Glass-Box Activity Discovery from Raw Sensor Data

Christian Imenkamp[1,*], Henrik Kirchmann[2], Svenja Plumhoff[1], Matthias Weidlich[2] and Agnes Koschmider[1]

[1]Chair of Business Informatics and Process Analytics, Universität Bayreuth, Wittelsbacherring 8, 95444 Bayreuth, Germany
[2]Hasso Plattner Institute, University of Potsdam, Germany

Abstract

Business processes increasingly run in sensor-rich environments, where Internet-of-Things devices record physical behavior as raw, continuous measurements. Such data is a promising input for process mining. However, process discovery methods expect an event log of named activities, whereas sensors provide only timestamped values and no activity alphabet. Constructing this alphabet is the real bottleneck: existing methods either rely on domain experts to define the relevant activities, or derive them automatically as opaque clusters or stochastic labels that cannot be checked against the sensor data. We study process discovery from raw sensor values without labels, domain mappings, or analyst input. Our key idea is to synthesize the activity alphabet rather than cluster or guess it: each activity is an explicit logical rule over sensor behavior. We segment the stream, describe each segment with interpretable path-signature features, and use Syntax-Guided Synthesis (SyGuS) to derive, for each segment class, a rule that is verified against the data by construction. The abstraction is therefore a glass box: rather than trusting an opaque model, one can check why each activity was formed and what sensor conditions define it, and optionally attach a human-meaningful name to each rule afterwards. A standard algorithm then discovers a process model over these rules. We evaluate the synthesized activities on two public datasets (HACCP and SWaT). Despite using no domain knowledge, the discovered activities cover almost all of the data and, on one dataset, outperform an expert-guided abstraction method, while remaining behind it in the other cases.

1. Introduction

Business processes are increasingly executed in sensor-rich physical environments. In manufacturing, logistics, and healthcare, Internet-of-Things (IoT) devices continuously record physical behavior as timestamped measurements, such as position, pressure, temperature, and motion [1]. In principle, this data is a rich source for *process mining*, the family of techniques that analyze operational processes from event data [2]. Its most fundamental task, *process discovery*, constructs a process model from observed executions and thereby makes the control flow of a process explicit. Bringing process discovery to sensor-observed physical processes would make a large and growing class of systems accessible to process analysis.

However, sensor data is not the kind of input that process discovery expects. Classical discovery requires an event log in which every event refers to a case, an *activity*, and a timestamp [3]. IoT devices supply only the timestamp: they emit continuous, multivariate streams with no activity names, no case boundaries, and, crucially, no *activity alphabet*. Before any model can be discovered, the raw stream must be turned into events through *event abstraction*, which maps fine-granular physical observations onto coarse-granular activity instances [4, 5]. Constructing this alphabet, rather than discovering control flow over a given one, is the real bottleneck of process mining on IoT data.

Today this step is paid for either with domain knowledge or with interpretability. Surveys of event abstraction show that the combination most relevant to IoT, unsupervised abstraction over continuous sensor data, is the least explored [4, 6]. Supervised and expert-guided methods assume that the relevant

BP-Meet-IoT'26: 10th International Workshop on Business Processes Meet the Internet of Things, affiliated with CBI/EDOC 2026
*Corresponding author.
christian.imenkamp@uni-bayreuth.de (C. Imenkamp); henrik.kirchmann@hpi.de (H. Kirchmann); Svenja.Plumhoff@uni-bayreuth.de (S. Plumhoff); matthias.weidlich@hpi.de (M. Weidlich); agnes.koschmider@uni-bayreuth.de (A. Koschmider)
0009-0007-4295-1268 (C. Imenkamp); 0009-0007-7521-2955 (H. Kirchmann); 0009-0008-1671-7942 (S. Plumhoff); 0000-0003-3325-7227 (M. Weidlich); 0000-0001-8206-7636 (A. Koschmider)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC-BY-4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-02

CEUR-WS.org/Vol-4310/paper1.pdf

activities are already known and merely need to be recognized [7, 8]. Recent unsupervised methods avoid this assumption but produce alphabets that cannot be inspected: cluster identifiers are opaque [9], LLM-generated labels are stochastic and not verifiable against the sensor data [10], and rule-based detectors can only fire on activities an expert already annotated [11]. In all cases, there is no formal, checkable link between an activity and the sensor behavior it claims to represent. This link matters because the activity alphabet is what every later step builds on: process discovery, conformance checking, and any decision drawn from the model inherit the meaning of its activities. When that meaning cannot be inspected and verified against the sensor data, an analyst cannot tell whether a discovered activity corresponds to a genuine physical regime or is merely an artifact of segmentation and abstraction, and the resulting model cannot be trusted to represent the observed process.

