures capable of adapting to unpredictable conditions. Together, these demands underscore the necessity for rigorous, principled approaches to CPS design.

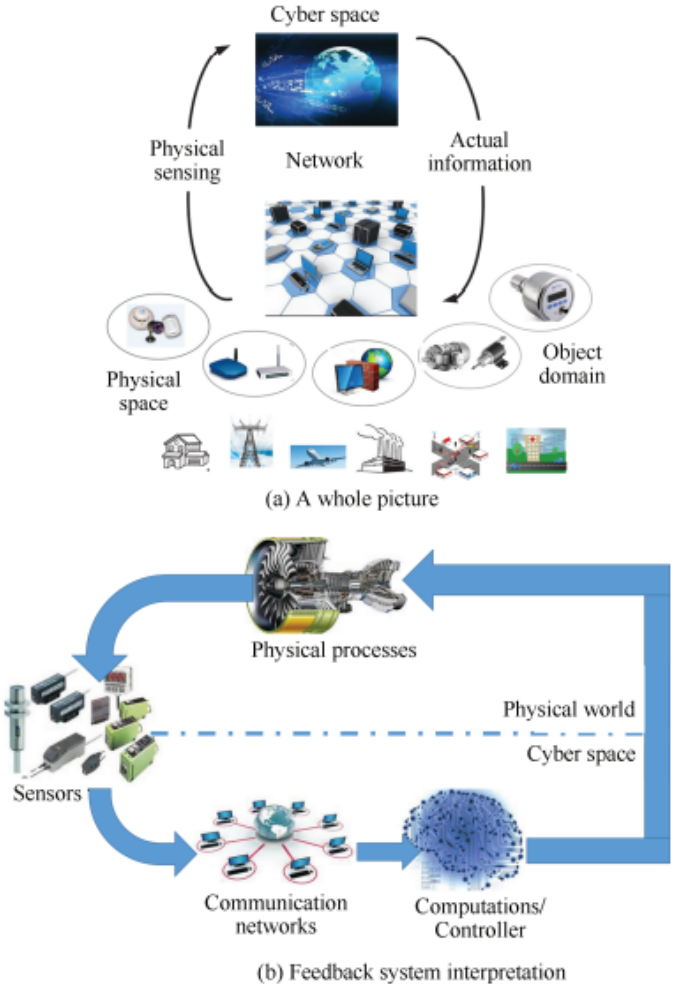


Figure 1: CPS Applications and Closed-Loop Interpretations

Hybrid system models, widely used for CPS abstraction, represent the system as a finite set of discrete modes, each with its own continuous dynamics (flow functions), and transitions triggered by guard conditions. Deriving accurate and interpretable models for such systems from real-world data remains a significant challenge.

The work in (4) proposes a novel Symbolic Regression (SR) approach to learn hybrid system models directly from data. Using Genetic Programming (GP), SR identifies both continuous dynamics and discrete transitions without prior structural assumptions. In contrast to the traditional trajectory-based methods, SR infers mode transitions from changes in system dynamics rather than spatial proximity.

The resulting models are concise, interpretable, and predictive, enabling rich system insight. In this project, SR is integrated into the Flowcean CPS identification framework, enhancing its modeling pipeline by reducing dependence on manual or black-box models. This integration supports verifiable, modular CPS design by aligning interpretability with automation and performance.

2 Problem Statement

2.1 The Need for Hybrid System Modeling in CPS

CPS integrates analog components, such as sensors and actuators, interconnected through wired or wireless networks, and governed by real-time cyber control. Their defining characteristic is the tight coupling between discrete decision-making logic and the continuous behavior of physical processes. CPS are increasingly deployed in safety-critical domains such as autonomous vehicles, medical devices, and industrial automation, where design flaws can lead to catastrophic consequences. As noted by (5), the growing complexity and autonomy of CPS amplify the challenges in ensuring correctness during both system design and verification. Achieving functional correctness requires not only reliable discrete control but also accurate modeling of continuous dynamics.

The hybrid automata framework proposed by (6) effectively bridges these domains by combining finite-state logic with real-valued, time-evolving variables. State transitions are triggered by guard conditions on these variables, enabling discrete mode shifts alongside continuous evolution. This makes hybrid automata particularly well-suited for representing CPS, offering a rigorous and formal structure for modeling, verification, and analysis. Their semantic precision supports robust and error-resistant CPS development.

Traditional system modeling approaches often rely on first-principles equations derived manually by domain experts. While effective in specific contexts, these methods struggle to handle highly nonlinear or hybrid systems where interactions are complex and difficult to express analytically. Moreover, such models are typically rigid; once constructed, they are difficult to adapt to new data or evolving system behaviors. As CPS grows larger and more integrated, this lack of scalability, flexibility, and automation becomes a significant bottleneck.

To address these challenges, researchers increasingly adopt data-driven system identification methods. Unlike first-principles based modeling, these approaches learn system dynamics directly from observed input-output data, avoiding the need for explicit knowledge of governing equations. This shift not only accelerates the modeling process but also enables the discovery of subtle and nonlinear behaviors that are difficult to capture analytically. In CPS, where components span digital controllers, actuators, and physical processes, data-driven modeling offers a powerful alternative, uncovering hidden relationships and improving automation in hybrid system modeling.

2.2 Symbolic Regression for Hybrid System Identification

SR bridges the gap between traditional analytical modeling and black-box machine learning approaches. Unlike neural networks or other opaque models, SR produces human-readable mathematical expressions that describe how system variables interact. This interpretability is critical in safety-critical CPS domains, where verification, reliability, and explainability are essential.

A key advantage of SR is its ability to model hybrid dynamics by learning both continuous governing equations and capturing discrete mode transitions. This makes SR

particularly suitable for identifying hybrid systems where mixed dynamics are prevalent. Among available SR tools, *PySR* stands out as a powerful choice, offering robust capabilities for modeling noisy, high-dimensional scientific datasets while maintaining interpretable results.

This project focuses on integrating an existing SR-based hybrid system identification approach proposed by (7) into the Flowcean framework ((8)), resulting in a unified toolchain for CPS identification. The goal is to develop an SR learner and model that align with Flowcean's modular architecture and to introduce a suitable evaluation metric to assess performance.

It is important to note that the primary focus is not on improving the original SR algorithms or enhancing the accuracy of the Flowcean framework. Instead, this work emphasizes seamless architectural integration, ensuring that SR components operate efficiently within Flowcean's CPS modeling pipeline. The case studies presented demonstrate structural requirements for building a complete CPS modeling workflow, highlighting the need for explicit, interoperable, and automated toolchains. Ultimately, the emphasis lies in validating the integration and workflow consistency rather than optimizing predictive precision alone.

3 Related Work

3.1 Overview

CPS integrate computational components with physical processes, requiring modeling approaches that capture both continuous dynamics and discrete event-driven behavior. This duality motivates research into both general CPS modeling paradigms and specialized hybrid system identification techniques.

3.2 Modeling of CPS

Reliable modeling is essential for understanding CPS behavior, predicting failures, and improving design((9)). According to (10), three major approaches exist: analytical, simulation, and hybrid models.

Analytical models estimate system dependability using mathematical and probabilistic techniques such as game theory, queuing models, fault trees, and reliability block diagrams((11)). They are computationally efficient but rely on simplifying assumptions. For example, (12) applies a cold-standby redundancy framework, modeling failure and repair transitions via a continuous-time Markov birth-death process with exponential failure assumptions. The Chapman-Kolmogorov equations are solved using the 4th-order Runge-Kutta method, showing that redundant subsystems sustain high reliability, while non-redundant components degrade sharply with small parameter changes. Although effective, the model assumes independent, identically distributed failures and perfect redundancy switching but can be extended with realistic failure distributions and mixed-redundancy strategies.

Simulation models are used when analytical solutions cannot capture complex CPS interactions. They employ computational techniques, including agent-based((13)), discrete-event((14)), Monte Carlo((15)), and continuous-time simulation((16)). In (12), CPS within virtual power plants are modeled as a multi-layered graph, where nodes represent components and edges represent dependencies. A vulnerability index combines electrical and cyber metrics, and cascading failures are simulated under faults and attacks. Repeated runs yield statistical reliability indicators similar to Monte Carlo sampling, providing a scalable and efficient alternative to resource-intensive co-simulation methods.

Hybrid models integrate analytical precision with simulation flexibility for accurate and scalable reliability analysis. Examples include co-simulation((17)), hybrid automata((18)), and hybrid bond graphs((19)). For instance, (20) proposes a hybrid approach combining Monte Carlo Simulation (MCS) and Artificial Neural Networks (ANN) for composite power systems. MCS generates system states and reliability metrics such as Loss of Load Expectation (LOLE), Loss of Load Probability (LOLP), Expected Energy Not Supplied (EENS), Probability of Load Curtailment (PLC), and Expected Interruption Rate (EIR). These data train an ANN, which predicts reliability indices for new configurations more efficiently than repeated MCS runs, combining the robustness of MCS with the speed and generalization capability of ANN for large-scale assessments.

While analytical, simulation, and hybrid approaches offer high-level frameworks for CPS reliability modeling, deeper insights arise from examining the underlying modeling

paradigms, which range from physics-based formulations and state-driven logic to intelligent agent-based methods and data-centric techniques((21)). Figure 2 illustrates these paradigms.

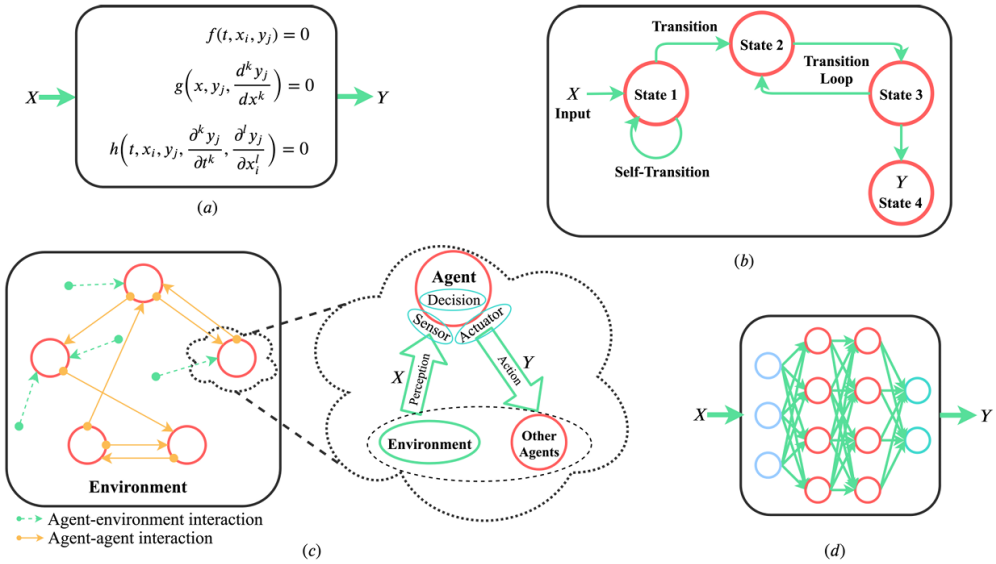


Figure 2: CPS Modeling Paradigms: (a) Physics-Based Equations; (b) State Machines; (c) Rule- and Agent-Based Models; (d) Data-Driven Models

Physics Equation-Based Modeling

Physical processes in CPS are modeled using algebraic relationships, ordinary differential equations (ODEs), and partial differential equations (PDEs), expressed generally as $f(t, x_i, y_j) = 0$ for time-dependent variables x_i and y_j. These models, grounded in physical laws, ensure properties such as causality, conservation, and objectivity. While highly interpretable and accurate, they struggle with uncertain environments, nonlinearities, and multi-domain CPS interactions.

State Machine-Based Modeling

Finite State Machines (FSMs) are widely used to represent discrete-event behaviors through a formal tuple (States, Inputs, Outputs, Update, InitialState). Mealy and Moore machines distinguish between state-driven and transition-driven outputs. FSMs effectively capture control logic and mode switching but face scalability issues when modeling complex, continuous–discrete interactions in CPS.