In this paper, we address this missing link between activities and sensor behavior. We study process discovery from raw sensor values when no activity alphabet is given, with no labels, no domain mappings, and no analyst decisions. The hard part, and the focus of this paper, is *constructing the activity alphabet itself*; once it exists, process discovery reduces to a standard downstream step. Our key idea is to *synthesize the activity alphabet* rather than cluster or guess it: each activity is an explicit logical rule over sensor behavior, such as *position increases while pressure stays stable*, instead of a cluster index or an LLM string. We segment the stream, describe each segment by interpretable features that capture how sensor values move and interact over time (using path signatures [12]), and apply syntax-guided synthesis (SyGuS) [13] to derive, for each segment class, a Boolean predicate that covers its examples and excludes the others. This keeps the whole abstraction a glass box: there is no learned embedding to trust and no opaque score, only named features and explicit logical rules. Because every rule is returned together with the specification it provably satisfies, an analyst can check why each activity was formed and what sensor conditions define it, rather than trust the procedure that produced it. The same rules also give a domain expert a concrete basis to attach human-meaningful activity names after discovery, so naming becomes an optional post-processing step rather than a prerequisite. A standard discovery algorithm then builds a process model over this interpretable, auditable alphabet.

More specifically, this paper makes the following contributions:

- We propose a domain-knowledge-free discovery pipeline that combines path-signature features with SyGuS to synthesize an interpretable rule alphabet and induce a sensor-derived event log (Section 3).
- We evaluate the discovered activities on two public sensor datasets (HACCP and SWaT) and compare them to an expert-guided method [8]. Even without any expert input, our activities cover almost all of the data and, on one dataset, are more accurate than the expert-guided method, while remaining behind it in the other cases (Section 4).

The remainder of the paper is organized as follows. Section 2 introduces the necessary preliminaries and positions our work against related event-abstraction approaches. Section 3 presents the synthesis pipeline. Section 4 reports the evaluation, and Section 5 discusses results, limitations, and open challenges.

2. Background

In this section, we introduce the necessary background for the paper. We first define the basic concepts used in later formalizations (Section 2.1). Then, we review related work (Section 2.2).

2.1. Preliminaries

We first introduce the basic concepts that define the input and output setting: sensor data, event data, and process models.

Definition 2.1 (Sensor data)**.** Let $d \in \mathbb{N}_{>0}$ be the number of sensor variables. A raw sensor log is a sequence $X = (x_0, \ldots, x_{N-1})$ with $x_t \in \mathbb{R}^d$ for each timestamp $t \in \{0, \ldots, N-1\}$. The component x_t^j is the raw sensor value of sensor j at timestamp t.

Sensor variables may be continuous, discrete, or binary before encoding. We assume that preprocessing maps them into real-valued vectors; and we use integer sample indices as the time domain.

Definition 2.2 (Event data)**.** An event alphabet A is a finite set of event types. An event is a tuple $e = (a, \tau^-, \tau^+)$ with type $a \in A$ and start and end timestamps τ^-, τ^+. A trace is a finite event sequence. An event log is a multiset of traces.

A case is one process instance represented by one trace. A case boundary separates two process instances in a sensor log. Classical process discovery receives an event log. In the present setting, A is not given and must be constructed from sensor behavior before discovery. We write the activity projection of a trace for the sequence of activity labels obtained by ignoring event timestamps.

Definition 2.3 (Process model)**.** A process model M over an event alphabet A defines a language $\mathcal{L}(M) \subseteq A^*$. Each word in $\mathcal{L}(M)$ is an activity sequence that the model permits.

2.2. Related Work

Event abstraction in process mining. Integrating IoT and process mining requires bridging low-level sensor data and high-level process concepts [1], and surveys consistently identify this *abstraction gap* as the central obstacle [6, 14]. Existing reviews [4, 5] show that most event-abstraction techniques are supervised and few address continuous data; supervised methods such as the pattern-based approach by Mannhardt et al. [15] require domain knowledge, and throughout, the activity vocabulary is provided or curated rather than synthesized from data.