Rule- and Agent-Based Modeling

Agent-Based Models (ABMs) extend FSMs by introducing autonomous agents that interact with each other and the environment based on predefined rules. Well-suited for heterogeneous, large-scale CPS, ABMs capture emergent behaviors and adaptability. However, they face challenges in computational scalability, model consistency, and validation for highly complex systems.

Data-Driven Modeling

Data-driven approaches learn system behavior directly from observations, bypassing explicit physics. Earlier methods include Bayesian learning, symbolic regression, and spline-based models, while recent advances leverage deep neural networks (DNNs) to capture complex nonlinear dependencies. Despite flexibility, purely data-driven models suffer from limited generalization, poor interpretability, and difficulty in enforcing physical constraints. Consequently, hybrid strategies combining machine learning with physics-based insights are increasingly adopted for robust CPS modeling.

Modeling Tools and Environments

CPS modeling often relies on specialized simulation tools, with *Modelica* and *Simulink/MATLAB* widely used for multi-domain co-simulation.

(22) highlight Modelica, an open-source, object-oriented, equation-based language designed for multi-domain CPS. Its acausal modeling paradigm enables seamless integration of physical components (mechanical, electrical, thermal) with control logic, reducing inconsistencies from separate simulation tools.

Figure 3 illustrates the Modelica-based system modeling cycle, integrating three core components:

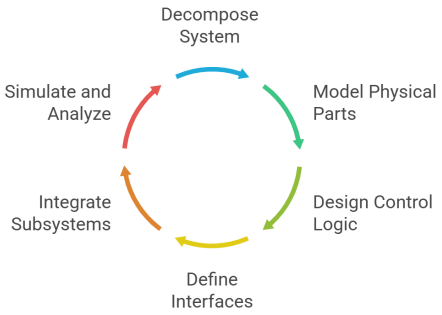


Figure 3: System Modeling Cycle using Modelica

- *Physical Modeling:* Represents continuous processes such as velocity, voltage, temperature, and force. Modelica's equation-based, acausal framework supports modularity and component reuse.
- *Cyber Modeling:* Captures decision-making, control logic, and software behavior using `when` clauses, Boolean logic, and state machines, modeling event-triggered dynamics accurately.
- *Interface Modeling:* Ensures synchronization between cyber and physical components via connectors, shared variables, event detection, and clock-driven sampling, supporting integrated hybrid simulations.

Modelica provides a unified environment combining continuous physics, discrete control, and interface dynamics, enabling high-fidelity CPS and hybrid system simulations.

3.3 Modeling of Hybrid Systems

Modeling CPS often requires capturing both continuous physical dynamics and discrete control logic, making hybrid systems a natural abstraction. Traditional parametric modeling approaches struggle with scalability and interpretability in complex CPS, motivating research into data-driven identification techniques for hybrid systems.

Recent advancements can be broadly classified into three categories: *optimization-based methods*, *symbolic regression approaches*, and *interpretable learning frameworks*. Representative works are summarized below.

Optimization-Based Identification

Dayekh et al. (23) present a method for identifying state-dependent switched nonlinear dynamical systems (SNDS) governed by polynomial ODEs, assuming the number of modes is known. The approach integrates trajectory segmentation into an optimization framework, estimating state derivatives via a five-point stencil and modeling nonlinearities with polynomial feature mappings for computational efficiency in low-dimensional systems. Mode transitions are detected by comparing forward and backward derivatives, and piecewise polynomial models are learned for each segment with regularization to control complexity. State-space partitions are recovered using logistic regression, and an active learning phase refines mode boundaries by generating counterexamples near transitions. This method is effective but relies on prior knowledge of mode count and polynomial degree, limiting its applicability in unknown or complex systems.

Symbolic Regression for Hybrid Systems

In contrast to parameter-dependent formulations, Ly et al. (24) introduce the *Multi-Modal Symbolic Regression* (MMSR) framework, which learns hybrid dynamical system structures directly from time-series data without assuming a predefined number of modes. MMSR integrates two subroutines:

- **Clustered Symbolic Regression (CSR):** Uses symbolic regression with genetic programming and the Expectation-Maximization (EM) algorithm to simultaneously cluster trajectory data and discover mode-specific expressions. Model selection is performed using the Akaike Information Criterion (AIC) to balance accuracy and interpretability.
- **Transition Modeling (TM):** Learns symbolic conditions for mode transitions by approximating switching boundaries using smooth sigmoidal functions, enabling differentiable classification of mode changes.

MMSR produces interpretable, human-readable models and has been shown to recover governing equations consistent with first-principles descriptions, making it a strong candidate for hybrid system identification in scientific domains.

Interpretable Decision-Tree-Based Modeling

(25) propose *FaMoS* (Fast Model Learning for Hybrid Cyber-Physical Systems), a framework designed for runtime efficiency, explainability, and robustness with limited data. FaMoS employs a four-stage pipeline:

- **Trace Segmentation:** Detects changes in dynamics using a sliding-window approach that compares the immediate past and future of each signal, including derivatives to capture higher-order dynamics. Peaks in the deviation function indicate changepoints, and trajectories are split accordingly.
- **Segment Clustering:** Groups similar segments into modes using a Dynamic Time Warping (DTW)-inspired similarity metric that combines alignment distance and diagonality. Optionally, clusters with overlapping dynamics are merged using Linear Matrix Inequalities (LMIs) to improve specificity. Each cluster then represents a distinct mode of the hybrid system.
- **Mode Characterization:** Fits difference equations to each cluster to define the dynamic flow, while hybrid model extraction constructs a hybrid decision tree. The tree encodes mode transitions and flow functions, using the current mode and system variables as features and the next mode as labels. The result is a compact, interpretable model capturing both continuous dynamics and discrete transitions.
- **Mode Transition Learning:** Learns interpretable decision boundaries for mode switches using Decision Tree Learning (DTL), resulting in a hybrid decision tree linking mode-specific ARX models.

Numerous process parameters, such as window size, derivative order, and clustering thresholds, are flexible, allowing adaptation to a wide range of systems. Additionally, usage of DTW provides a fast, intuitive similarity check, while LMI refinement ensures accurate identification of similar dynamics across segments. FaMoS demonstrates strong interpretability and computational efficiency, making it suitable for industrial CPS applications where transparency and runtime performance are critical.

The literature highlights two main trends. For CPS modeling, physics-based, state-machine, data-driven, and agent-based paradigms provide foundational frameworks, often supported by integrated simulation environments such as Modelica, and can be categorized as analytical, simulation, or hybrid models depending on whether they rely on mathematical formulations, computational emulations, or a combination of both. For hybrid system identification, recent approaches employ optimization, symbolic regression, and decision-tree learning to capture both continuous dynamics and discrete transitions. Building on these insights, this work integrates symbolic regression within the Flowcean framework to enable interpretable, data-driven modeling of hybrid systems.

4 Preliminaries

4.1 Overview

To understand the integration of SR into the `Flowcean` framework, it is essential to first establish the foundational concepts that underpin this work. This chapter introduces the key theoretical principles and algorithmic components that form the basis of the proposed methodology.

The discussion begins with an overview of *hybrid systems*, which formally represent the interaction between continuous dynamics and discrete control logic that are the core characteristics of modern CPS. This is followed by an introduction to SR, a data-driven technique that derives concise, interpretable mathematical models directly from observed data, making it particularly suitable for hybrid system identification. Finally, the chapter presents `PySR`, a Python-based SR library selected for its efficiency, flexibility, and strong support for generating interpretable models, which are critical for CPS analysis.

4.2 Hybrid Systems

A dynamical system evolves through states over time, governed by rules that define either continuous changes (e.g., via differential equations) or discrete updates, depending on the nature of the system variables and time.

As described by (5), dynamical systems can be broadly classified into three categories:

- *Discrete systems:* Both time and state variables evolve in discrete steps.
- *Continuous systems:* Both time and state variables change continuously.
- *Hybrid systems:* Systems that combine continuous dynamics with discrete transitions, where time progresses continuously, but discrete mode switches occur when certain conditions on the continuous variables are met.

This project focuses on **hybrid systems**, which provide a faithful abstraction of real world CPS. These systems naturally integrate continuous physical processes with discrete control logic. Modeling them solely as either discrete or continuous systems often leads to oversimplification, particularly in safety-critical domains. Hybrid models capture this dual behavior by defining *discrete modes*, each associated with its own *continuous dynamics*. Their semantic clarity and structured representation make them highly suitable for CPS design, simulation, and formal verification.

Hybrid automata are formal mathematical models that combine extended finite-state machines with continuous dynamical systems. Each discrete mode is associated with specific continuous dynamics, typically represented by ordinary differential equations (ODEs). This integration enables accurate modeling of systems exhibiting both discrete transitions and continuous evolution, making hybrid automata particularly effective for representing and analyzing cyber-physical systems.

Definition 1. *Hybrid Systems ((4)) A hybrid system can be defined as a tuple* (X, Q, F, T, Σ) *where:*

- *$X = \{x_1, x_2, \ldots, x_n\}$ is the set of system variables. Also $X = I \cup O \cup S$, where I is the set of input variables, O is the set of output variables, and S is the set of state variables. Derivatives of X can also be considered here.*

- *Q is the finite set of modes.*

- *$F = \{f_1, f_2, \ldots, f_q\}$ is the set of flow functions. A flow function defines how the state variables S and current output variables O evolve over continuous time t, based on the current values of state variables and input variables, within a given mode $q \in Q$:*

  $$[O(t), S(t)] = f_q(S(t), I(t), t)$$

  Each flow function f_q for mode $q \in Q$ is assumed to be Lipschitz-continuous, to ensure unique solutions for the ODEs representing the flow dynamics.

- *Σ is the set of events that lead to transitions between modes. Every event has a set of conditions over variables in X. A transition is triggered when these conditions are satisfied.*

- *T is the transition function between modes, defined as:*

  $$T : Q \times \Sigma \to Q$$

  A transition occurs when the respective event $\sigma \in \Sigma$ is active.

- *A hypothetical clock c is used to record time evolution. Whenever a transition occurs, the clock can be reset to handle relative time steps.*

It is evident that the hybrid system model elegantly captures both discrete and continuous behavior within a unified formalism. The continuous dynamics are governed by flow function F, typically represented as ODEs, while the discrete behavior is modeled using a finite set of modes Q and transitions T that define how the system evolves between these modes based on specific conditions.

4.3 Hybrid Automaton: Smart Sensor as a Hybrid System

The simulated system is a smart sensor modeled as a hybrid dynamical system with two discrete modes of operation: **Compression** ($q = 1$) and **Amplification** ($q = 2$). These modes determine the system's output behavior in response to the continuous input signal $u(t)$. The system evolves over time with a fixed sampling interval, and the input signal is defined as:

$$u(t) = 2.5 \cdot \sin(2\pi \cdot 0.2 \cdot t) + 2$$

This sinusoidal input causes the system to transition between modes according to threshold-based conditions.

The mode transitions are governed by a hysteresis-based switching mechanism, which prevents rapid toggling due to small fluctuations or noise in the input. The switching logic is as follows:

- The system initially starts in mode $q = 1$.
- A transition to mode $q = 2$ occurs when $u(t) \geq 2$, provided the system is currently in mode 1.
- A transition back to mode $q = 1$ occurs when $u(t) \leq 1$, provided the system is currently in mode 2.
- For values of $u(t)$ within the interval $(1, 2)$, the system retains its current mode.

This hysteresis mechanism improves stability by avoiding unnecessary mode switching caused by minor oscillations in the input. The system's output $y(t)$ depends on the current mode:

$$y(t) = \begin{cases} \sqrt{u(t) + 1}, & \text{if } q = 1 \\ 5\,u(t)^2 + 3, & \text{if } q = 2 \end{cases}$$

These mode-specific output equations produce a nonlinear and discontinuous response, representative of many real-world hybrid control systems.