Abstracting IoT sensor data into event logs. Work that targets sensor data directly splits along two dimensions: how interpretable the resulting activities are, and how much human input they require. Clustering-based methods are unsupervised, and thus closest to our setting, from smart-home event streams [16] to continuous industrial and smart-product streams [9, 17, 18]. These methods describe each cluster by statistics, centroids, or characteristic shapes, so they are not entirely opaque. The difference is one of form: cluster membership is implicit and relational, a segment's activity is decided by which centroid it is nearest to, so the boundary between activities is never stated and can only be recovered from the fitted model. An interval rule instead makes that boundary explicit, as a condition on named features that can be read off and checked on a single segment in isolation. Mapping clusters to named activities, or even selecting which features to cluster on, also still relies on a manual, domain-knowledge step [17, 18]. Conversely, the methods closest to ours in spirit produce interpretable, rule-like activity definitions: Seiger et al. interactively guide an analyst to detect and label activity executions and derive reusable sensor patterns [7], later generating complex-event-processing rules for online detection [11], while Sensor2EventLog [8] maps sensor behavior to named states through a machine-teaching loop. All three, however, obtain these rules from domain knowledge and analyst decisions; LLM labeling [10] is similarly convenient but yields labels rather than explicit rules that can be checked against the sensor data. Even closer to us in representation, Nadim et al. [19] extract decision-tree patterns, explicit conditions over variables, as events, but under supervision (a KPI-based labeling) and for causality analysis rather than the unsupervised synthesis of an event alphabet. Sensor2EventLog provides the quantitative reference for this paper. Its published diagnostics support a partial comparison with our rule metrics. Our pipeline excludes its teacher loop and expert event rules.

Building blocks: signatures and predicate synthesis. We represent each candidate segment of the sensor stream by path-signature features [12], an established feature-extraction method for multivariate time series [20] and sensor-based tasks [21]. As each such feature has a concrete meaning (for example,

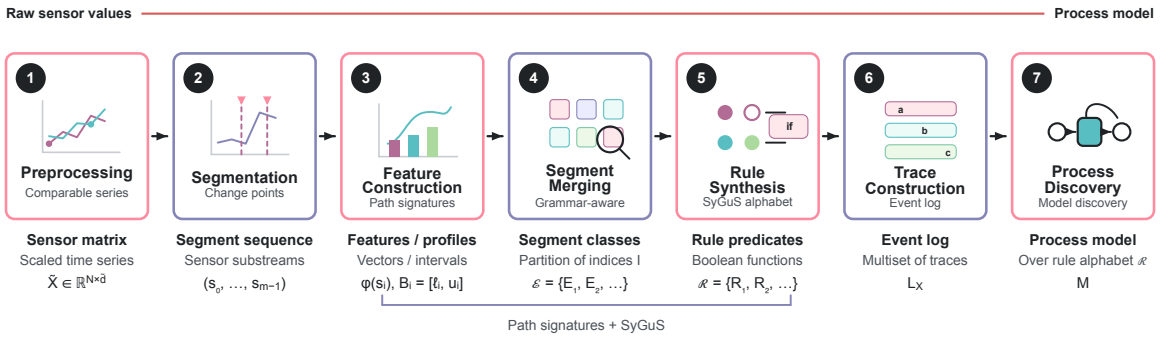


Figure 1: Overview of the preprocessing, abstraction, synthesis, and discovery steps, with output data types shown beneath each stage.

a per-sensor increment or a cross-sensor interaction), it can serve as a *named* feature space rather than an anonymous model input. This is what makes a rule-based alphabet worthwhile: where prior work feeds these features into a black-box classifier that returns a score, we synthesize over them an explicit predicate per activity, a condition, stated only in named features, that can be read, tested on a single segment, and later assigned a meaningful name. Constructing the alphabet is then a synthesis problem: syntax-guided synthesis [13] with SMT-based solvers [22, 23] and example-driven invariant learning [24] find predicates separating labeled examples. Whereas such predicate synthesis is normally used to explain or guard choices at existing decision points, we use it to construct the event alphabet itself.

Positioning. Our contribution makes sensor-derived event types explicit. Each event type has a predicate over named sensor features. The predicate provides a checkable definition beyond a cluster identifier or free-text LLM label [16, 9, 10]. The pipeline excludes expert event rules and iterative teacher feedback. Its configuration still requires human choices.

3. Pipeline

Figure 1 summarizes the implemented training procedure and the data types produced by its stages.[1] The configuration controls segmentation, feature construction, abstraction mode, synthesis bounds, and discovery thresholds.

Held-out evaluation reuses the fitted transformations and rules. Segmentation operates on the new sensor log. Rule applicability records segments with exactly one matching predicate.

3.1. Raw Data Acquisition

The input is a raw sensor log $X = (x_0, \ldots, x_{N-1})$. Each vector x_t contains the values recorded at timestamp t. No activity labels are available. Thus, the input is a raw multivariate stream, not yet a classical event log.