System Behavior: The system starts in mode $q = 1$. At $t = 0$, the input satisfies $u(0) = 2.00 \geq 2$, triggering a switch to mode $q = 2$. While in mode 2, the system remains in this mode as long as $u(t) > 1$. It switches back to mode 1 only when $u(t) \leq 1$, which occurs near $t = 0.65\,\text{s}$ (refer to Figure 4 and Table 1).

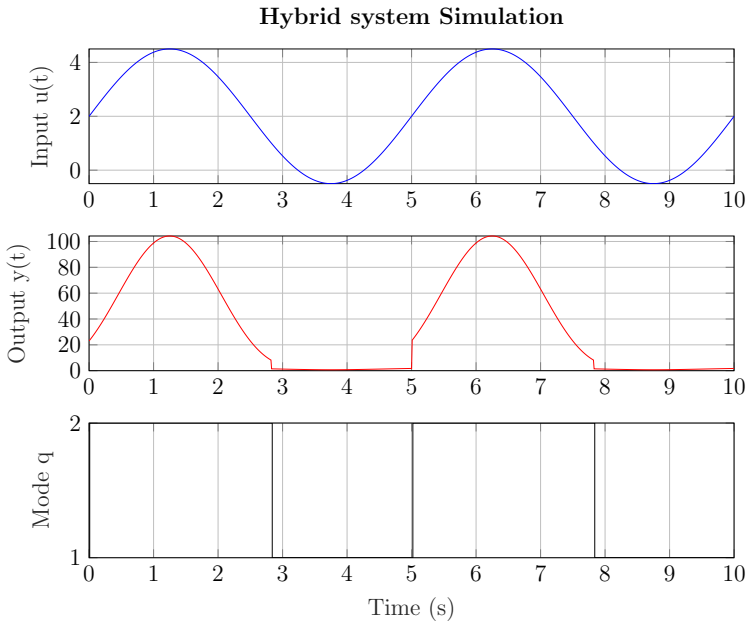


Figure 4: System Simulation

4.4 Symbolic Regression

SR is a subfield of machine learning focused on discovering interpretable mathematical expressions that describe relationships in data. Unlike black-box models such as neural

Time (s)	Input $u(t)$	Mode q Before	Mode q After	Switch
0.00	2.00	1	2	Yes
0.10	2.94	2	2	No
0.25	4.50	2	2	No
0.40	2.94	2	2	No
0.50	2.00	2	2	No
0.60	1.06	2	2	No
0.65	0.85	2	1	Yes
0.75	0.50	1	1	No
0.85	1.06	1	1	No
1.00	2.00	1	2	Yes
1.10	2.94	2	2	No

Table 1: Hybrid System Mode-Switching Behavior

networks, SR produces human-readable equations, providing both predictive accuracy and insight into system behavior. Key features of SR include:

- *Interpretability:* Generates concise, human-readable expressions that reveal underlying system dynamics.
- *Function Space Exploration:* Searches over a discrete set of symbolic expressions constructed from a predefined library of operators (e.g., $+$, $\times$, sin, log), represented as expression trees or in prefix (Polish) notation.
- *Loss-Driven Objective:* Seeks expressions that minimize a predefined loss over the training data.
- *Combinatorial Complexity:* Operates over a vast, discrete, non-convex search space, making optimization computationally challenging.

Given a dataset:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \in \mathbb{R}^d,\, y_i \in \mathbb{R},$$

SR aims to find $f \in \mathcal{F}$ that minimizes a loss function:

$$\mathcal{L}(f) = \sum_{i=1}^{n} \ell(f(x_i), y_i),$$

where $\mathcal{F}$ is the discrete function space defined by a symbolic library of elementary operations and constants, e.g., {`id()`, `add()`, `sub()`, `mul()`, +1, -1}.

Symbolic expressions are represented either as *expression trees*, where internal nodes are operations and leaves are variables/constants, or in *Polish notation*, a linear traversal of the tree suitable for compact storage and evolutionary manipulation. Figure 5 presents a taxonomy of SR methods ((26)). Here, ϕ denotes a neural network, W its parameters, $\mathbf{x}$ the input data, $\mathbf{z}$ a reduced representation, x' a transformed feature, and T the population of candidate expression trees in genetic programming.

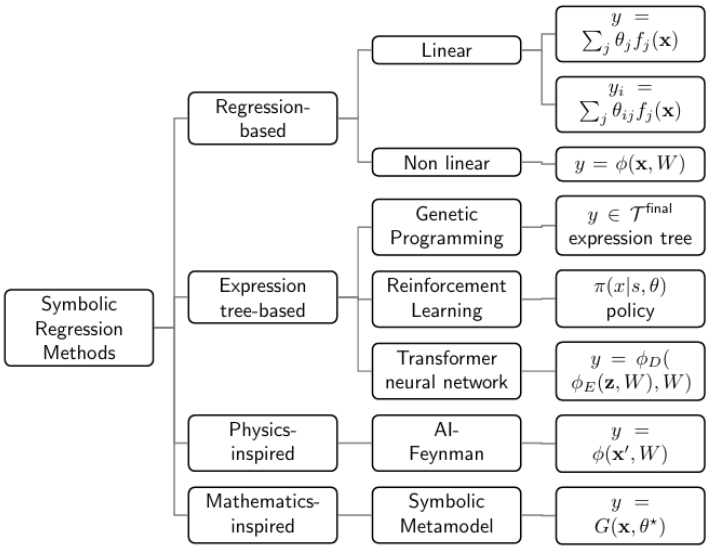


Figure 5: Taxonomy of SR

4.5 PySR: A Tool for Symbolic Regression

`PySR` is an open-source Python library for SR, designed to derive interpretable mathematical models directly from data. In scientific domains, such as quantum mechanics, manually identifying empirical relationships is impractical, and SR is NP-hard even in low-dimensional spaces.

`PySR` addresses these challenges by balancing prediction accuracy with model simplicity, handling high-dimensional datasets with real-valued constants, and supporting domain-specific scientific requirements such as physical plausibility and heteroscedastic noise ((27)). It is open-source, cross-platform, and integrates seamlessly with popular scientific ecosystems.

Core Algorithmic Enhancements

`PySR` employs a multi-population genetic programming framework, evolving multiple populations (islands) independently with periodic migrations to maintain global diversity. Key innovations include:

- *Age-Regularized Evolution:* Replaces the oldest individual rather than the least fit, preserving diversity and avoiding premature convergence.
- *Simulated Annealing-Based Selection:* Acceptance of candidates follows

$$p = \exp\left(\frac{\ell_f - \ell_e}{\alpha T}\right),$$

where ℓ_f and ℓ_e are fitness values of the mutated and original individuals, α is a scaling factor, and T is the annealed temperature. High T encourages exploration, low T favors convergence.

- *Evolve–Simplify–Optimize Loop:* The Evolve–Simplify–Optimize loop in `PySR` operates in three stages. During the *Evolve* phase, new candidate expressions are generated using genetic operations such as mutation and crossover. In the *Simplify* phase, occasional algebraic simplifications are applied to remove redundancies and produce more compact expressions. Finally, in the *Optimize* phase, real-valued constants within the expressions are fine-tuned using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) optimizer from `Optim.jl`, improving the overall model accuracy.
- *Adaptive Parsimony:* Uses a population-aware, dynamic loss to balance simplicity and predictive accuracy while encouraging diversity.

`PySR` also provides advanced features for robustness and flexibility:

- *Noise Robustness:* Gaussian process denoising of targets.
- *Weighted Fitting:* Assign weights to data points to handle noise and outliers.
- *Custom Loss Functions:* Domain-specific optimization criteria.
- *Domain-Specific Operators:* Users can define custom symbolic operators.
- *Feature Selection:* Manual or automated via gradient boosting trees.
- *Structural Constraints:* Invalid expressions discarded during mutation.
- *Controllable Expression Complexity:* Restrict expression size, depth, and operator nesting for interpretable models.

Overall, `PySR`'s multi-level evolutionary design and rich feature set enable efficient exploration of complex symbolic search spaces while producing accurate and interpretable scientific models.

4.5.1 Software Implementation

The backend of `PySR` is implemented using pure `Julia` under the library name `SymbolicRegression.jl`, providing a high-performance engine for symbolic discovery. Julia's Just-In-Time (JIT) compilation significantly accelerates operator evaluation and optimization. One of its key features is operator fusion, where multiple operators are combined into a single fused operator, effectively resolving evaluation bottlenecks and improving computational efficiency. The frontend of `PySR` is implemented in Python and follows a familiar `scikit-learn`-style API, making it intuitive for machine learning practitioners while ensuring seamless integration with Python-based scientific workflows.

5 Modeling Design and Methods

5.1 Overview

Understanding the implementation necessitates establishing the methodology, which is presented in two steps. The first step is to comprehend an existing framework specifically designed for CPS modeling, and the second is to analyze the implementation of SR for hybrid system identification. This section follows the same structure, first introducing the Flowcean framework for CPS modeling ((8)), and then discussing symbolic regression as a strategy to identify hybrid systems ((7)).

5.2 Flowcean—Modeling Framework for CPS

The need for a data-driven approach that can automatically capture the dynamics of CPS from available data is crucial for reducing dependency on domain expertise and minimizing human error. While traditional machine learning libraries, such as `TensorFlow` and `PyTorch`, provide comprehensive implementations of various established learning paradigms and support both time-discrete and time-series behaviors in CPS, they lack a built-in modeling pipeline specifically designed to handle the diverse challenges and heterogeneous learning scenarios encountered in CPS modeling.

A typical data-focused modeling process typically follows a three-stage approach, as illustrated in Figure 6. To overcome the limitations of existing libraries and enable efficient CPS modeling, this work adopts `Flowcean` as the foundational framework, as it provides the flexibility and capabilities required to support such data-driven methodologies.

Figure 6: Data-Driven Modeling Workflow

Flowcean is a modular and extensible framework designed to support multiple learning strategies within a unified architecture ((8)). Unlike general-purpose libraries, it refines the data-driven modeling process to align closely with CPS-specific requirements, thereby improving flexibility and scalability across a wide spectrum of application domains.

As illustrated in Figure 7 ((8)), the CPS modelling workflow in Flowcean is organized into two main components: the *learning strategy* (A, B, C) and the *evaluation strategy* (D, E, F). Each strategy consists of well-defined stages that capture the essence of data-driven modelling. The *Environment* (A, D) represents the various data sources; *Transforms* (B) perform the necessary preprocessing tasks; and the processed data is passed to the *Learner* (C), which generates the *Model* (E). While the learning strategy focuses on model construction, the evaluation strategy assesses the model performance on unseen data (D)

using defined performance metrics (F). This structured design makes Flowcean well suited for hybrid system identification and CPS analysis.

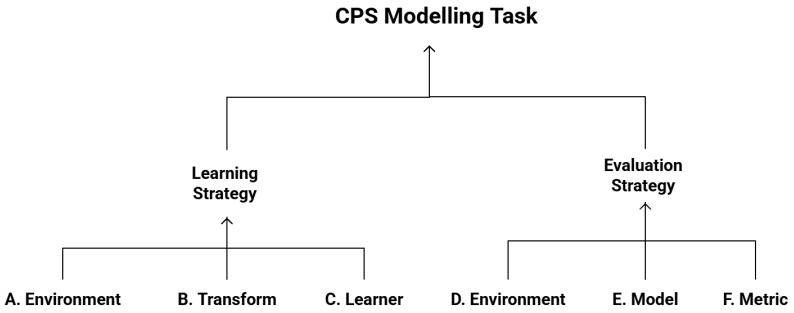


Figure 7: Component-View of the Concepts-Learning (A to C) and Evaluation (D to F)

The core concepts of Flowcean are summarized below, highlighting its modular architecture and the functionalities provided by its components.

Environments: Environments abstract different data sources and act as the unified entry point for the learning strategy. Flowcean supports three types of environments compatible with all learning strategies:

- *Offline Environment:* Works with precollected datasets stored in files or databases and supplies data in a single batch.
- *Incremental Environment:* Streams data sequentially to the learner. The data source can be real-time sensor data from simulations or batch-wise replay of stored data.
- *Active Environment:* Interacts directly with real or simulated CPS, enabling the learner to actively request data from the system when needed.

Transforms: Transforms prepare and adapt raw data for processing according to the learning strategy. Flowcean provides a rich set of transformation tools, including preprocessing, feature engineering, and data augmentation techniques. Common tasks include normalization, dimensionality reduction, and one-hot encoding. Multiple transforms can be combined into pipelines, enabling complex, customized workflows.

Learning Strategies: A learning strategy integrates the environment, applies transforms, and employs a learner to generate a predictive model. Flowcean supports three learning paradigms:

- *Offline Learning:* Uses data from the Offline Environment to train a model in a single batch.
- *Incremental Learning:* Processes sequential data from the Incremental Environment. At each step, the learner remains passive while the environment autonomously provides new data.

- *Active Learning:* Enables the learner to interact with the Active Environment, where its decisions influence data sampling, similar to reinforcement learning.

Model: The model is the output of the learning strategy and serves as an abstraction of the underlying CPS. It captures the knowledge acquired by the learner and is subsequently used for prediction of unseen data. By explicitly separating the learner from the model, Flowcean ensures modularity, allowing the trained models to be reused multiple times without requiring retraining of the learner.

Evaluation Strategies: Evaluation strategies provide insights into the performance of a model on test data and guide decisions related to comparison, optimization, and deployment. During evaluation, transformations are applied to the test data by the selected environment, which is then processed by the model, and performance is assessed using well-defined metrics.

The entire Flowcean architecture is implemented in Python. Its modular structure is organized into dedicated libraries and tools for environments, transforms, and learning/evaluation strategies. Both learning and evaluation strategies are implemented as functions that accept modular components as input arguments, improving extensibility and reusability. Additionally, Flowcean provides utilities for data loading and logging, supporting standard formats such as CSV and ROS bag files. Figure 8 outlines existing modules in the Flowcean framework.

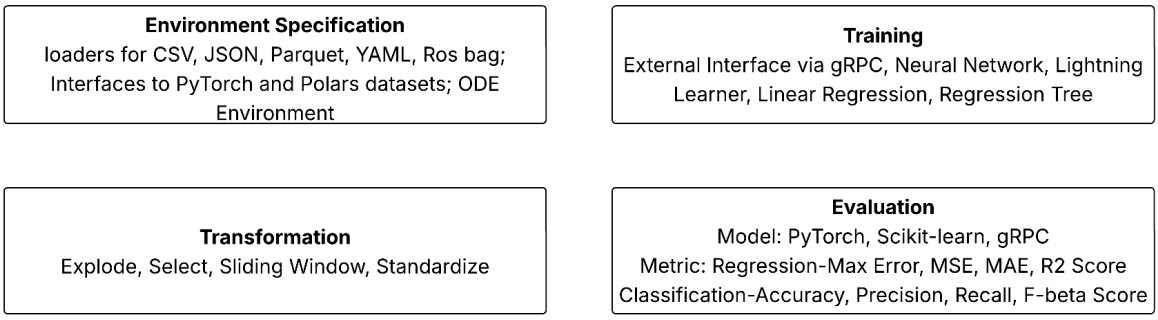


Figure 8: Existing Flowcean Modules

5.3 Symbolic Regression—Hybrid System Identification

Figure 9: Steps for Hybrid System Identification

The general approach to hybrid system identification using *Symbolic Regression* is illustrated in Figure 9 ((7)). SR provides a powerful framework for discovering interpretable symbolic expressions that describe the continuous dynamics of a system while also detecting transitions between modes and grouping segments governed by the same

dynamics. Unlike traditional approaches, which often rely on similarity-based heuristics to detect mode changes, this method identifies transitions directly from the underlying system dynamics ((7)).

This approach assumes that each discrete mode of a hybrid system is characterized by a unique symbolic expression. Initially, a small time window is used to learn a symbolic expression from the observed data. The window is then iteratively extended until a transition is detected, which is signaled by a significant drop in the fitness (loss function) of the learned expression. The data between detected transition points is segmented, and these segments are subsequently grouped into discrete modes. The grouping process also leverages SR, i.e., expressions are relearnt on unions of segments, and if combining them improves the loss, they are merged. The resulting symbolic expression then defines the flow function for the identified mode.

As shown in Figure 9, this identification strategy can be decomposed into four main sub-tasks:

1. Detecting discrete transitions between continuous dynamics, forming initial segments;

2. Grouping segments with similar dynamics into discrete modes;

3. Learning the continuous dynamics (flow function) for each mode;

4. Aggregating the identified modes to construct the complete hybrid system model.

In this project, we closely follow the methodology proposed in (7). This existing methodology is with a modification to Figure 9: the second and third steps are merged into a unified *grouping and identification* stage. As a result, the overall pipeline consists of three main phases:

1. *Detection of transitions between modes* using symbolic regression;

2. *Grouping and identification of mode dynamics* based on the segmented trajectories.

3. *Model construction* that involves combining the identified modes and respective symbolic equations into a comprehensive hybrid system model.

Detection of Transitions:

Given a data trajectory, the algorithm begins by selecting a fixed-length window over the initial portion of the data. As long as the segmentation criterion is satisfied, i.e., the loss of the learned symbolic expression does not significantly drop, the window is iteratively extended to include additional data points. A transition is detected when the criterion is violated, and the current window is stored as a segment in the set T.

Throughout this process, symbolic expressions are incrementally refined using n_{update} inner iterations of symbolic regression within each outer search iteration, defined by `niterations`. In the latest PySR version, n_{update} corresponds to the parameter `niterationspercycle`, whereas `niterations` determines the total number of evolutionary search steps. This incremental learning reuses prior knowledge, such as discovered sub-expressions and constants, enabling the expression to adapt as new data is added. For example, a segment that initially appears linear may later reveal a higher-order trend

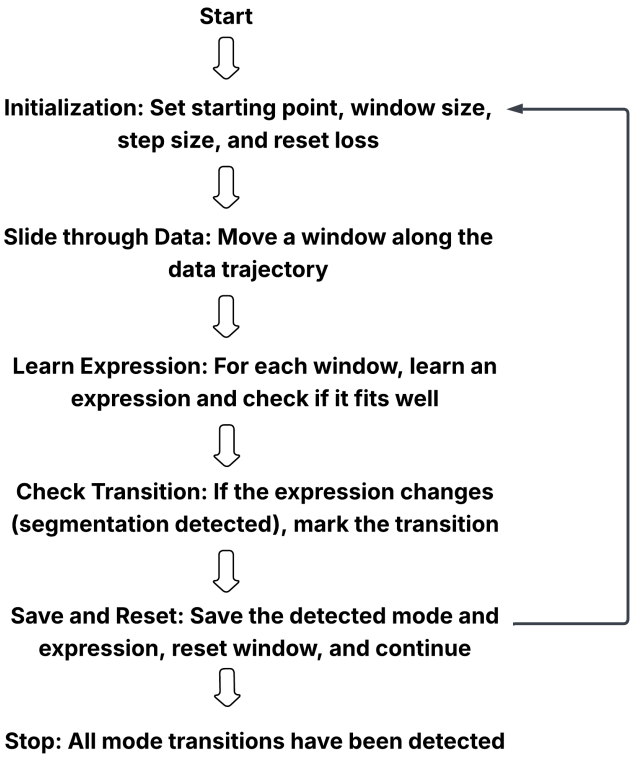


Figure 10: Detection of Mode Transitions

as the window expands. Once a transition is detected, the learning process restarts from the end of the previous segment with a fresh window, and a new expression is learned from scratch. The segmentation decision relies on a segmentation criterion, defined as follows:

$$\text{loss[end]} < \tau \quad \text{or} \quad \text{loss[end]} \leq \text{loss[end} - 1]$$

Here, τ represents a stagnation threshold on the loss value and is determined according to the desired modeling loss for the learned flow functions, with MSE loss employed to quantitatively evaluate the deviation. The term "loss[end]" denotes the loss value of the latest window being processed. This procedure is illustrated in Figure 10, where "fit" refers to the segmentation criterion and "reset loss" corresponds to the loss function.

Grouping and Mode Identification: Segments that follow the same symbolic expression are assumed to belong to the same mode. The goal of the second stage is to group such segments and learn a shared symbolic expression for each group.

The input is the set of detected segments T. The algorithm begins by initializing the first candidate group with the first segment. Then, for each subsequent segment, it attempts to combine with each existing group. A symbolic expression is re-learned on the union of the group and the current segment. Although computationally expensive, this relearning step helps to capture minor variations in dynamics that could have been overlooked earlier.

Let s be the current segment and g the current group. If the segment and group

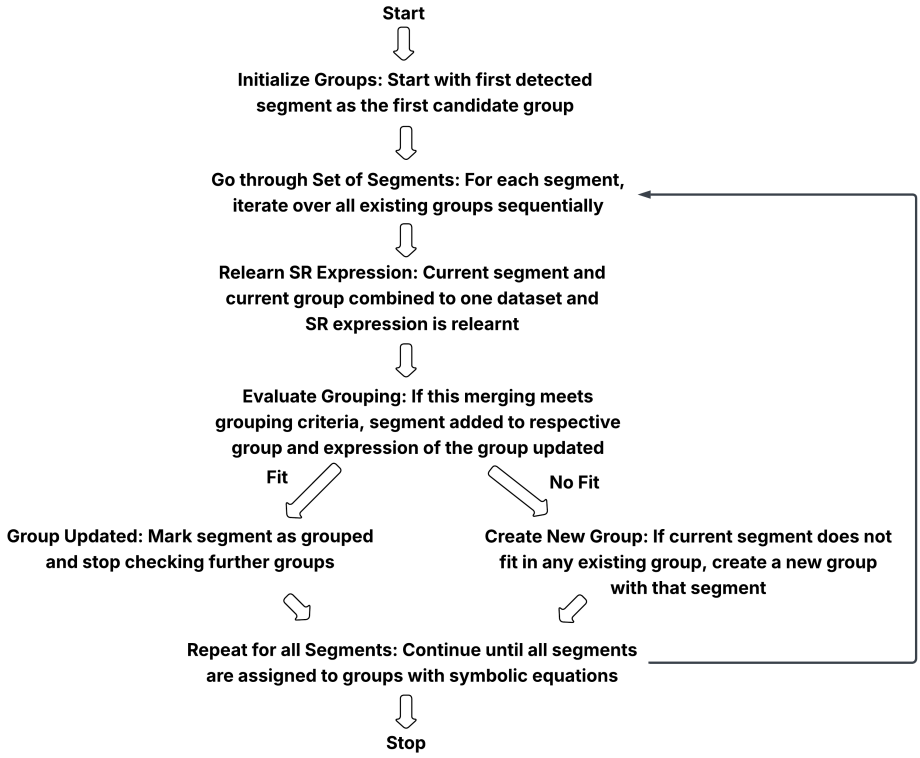


Figure 11: Grouping of Modes

combined $(s \cup g)$ yields a symbolic expression with sufficiently low loss compared to the loss of the group g before combining with segment s, the segment is added to the group. Otherwise, a new group is formed.

The grouping criterion is defined as

$$\text{loss}(s \cup g) \leq \phi \cdot \text{loss}(g)$$

where ϕ is a relaxation parameter and $\text{loss}(g)$ is the loss for the group before merging. A lower loss after merging indicates similarity in dynamics and justifies grouping. The entire process of grouping is shown in Figure 11.

The processes of detecting transitions and mode identification together result in a set of groups, each associated with a symbolic expression that defines the continuous dynamics of discrete modes in the hybrid system. The aggregation of segments, mode groupings, and flow functions together yields a complete hybrid system model.

6 Architecture and Implementation

6.1 Overview

The implementation of this project focuses on extending the Flowcean framework ((8)) to enable modeling of CPS exhibiting continuous-discrete (hybrid) behavior by integrating the dynamics-based identification method using SR proposed by (7). The implementation involves enhancements to learning strategy, addition of the SR model and evaluation metric of Root Mean Squared Error (RMSE) to evaluation strategy, and hyperparameter tuning using the `Optuna` library. Figure 12 illustrates the modified Flowcean modeling pipeline.