In the running example, $x_t = (p_t, q_t)$ contains position p_t and pressure q_t. The raw log records a path through the two-dimensional sensor space. The path is not yet a trace. It contains neither events nor activity names.

3.2. Preprocessing and Feature Scaling

Raw sensor logs cannot be used directly given that missing samples, heterogeneous units, and different value ranges make changes across channels difficult to compare. Therefore, preprocessing creates a

[1] https://github.com/chimenkamp/IoT-Only-Process-Discovery

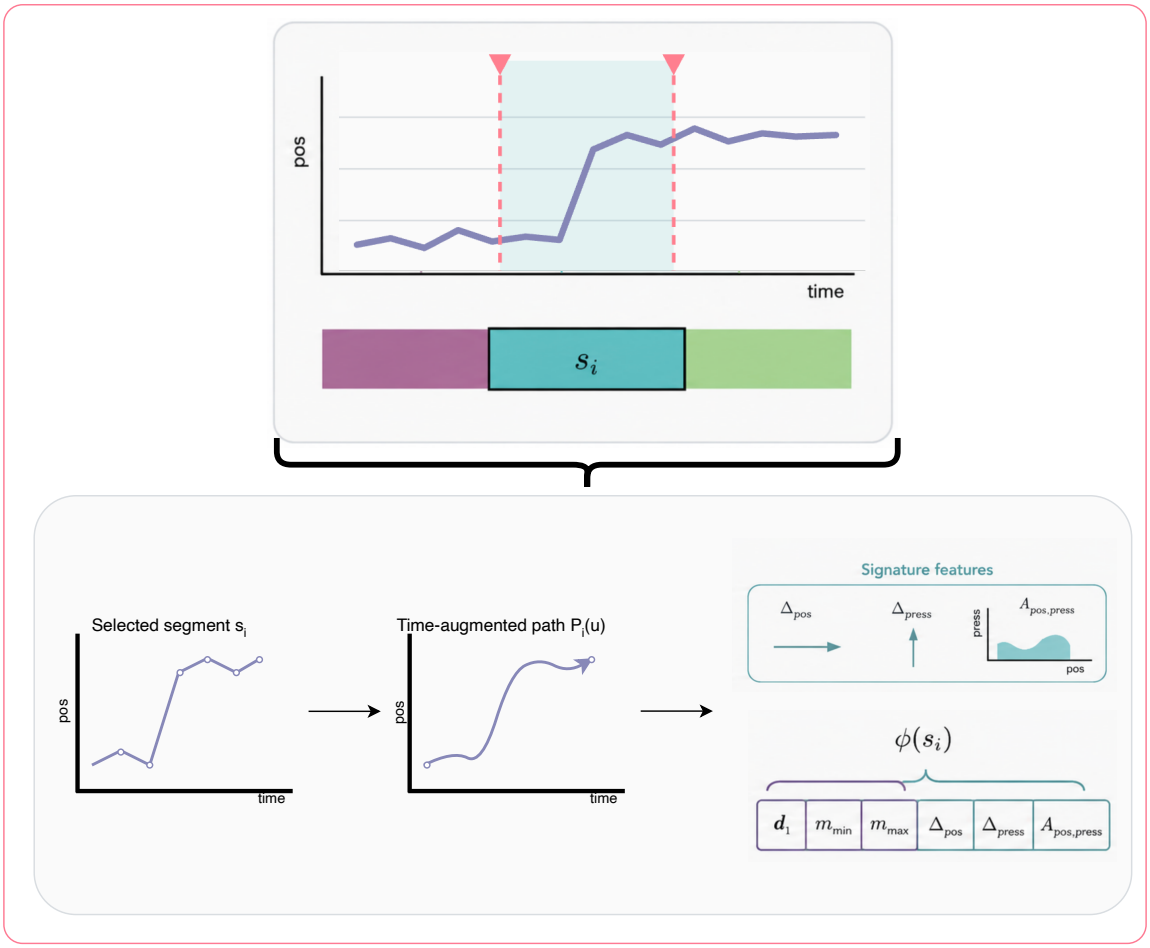


Figure 2: Change point detection highlights colored segment boundaries, and signature-aware abstraction extracts path features from each segment.

stable feature space in which later segmentation and rule synthesis depend on sensor behavior rather than on measurement scale.