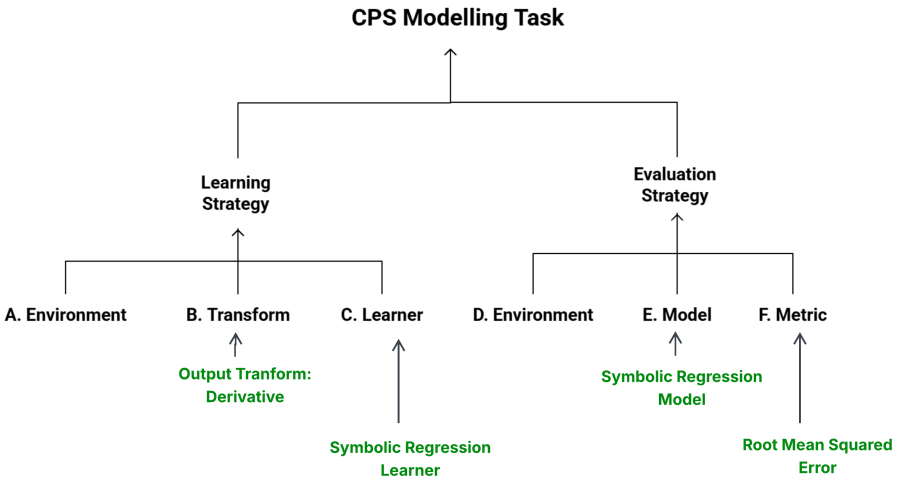


Figure 12: Modified Flowcean Modelling

6.2 Learning Strategy

A learning strategy integrates the environment, transforms, and a learner to generate a model, where the learning process is facilitated through a structured learning pipeline. In this project, we primarily adopt the *offline learning strategy*, which serves as the central focus of our implementation. Figure 13 illustrates the workflow of the offline learning strategy in the form of a flowchart.

Recent enhancements to the learning pipeline include the addition of a **Derivative** output transform and the introduction of the **SR learner**.
Data transformation is a critical step in the pipeline, involving the conversion of raw data into a suitable format for analysis and model training. Its primary purpose is to ensure compatibility of the data with the specific requirements of the CPS under consideration. By cleaning, encoding, and structuring data, transformation enables the extraction of meaningful insights and supports the development of accurate predictive models. The

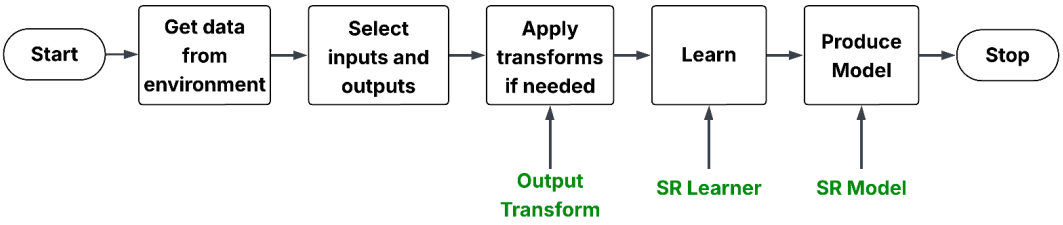


Figure 13: Modified Offline Strategy

choice of transformation techniques depends on the nature of the data and the learner algorithm. Flowcean provides multiple transformation techniques, including flatten, mean, median, mode, one-hot encoding, and several other options.

In this implementation, the integration of the SR learner requires the inclusion of a Derivative transform. The Derivative transform computes the *first-order difference* between consecutive data points, measuring the change of a variable over time. Mathematically, the first-order difference is expressed as:

$$\Delta y_t = y_t - y_{t-1} \tag{1}$$

where

- y_t is the current value at time t,
- y_{t-1} is the previous value, and
- Δy_t represents the change between two consecutive observations.

This technique is particularly useful when the CPS raw data captures state variable values at equally spaced time intervals. While the raw data provides absolute measurements, discrete models can often rely on the rate of change to formulate continuous dynamics.

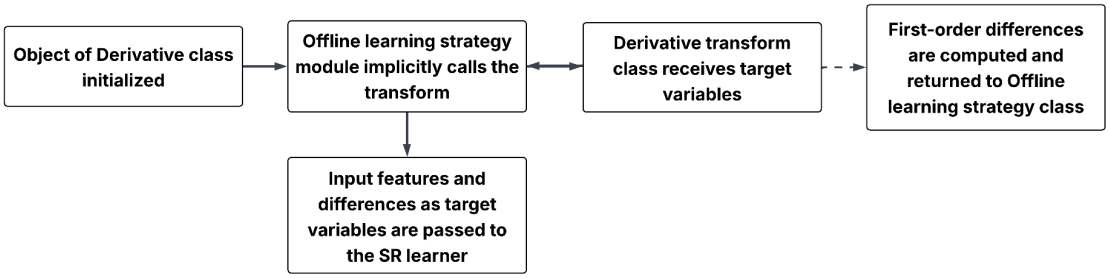


Figure 14: Derivative Transform

Within the SR learning framework, the Derivative transform is primarily applied as a transform on the target variable. It processes a dataframe containing the target variables and returns a new dataframe of first-order differences for each consecutive target variable

pair. Figure 14 outlines the implicit execution of Derivative transform when invoked within the pipeline.

The next most significant implementation step is integrating the SR learner into the Flowcean ecosystem. Unlike other learners (e.g., regression trees from scikit-learn), the SR learner incorporates a two-phase hybrid modeling approach based on (7) methodology:

1. **Mode Transition Detection**
    - A *sliding window* technique scans the dataset iteratively.
    - Symbolic regression is repeatedly applied within the window.
    - A *fitness function* (based on expression loss) is evaluated at each step.
    - Failure to meet the segmentation criterion results in a flagged mode transition.
2. **Mode Grouping**
    - After detecting transitions, similar time-series segments are grouped together.
    - Symbolic regression is reapplied to unions of segments.
    - When the grouping criterion is met, the segments are merged into a shared discrete group.

This two-step approach captures points of transitions as well as the underlying continuous dynamics within each mode.

The Flowcean integrated SR learner is implemented under the wrapper of the `symbolicregression.py` script. This design ensures complete encapsulation and cohesion by integrating all related methods and properties into a single unit, eliminating the need to switch between multiple files or import numerous modules. A single class import is sufficient to utilize the learner. Figure 15 illustrates the control flow of the integrated SR learner and highlights key class methods.

The `Symbolic Regression class` is responsible for learning symbolic models that describe the system's behavior. During the learning process, it relies on `Segmentor class` to detect transitions between different operating modes based on the data. The segmentation process is guided by the `Segmentation Criteria class`, which specifies the conditions used to identify mode transitions.

After segmentation, the `Group Identificator class` organizes the detected segments into meaningful groups of operating modes. The grouping process is carried out using the `Grouping Criteria class`, which establishes the criterion for merging segments into their corresponding groups. The `Group class` manages the aggregation of segments into their respective modes, while the `Segmented Data` and `Grouped Data classes` provide functionalities for visualizing results, formatting outputs, and exporting data for further analysis. The `Symbolic Regression class` ultimately produces a *SR model object* that contains the identified operating modes, their corresponding windows, and the symbolic equations that describe the system's behavior within each mode.

The Flowcean-integrated SR learner removes the need for verbose YAML (YAML Ain't Markup Language)-based input configurations. A systematic analysis of prior SR example case studies helped to classify parameters to the SR learner into two categories:

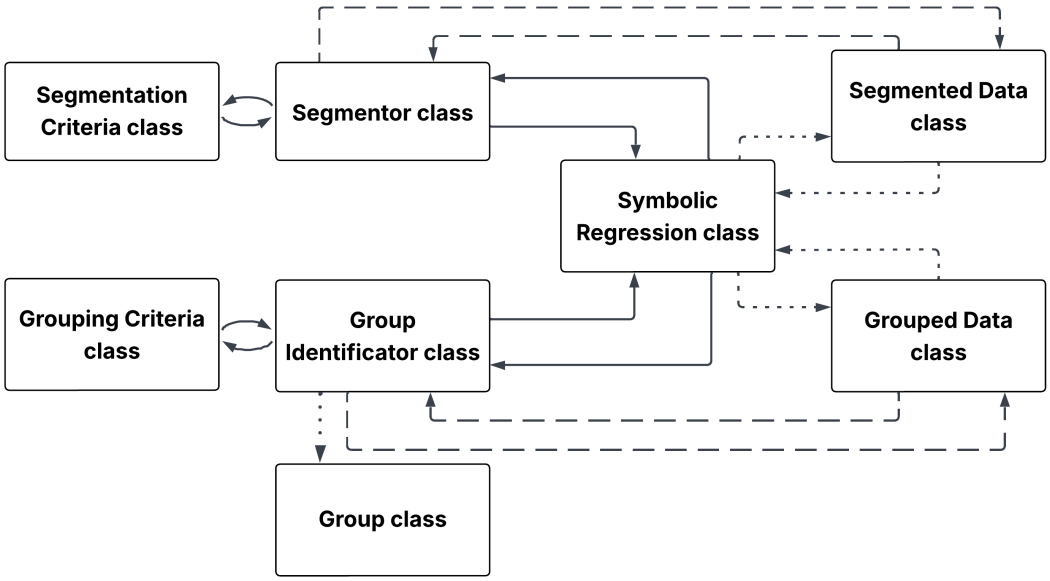


Figure 15: Flowcean Integrated SR Learner

1. User-defined arguments — dynamic parameters that vary based on the CPS being modeled and are provided during the initialization of the `Symbolic Regression class` object (Table 2)

2. Default arguments — fixed within the `Segmentor` and `Group Identificator classes`, particularly hyperparameters passed to `PySR()` as per (27) (Table 3)

The `segmentation_args` and `grouping_args` are configurable hyperparameters that allow overriding some or all default arguments given to `PySR()` settings. This provides flexibility when precise hyperparameter tuning is required for highly hyperparameter-sensitive segmentation and grouping. The underlying design and utilization of `PySR` and `Julia` for segmentation and the generation of symbolic equations remain unchanged.

Parameter	**Description**
`features`	List of input features to be considered during the learning.
`start_width`	Length of the window to be used during segmentation.
`step_width`	Step size for incrementing the segmentation window.
`target_var`	The target variable to be modeled.
`segmentation_args` & `grouping_args`	Configurable keyword arguments (`**kwargs`).

Table 2: User-Defined Arguments

Hyperparameter	Description
`niterations`	Number of iterations for PySR algorithm to run. *Default: 100*
`random_state`	Pass an integer seed to ensure reproducible results across multiple PySR function calls. *Default: 42*
`deterministic`	Ensures that PySR produces identical results on every run. *Default: True*
`parallelism`	Specifies the parallelization strategy for the search. *Default: serial*
`parsimony`	Multiplicative factor used to penalize equation complexity. Higher values favor simpler models. *Default: 0.32*
`binary_operators`	List of binary operators used during the search. *Default: [+, -, *, /]*
`unary_operators`	List of unary operators. *Default: [sqrt]*
`populations`	Number of populations running. *Default: 42*
`verbosity`	Controls the amount of logging information displayed. A value of `0` produces minimal output. *Default: True*
`model_selection`	Criterion used when selecting the final expression from the list of equations for each iteration. *Default: best*

Table 3: Default Arguments - PySR Hyperparameters

6.3 Evaluation Strategy

The `SymbolicRegressionModel class` is initialized with the grouped segments and their corresponding symbolic equations obtained from the SR learner. The flow of control between the SR learner and model is shown in Figure 16.

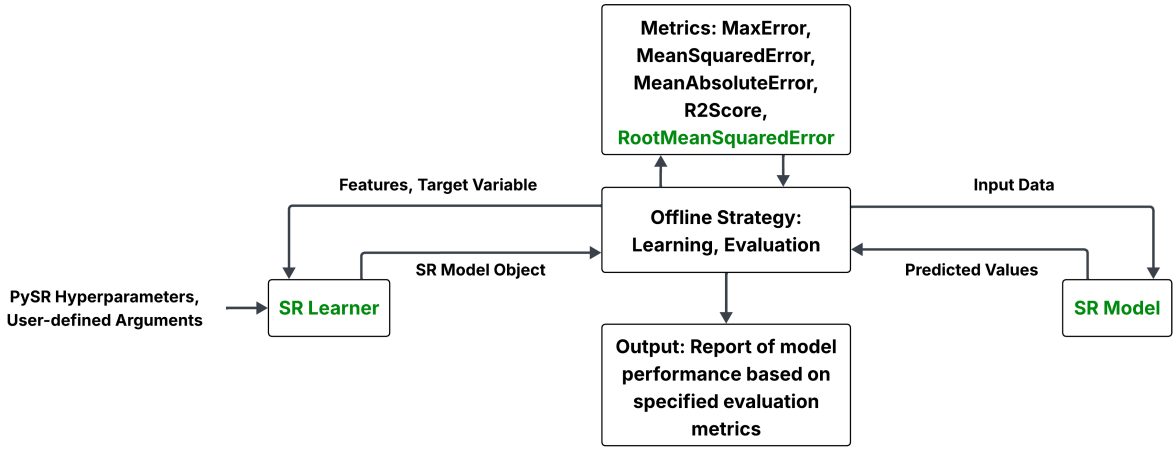


Figure 16: SR Integration Overview

Upon receiving input data, the model applies *static grouping* by directly mapping the start and end indices of each group, derived from the SR learner results, onto the input dataset. Each data point is then assigned a corresponding group ID. All the data points

with the same group ID are merged into blocks that form different discrete modes of the hybrid system. Using these group IDs, the symbolic equation associated with each mode is also assigned to the respective discrete modes. This symbolic equation is the flow function governing the continuous dynamics of each mode.

For effective evaluation using metrics such as MSE, the model must generate quantitative predictions instead of returning only grouped data and symbolic equations. The model subsequently evaluates these symbolic equations to compute the numerical predicted values for all input data points. A dataframe containing predicted values is returned as the model's output. Figure 17 outlines the working of the SR model.

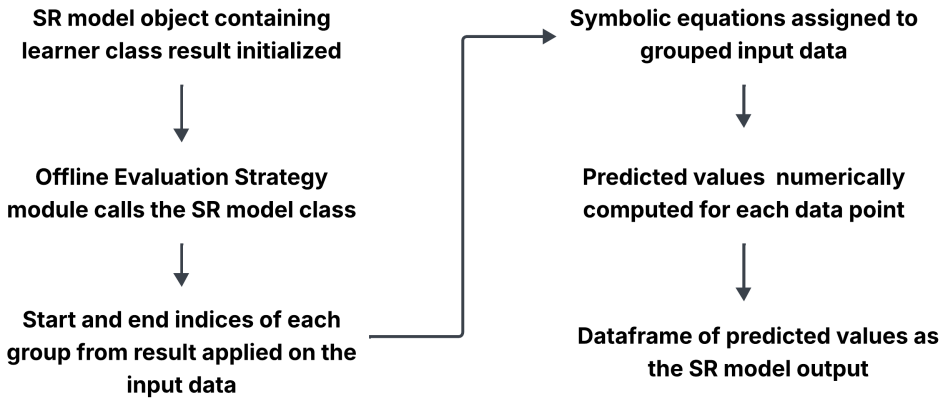


Figure 17: SR Model Working

Both *learning and evaluation* are conducted on the *same dataset*, with the primary objective being pipeline validation rather than achieving high predictive accuracy. This design is to ensure smooth transfer of intermediate results between learner and model modules, compatibility of the SR algorithm with the Flowcean framework, and support for modular extensions in alignment with Flowcean's CPS modeling objectives. By validating architectural soundness and interoperability first, the implementation lays the groundwork for future improvements, such as multi-stage learning and cross-validation.

The Flowcean framework supports several evaluation metrics, namely Mean Squared Error (MSE), Mean Absolute Error (MAE), Coefficient of Determination (R^2 Score), and Max Error. Additionally, this implementation introduces *Root Mean Squared Error (RMSE)*, defined as:

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

Here, y_i represents the actual values, $\hat{y}_i$ denotes the predicted values, and N refers to the total number of samples. The RMSE metric is particularly significant as it is highly sensitive to large deviations, effectively capturing substantial mismatches between symbolic predictions and actual values (in contrast to MAE). Unlike Mean Absolute Percentage Error (MAPE) or Symmetric Mean Absolute Percentage Error (sMAPE), RMSE avoids division by near-zero values, thereby ensuring mathematical stability. Moreover, since RMSE retains the same units as the predicted variable (distinct from MSE), it

enhances the interpretability of the results. As the SR model focuses on deriving interpretable symbolic expressions, RMSE serves as a reliable indicator of how well the derived equations approximate observed system dynamics.

6.4 Hyperparameter Tuning for SR

SR relies on sensitive hyperparameters that significantly affect both model accuracy and computational efficiency. The SR learner in this work leverages the PySR engine, which provides several tunable parameters. Fine-tuning of these parameters is essential to balance model expressiveness and computational tractability. Initially, a manual trial-and-error approach was used to adjust the values. Although this method may eventually yield good results, it is inherently time-consuming, highly intuitive, and not easily reproducible by other users. Hyperparameters that accept floating-point values posed even greater difficulty, as small changes could lead to dramatically different outcomes, making optimization intricate.

To overcome these limitations, **Optuna** ((28)) is used to automate the hyperparameter tuning of the PySR function. Optuna explores various combinations of hyperparameter values in multiple trials, gradually improving its search strategy based on previous results. In this work, the objective function in the tuning process is defined as the minimization of the loss function (MSE) computed on the dataset using a PySR function. Since the dataset is non-uniform and segmented into localized groups of data points, the combined MSE across all groups is used as the loss metric.

Unlike manual tuning or traditional search techniques such as grid search and random search, Optuna employs a dynamic and efficient approach based on the Tree-structured Parzen Estimator (TPE) algorithm ((28)) . TPE intelligently learns from previous trials and focuses on the most promising regions of the hyperparameter search space. Furthermore, Optuna implements automated trial pruning that stops underperforming configurations early on, reducing computational overhead. This is essential for PySR, where each trial can be computationally expensive, particularly for high-dimensional datasets.

Optuna also provides an integrated visual dashboard that assists in analyzing the tuning process. The dashboard helps to visualize how each hyperparameter affects the minimization of loss, understand the relative importance (per cent) of hyperparameters in optimizing the objective function as depicted in Figure 18, track the evolution of loss (MSE) across multiple trials (Figure 19), and explore relationships between hyperparameter combinations and model performance. This combination of adaptive optimization, automated pruning, and visual interpretability provides a robust and efficient framework for fine-tuning PySR hyperparameters, ensuring improved model accuracy and computational efficiency.

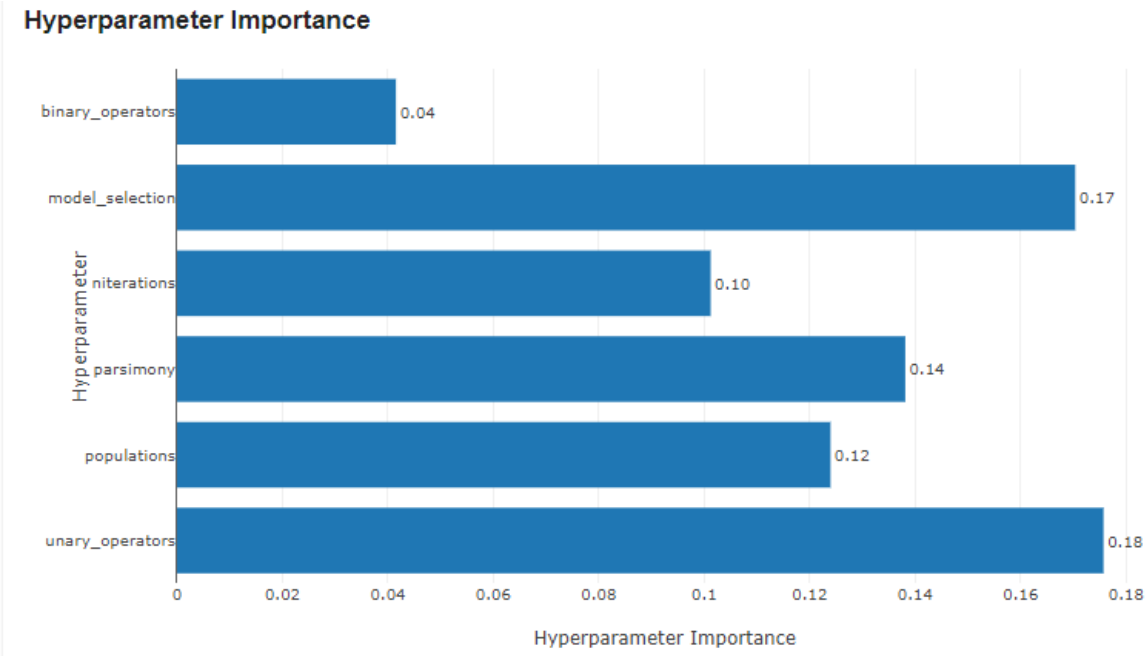


Figure 18: Hyperparameter Importance Plot

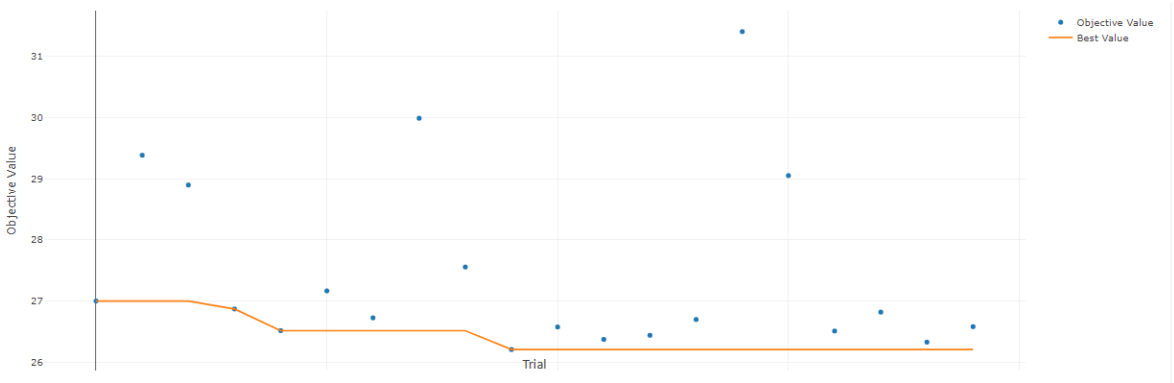


Figure 19: Evolution of Objective Function over Trials

7 Experimental Evaluation

7.1 Overview

To validate the integration of the SR learner into the Flowcean framework, a series of different hybrid system case studies are listed below. These examples were selected to demonstrate different levels of system complexity, nonlinearities, and switching behaviors.

Rather than focusing solely on the numerical accuracy of the discovered equations, the objective is to evaluate the SR learner's ability to segment system dynamics into distinct operational modes, discover governing relationships between variables, and adapt to both multi-mode and single-mode systems. The impact of Optuna-based hyperparameter tuning on performance is also explored.

7.2 Case Evaluations

The PySR algorithm is highly sensitive to its hyperparameters. For example, different values of `niterations`, such as 100 or 130, may both appear to work during manual tuning, but they offer no concrete justification for others attempting to replicate or understand the modelling process. Similarly, the `parsimony` parameter can be extremely sensitive; minor differences, such as 0.00324 versus 0.003246, can result in substantially different segmentation and grouping outcomes. Optuna-based hyperparameter optimization systematically selects parameters based on the dataset, providing data-driven, reproducible, and interpretable models. In CPS modelling, where segmentation and grouping are critically influencing the model behavior, automated tuning eliminates ambiguity and improves both robustness and interpretability. In each case study, the optimization is conducted over 20 trials, with the goal of minimizing the objective function (MSE as loss metric) in each trial.
It is worth noting that values of evaluation metrics such as MSE or RMSE, which reflect model performance, are not discussed. This is intentional, as the focus of this work is on the architectural implementation rather than model accuracy. Moreover, since the training and testing datasets are identical, these metrics have limited significance at this stage. Nonetheless, RMSE has been incorporated into the framework to allow assessment of symbolic regression model performance in future developments.

7.2.1 Vehicle Cruise Control

The first experiment models a vehicle cruise control system with two operating modes - Accelerate and Brake. Mode switching occurs based on velocity thresholds: the vehicle accelerates below $50\,m/s$ and brakes above it (refer Table 4). The system is simulated over $T = 1000\,s$ using a timestep of $1\,s$. The dataset records time, initial velocity $v(t)$, final velocity $v(t+1)$, and velocity change dx.

With a window width of 100 and a step size of 40, the model achieved performance metrics of MSE with a value of 0.0231 and RMSE being 0.1522. When the objective function shows minimal variation across trials, Optuna cannot determine the influence of individual hyperparameters. Due to the near-zero variance, the importance calculation

Quantity	Expression	Mode Switching
System Dynamics	$\begin{cases} v(t+1) = v(t) + 5, & \text{if mode} = \text{Accelerate} \\ v(t+1) = v(t) - 7, & \text{if mode} = \text{Brake} \end{cases}$	If $v(t) < 50$, Accelerate If $v(t) \geq 50$, Brake

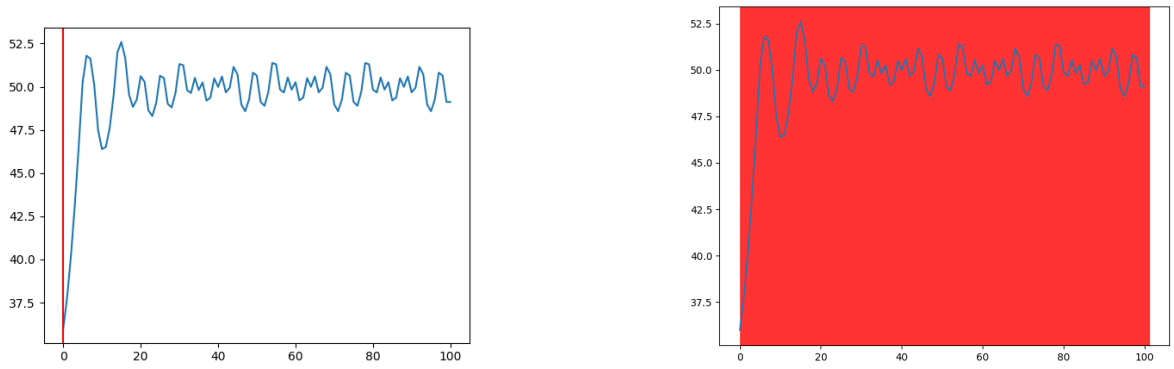
Table 4: System Equations

Parameter	Value	Importance (%)
niterations	60 (segmentation only)	NaN
parsimony	0.00018142230721673977	NaN
binary_operators	`[+, -]`	NaN
unary_operators	`[sqrt, sin, cos]`	NaN
population_size	56	NaN
model_selection	best	NaN

Table 5: Tuned Hyperparameters

becomes unstable, often resulting in NaN values (refer Table 5). Despite this, the SR learner produced a single unified equation (Equation (3)), where dx captured both acceleration and braking behaviors. Figure 20 shows the segmentation and grouping outputs.

$$v(t+1) = v(t) + dx \tag{3}$$



(a) Segmentation Output

(b) Grouping Output

Figure 20: X-axis depicts the Dataframe Number and Y-axis is Final Velocity $v(t+1)$

Although the system operates in two distinct modes, the SR learner implicitly captured both within a single continuous relationship through dx, without distinguishing between positive and negative signs. This outcome underscores the SR model's ability to unify seemingly disparate regimes when the underlying dynamics share structural similarities.

7.2.2 Smart Sensor (Two-Mode System)

Next, the SR learner is evaluated on a smart sensor system that alternates between two operational regimes - Amplification (quadratic law) and Compression (square root law).

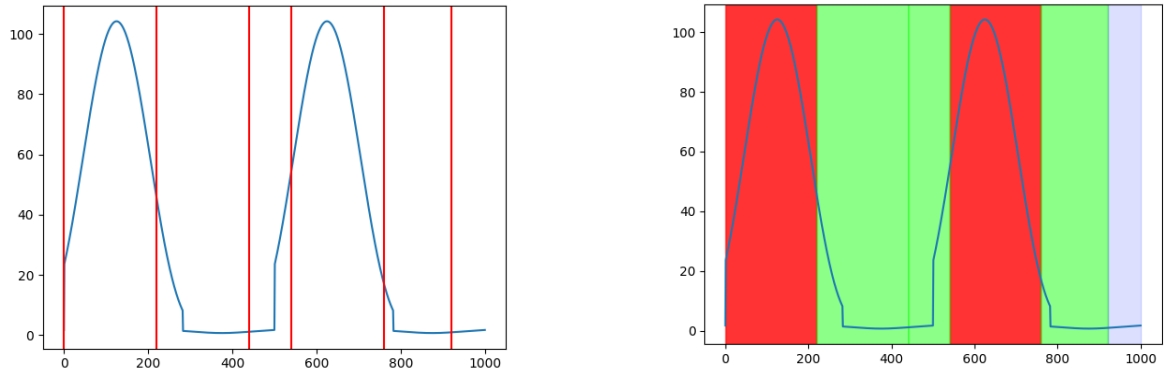
Mode transitions are triggered when the input $u(t)$ crosses specific thresholds, introducing switching (refer Table 6). The system is simulated for $T = 10\,s$ with a fine-grained timestep of $0.01\,s$. The dataset records time, the input $u(t)$ and the output y, providing a detailed view of switching behavior.

Quantity	**Expression**	**Mode Switching**
Output y	$y = \begin{cases} \sqrt{u+1}, & q = 1 \text{ (Compression)} \\ 5u^2 + 3, & q = 2 \text{ (Amplification)} \end{cases}$	If $q = 1$ and $u > 2$, switch to mode 2 If $q = 2$ and $u < 1$, switch to mode 1

Table 6: System Equations

Parameter	**Value**	**Importance (%)**
niterations	120 (segmentation only)	9%
parsimony	5.158×10^{-5}	3%
binary_operators	`[+, -, *]`	62%
unary_operators	`[sqrt, sin, cos]`	3%
populations	48	4%
model_selection	best	5%

Table 7: Tuned Hyperparameters



(a) Segmentation Output

(b) Grouping Output

Figure 21: X-axis depicts the Dataframe Number and Y-axis is Output y

$$\text{Group 0:} \quad y = u \cdot (u \cdot 4.781374 + 1.6382203) \tag{4}$$

$$\text{Group 1:} \quad y = \frac{u^2}{0.19230054} \tag{5}$$

$$\text{Group 2:} \quad y = \sqrt{u+1} \tag{6}$$

The tuned hyperparameters obtained using the Optuna framework, along with the corresponding importance scores for each parameter are presented in Table 7. A window of start width of 100 and step width of 60 is used. The resulting MSE and RMSE are of 5.732 and 2.394, respectively.

The findings are particularly illuminating. The emergence of more than two governing equations can be attributed to the influence of the sliding window. Group 1 captures the mixed-mode dynamics, characterizing the transitional regime in which both modes are simultaneously represented within the data encompassed by the window. In contrast, Group 0 and Group 2 correspond to distinct single-mode behaviors, associated with $q = 1$ and $q = 2$, respectively, when learned. Since variations in the dynamics trigger transitions, the SR learner treats this transitional regime as a distinct, separate mode.

7.2.3 Smart Sensor (Single-Mode System)

To test the SR learner's stability in simpler scenarios, a single-mode (Compression) smart sensor is considered governed by:

$$y = \sqrt{u + 1},$$

with the input u defined as

$$u = 2.5 \sin(2\pi \times 0.2 \times t) + 2,$$

In this configuration, no mode switching occurs, the system operates under a smooth, continuous relationship. The system is simulated over T = 10s with a timestep of 0.01s. The dataset contains time, input u, and output y, ensuring a clean, noise-free signal.

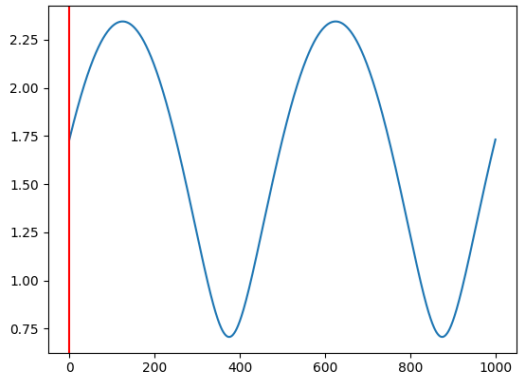
Parameter	**Value**	**Importance (%)**
niterations	130	7%
parsimony	1.585×10^{-6}	5%
binary_operators	`[+, -, *]`	15%
unary_operators	`[sqrt, sin, cos, exp, log]`	18%
populations	49	8%
model_selection	accuracy	1%

Table 8: Tuned Hyperparameters

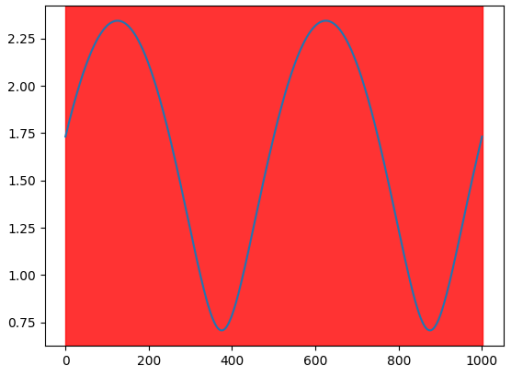
The optimized values of the hyperparameters, determined through Optuna's tuning process, are summarized in Table 8, along with their respective contributions to the model's performance. The analysis was conducted with a window initialized at a width of 100 and advanced in increments of 100. Under this configuration, the model achieved an MSE of 7.753×10^{-30} and a RMSE of 2.784×10^{-15} . As expected, the SR learner correctly identified a single governing equation identical to the ground truth (Equation (7)). The plots in Figure 22 confirmed that the entire dataset belonged to one group.

$$y = \sqrt{u + 1} \tag{7}$$

This experiment demonstrates the SR model's robustness in handling single-mode systems. In contrast to multi-mode cases, no overfitting or unnecessary segmentation was observed, confirming the learner's adaptability.



(a) Segmentation Output



(b) Grouping Output

Figure 22: X-axis depicts the Dataframe Number and Y-axis is Output y

7.2.4 DC-DC Converter

Finally, the SR learner is tested on a DC-DC buck converter, a classical hybrid system widely used in power electronics. The converter operates in two distinct modes–switch ON, during which the inductor charges, and switch OFF in which the inductor discharges. The governing dynamics are represented by ODEs (refer to Table 9). Using *explicit Euler integration*, the converter under switching conditions is simulated. The system is simulated for $T = 0.1$ s using a timestep of 1 μs. The dataset comprisies of time, inductor current i_L, capacitor voltage V_C, switch state, input voltage Vin, as well as the component parameters inductance L_H, capacitance C_F, and resistance R_Ohm.

Quantity	**Expression**	**Switching Logic**
Inductor Current (i_L)	$\frac{di_L}{dt} = \begin{cases} \frac{V_{in}-V_C}{L}, & \text{ON mode} \\ \frac{V_C}{L}, & \text{OFF mode} \end{cases}$	Hysteresis control: ON $\rightarrow$ OFF: $V_C \geq V_{\text{ref}} + \text{hyst}$ OFF $\rightarrow$ ON: $V_C < V_{\text{ref}} - \text{hyst}$
Capacitor Voltage (V_C)	$\frac{V_C}{dt} = \frac{i_L - \frac{V_C}{R}}{C}$ (both modes)	

Table 9: System Equations

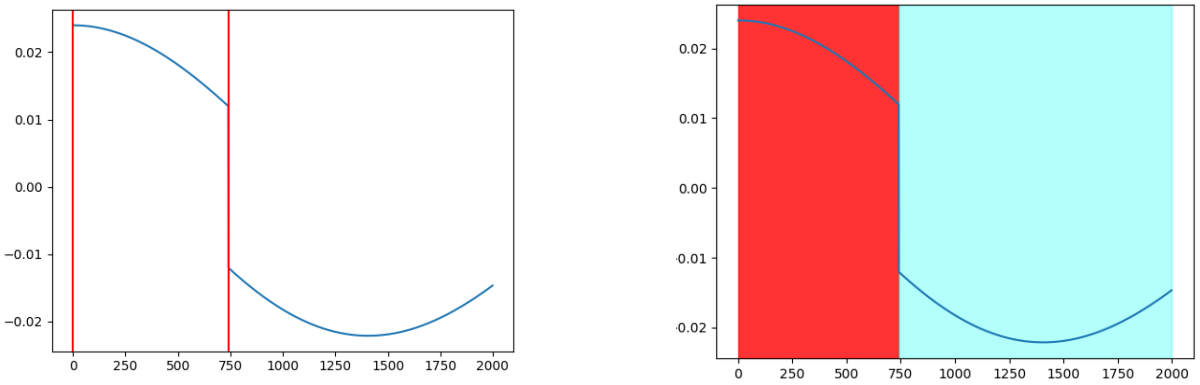
$$\text{Group 0:} \quad di_L = 6.640 \times 10^{-6} \cdot L_H \Big(0.001 \cdot \sqrt{V_C} - V_C + 0.037 \cdot L_H + V_{\text{in}} + 0.001 \cdot \sqrt{V_C \cdot V_{\text{in}} - V_C + 0.009} \Big) \tag{8}$$

$$\text{Group 1:} \quad di_L = 3.292 \times 10^{-10} \cdot V_C \tag{9}$$

The SR learner, along with the derivative transform on i_L, guided by tuned hyperparameters (Table 10), successfully identified two distinct groups of dynamics (Equation (8)

Parameter	Value	Importance (%)
niterations	120 (segmentation only)	10%
parsimony	6.768×10^{-5}	14%
binary_operators	`[+, -, *, /]`	4%
unary_operators	`[sqrt, sin, cos, exp, log]`	18%
populations	60	12%
model_selection	accuracy	17%

Table 10: Tuned Hyperparameters



(a) Segmentation Output

(b) Grouping Output

Figure 23: X-axis depicts the Dataframe Number and Y-axis is $i_L(\mathtt{A})$

and Equation (9)). Using a window of width 100 with a step size of 40 yields an MSE of 70.476 and an RMSE of 8.395. Figure 23 shows the segmentation and grouping of two operational regions. Although the SR model correctly identified the two operational modes, the derived equations are complex and nested. This complexity arises from nonlinearity in the data, which can prompt the model to fit minor fluctuations, and from the availability of a large operator set, which allows numerous ways to improve expression loss. Consequently, the resulting symbolic equations may become convoluted, particularly when representing intricate system dynamics. Nonetheless, the model successfully captured the essential ON/OFF switching behavior, demonstrating its capability for mode segmentation and grouping.

All of these case studies also highlight the advantages of SR for the identification of hybrid systems. In contrast, using a neural network (NN) would result in a single continuous function fitted across all data points, with loss minimized at each sample, but it produces trained weights and activations, offering no human-interpretable equations or distinct mode transitions. For hybrid systems like Smart Sensor, NN models would attempt to fit mixed-mode data by averaging the outputs of other two modes, providing no insight into the underlying system dynamics. NNs cannot inherently handle piecewise behaviors; accurate modeling would require explicitly segmented data for each mode or the inclusion of mode identifiers as input features. SR, in contrast, captures both continuous dynamics and discrete mode transitions, providing interpretable expressions and eliminating the need for manual segmentation or additional features.

8 Conclusion and Future Work

This work demonstrates the integration of Symbolic Regression within the Flowcean framework for identifying and grouping the modes of hybrid systems based on their underlying dynamics. By leveraging Flowcean's modular architecture, the integration of SR enables a seamless and unified toolchain for hybrid system identification.

The inclusion of SR learner along with Derivative transform and SR model enhances the ability of the framework in hybrid system identification and flexibility by replacing verbose inputs based on YAML with concise, user-friendly function arguments. Hyperparameters that rarely require modification are maintained as defaults, simplifying usage while preserving customization options when needed. The integration of auxiliary tools, such as `Optuna`, has also been streamlined, enabling efficient hyperparameter tuning as a pre-processing step.

To provide a more comprehensive model evaluation, the incorporation of the Root Mean Squared Error complements the existing set of performance metrics. Extensive case studies demonstrate that the SR learner successfully identifies mode transitions, captures transition dynamics, and robustly models both single-mode and multi-mode systems. These findings validate the framework's adaptability and scalability for hybrid system analysis, supporting Flowcean's vision of modularity and seamless integration of diverse learning strategies for CPS applications. Table 11 summarizes the key insights, findings, and takeaways derived from these studies, highlighting the Flowcean integrated SR framework's adaptability across different scenarios.

Case Study	**Key Observation**
Cruise Control	Braking and acceleration are unified into a single governing equation.
Smart Sensor (Two-Mode)	Identified a transitional regime occurring between the modes.
Smart Sensor (Single-Mode)	Accurately modeled single-mode dynamics; no over-segmentation.
DC-DC Converter	Correct segmentation; equation accuracy limited by hyperparameter sensitivity.

Table 11: Summary of SR Model Performance

Unlike traditional neural network models, PySR does not rely on weights; instead, its performance is governed entirely by hyperparameters, making accurate tuning critical. Future extensions of this work could focus on the following directions:

1. *Enhanced Hyperparameter Optimization*
   Integrate Optuna as a native Flowcean module, leveraging its hyperparameter importance scores to identify less impactful parameters. A dedicated module could then automatically detect and exclude these negligible parameters, thereby simplifying the model and reducing computational overhead.

2. *Improved Interpretability*
   Incorporate SymPy, an open-source Python library for symbolic mathematics that

enables equation manipulation and simplification, into the framework to simplify symbolic equations generated by PySR, making results easier to interpret in practical CPS applications. This is useful for case studies like DC-DC Converter.

3. *Scaling to High-Dimensional Datasets*
   Extend case studies to hybrid systems with rich, high-dimensional datasets. Training on such datasets could reduce the required number of learning iterations, improving efficiency while maintaining accuracy and interpretability.

4. *Automated Window Parameter Selection*
   Develop systematic methods for selecting window start and step widths in segmentation tasks. In contrast to labor-intensive trial-and-error approaches, data-driven techniques could be employed to adaptively determine optimal width parameters.

By combining these enhancements, Flowcean can evolve into a more scalable, interpretable, and efficient framework for hybrid system identification and modeling, further broadening its applicability across CPS domains.

References

[1] E. A. Lee, "Cyber-physical systems: Design challenges," *2006 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pp. 363–369, 2006.

[2] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," *Design Automation Conference*, 2010.

[3] P. Derler, E. A. Lee, and A. Sangiovanni-Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012.

[4] S. Plambeck, M. Schmidt, G. Fey, A. Subias, and L. Travé-Massuyès, "Dynamics-based identification of hybrid systems using symbolic regression," *Proceedings of the 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 64–71, 2024.

[5] S. Krishna and Others, "Hybrid automata for formal modeling and verification of cyber-physical systems," *Proceedings of the 10th International Conference on Embedded Software (EMSOFT)*, pp. 123–134, 2008.

[6] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," *Lecture Notes in Computer Science*, vol. 736, pp. 209–229, 1993.

[7] S. Plambeck, M. Schmidt, G. Fey, A. Subias, and L. Travé-Massuyès, "Dynamics-based identification of hybrid systems using symbolic regression," in *Proceedings of the ... (Conference/Workshop Name)*, Toulouse, France.

[8] M. Schmidt, S. Plambeck, G. Fey, J. C. Wieck, M. Knitt, H. Rose, and S. Balduin, "Flowcean—model learning for cyber-physical systems," Hamburg, Germany.

[9] H. Mo, G. Sansavini, and M. Xie, *Cyber-Physical Distributed Systems: Modeling, Reliability Analysis and Applications.* John Wiley & Sons, 2021.

[10] M. Uddin, H. Mo, and D. Dong, "A review on modelling, evaluation, and optimization of cyber-physical system reliability," *arXiv preprint arXiv:2503.10924*, 2025.

[11] L. Cao, X. Jiang, Y. Zhao *et al.*, "A survey of network attacks on cyber-physical systems," *IEEE Access*, vol. 8, pp. 44 219–44 227, 2020.

[12] Z. Leng, Z. Zhou, Y. Tang *et al.*, "Complex network based virtual power plant cps reliability analysis," *Proceedings of the 2023 IEEE International Conference on Power Science and Technology (ICPST)*, pp. 401–405, 2023.

[13] R. Bemthuis, M. Mes, M.-E. Iacob *et al.*, "Using agent-based simulation for emergent behavior detection in cyber-physical systems," *Proceedings of the 2020 Winter Simulation Conference (WSC)*, pp. 230–241, 2020.

[14] I. Onaji, P. Soulatiantork, B. Song *et al.*, "Discrete event simulation for a cyber-physical system," *Advances in Manufacturing Technology XXXIII*, pp. 213–218, 2019.

[15] Y. Liu, L. Deng, N. Gao *et al.*, "A reliability assessment method of cyber physical distribution system," *Energy Procedia*, vol. 158, pp. 2915–2921, 2019.

[16] I. Graja, S. Kallel, N. Guermouche *et al.*, "A comprehensive survey on modeling of cyber-physical systems," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 15, p. e4850, 2020.

[17] T. Nägele and J. Hooman, "Rapid construction of co-simulations of cyber-physical systems in hla using a dsl," *Proceedings of the 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 183–190, 2017.

[18] A. Rajhans, "An architectural approach to the design and analysis of cyber-physical systems," *Electronic Communications of the EASST*, vol. 21, 2009.

[19] G. Simko, T. Levendovszky, M. Maroti *et al.*, "Towards a theory for cyber-physical systems modeling," *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, 2014.

[20] T. Jain and K. Verma, "Mcs-ann-based hybrid approach for reliability assessment of composite power system," *Proceedings of the 2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON)*, pp. 1–6, 2021.

[21] R. Rai and C. K. Sahu, "Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus," *IEEE*, 2020, available online.

[22] J. Tang, J. Zhao, J. Ding, L. Chen, G. Xie, B. Gu, and M. Yang, "Cyber-physical systems modeling method based on modelica," *IEEE Sixth International Conference on Software Security and Reliability Companion*, 2012.

[23] H. Dayekh, N. Basset, and T. Dang, "Hybrid system identification through optimization and active learning," *IFAC-PapersOnLine*, vol. 58, no. 11, pp. 87–92, 2024.

[24] D. L. Ly and H. Lipson, "Learning symbolic representations of hybrid dynamical systems," *Journal of Machine Learning Research*, 2012.

[25] S. Plambeck, A. Bracht, N. Hranisavljevic, and G. Fey, "Famos – fast model learning for hybrid cyber-physical systems using decision trees," in *Proceedings of the [Conference Name]*, Hamburg, Germany, 2024.

[26] N. Makke and S. Chawla, "Interpretable scientific discovery with symbolic regression: A review," *arXiv preprint arXiv:2305.01592*, 2023, qatar Computing Research Institute, HBKU, Doha.

[27] M. Cranmer, "Pysr: Fast parallel symbolic regression in python and julia," https://github.com/MilesCranmer/PySR, 2023, accessed: 2025-07-28.

[28] T. Y. T. O. Takuya Akiba1, Shotaro Sano1 and M. Koyama1, "Optuna: A hyperparameter optimization framework," https://optuna.org/, accessed: 2025-07-28.