Preprocessing maps X to a preprocessed log $\widetilde{X} = (\widetilde{x}_0, \ldots, \widetilde{x}_{N-1})$. The preprocessing function handles missing values, normalizes sensor scales, and may add delta channels. If preprocessing changes the number of channels, we denote the resulting dimension by $\widetilde{d}$. The preprocessing parameters, such as scaling constants and selected derived channels, are fitted once on the training log and stored as part of the learned procedure. They must be reused for validation logs so that interval predicates remain in the same feature space. Preprocessing settings remain design choices. Fitted scaling parameters make sensor variables comparable for later segmentation and synthesis.

Sensor variables may use incompatible units. For example, position and pressure may have different numerical ranges. A large range in one sensor should not dominate change point detection. Scaling therefore preserves the temporal structure while reducing unit effects. The time channel used for signatures is normalized to local segment time; absolute duration remains available as an envelope feature.

3.3. Candidate Change Point Detection

Change point detection partitions $\widetilde{X}$ into candidate sensor segments. It computes boundaries

$$0 = \tau_0 < \tau_1 < \cdots < \tau_m = N.$$

These boundaries define segments $s_i = \widetilde{X}_{\tau_i:\tau_{i+1}}$ for $i \in \{0, \ldots, m-1\}$.

The change point detector is a candidate generator. It uses an offline change point method [25, 26]. The detector should favor mild over-segmentation, since subsequent labeling and trace construction can merge repeated labels when the subsegments remain rule-equivalent. This mitigates the log-level impact of occasional spurious boundaries, whereas a missed boundary may conflate distinct activities and obscure the underlying process structure.

In the running example, this step produces three segments. The first segment contains rightward movement. The second segment contains pressure increase. The third segment contains leftward movement. Still, all segments are unlabeled.

3.4. Signature-Aware Segment Abstraction

Each segment is then converted into a feature vector. For each segment s_i, the feature map ϕ computes envelope features and truncated signature features. Envelope features describe value ranges. Signature features describe movement along the segment. The lower part of Figure 2 shows the selected segment s_i, its time-augmented path $P_i(u)$, and the resulting feature vector $\phi(s_i)$.

Following Chevyrev and Kormilitzin [12], the signature of a path $P : [a, b] \to \mathbb{R}^q$ is the sequence of iterated integrals

$$S(P)_{a,b} = \left(1,\, S^1(P)_{a,b}, \ldots, S^q(P)_{a,b},\, S^{1,1}(P)_{a,b}, S^{1,2}(P)_{a,b}, \ldots\right).$$

For a word $(i_1, \ldots, i_\ell)$ over path coordinates, the corresponding term is

$$S^{i_1,\ldots,i_\ell}(P)_{a,b} = \int_{a<u_1<\cdots<u_\ell<b} dP^{i_1}_{u_1} \cdots dP^{i_\ell}_{u_\ell}.$$

A truncated signature keeps only terms up to a fixed level K.

For signature computation, the discrete samples in s_i are converted into a piecewise-linear interpolation $\bar{x}_i : [0, 1] \to \mathbb{R}^{\widetilde{d}}$. The time-augmented path

$$P_i(u) = (u, \bar{x}_i(u)), \quad u \in [0, 1],$$

is then used for signature computation. Level-one signature terms are total increments. Level-two signature terms capture signed areas and cross-sensor order. Duration is included separately among the envelope features. Thus, $\phi(s_i)$ can represent direction, duration, and coordination between sensors without letting raw time scale dominate mixed signature terms.

This representation connects sensor measurements to rule synthesis. Raw values alone are too fine-grained to serve as activity labels. Segment features provide the finite examples used by SyGuS. The running example yields features such as duration, minimum position, maximum position, position increment, pressure increment, and a signed area term.

3.5. Grammar-Aware Segment Merging

Segment merging coarsens an initial partition of segment occurrences. The occurrence indices form $I = \{0, \ldots, m-1\}$. The default `interval` mode starts with singleton classes. The `clustered_interval` mode starts with k K-means classes (i.e., 20 initializations and a configured random seed). The configuration supplies k.

Figure 3 illustrates how feature profiles from repeated segment occurrences are grouped into classes.

Class conflicts determine each merge. The profile $B_i = [\ell_i, u_i]$ contains the feature values of segment i. The hull $H_E = \prod_j [\min_{i \in E} \ell_i^j, \max_{i \in E} u_i^j]$ encloses class E. Two distinct classes conflict exactly when

$$E \sim_c E' \quad \Longleftrightarrow \quad (\exists\, i \in E' :\ H_E \cap B_i \neq \varnothing) \ \vee\ (\exists\, i \in E :\ H_{E'} \cap B_i \neq \varnothing).$$

The intersection test includes touching boundaries.

Each pass merges the connected components of the conflict graph. The next pass recomputes the class hulls. The procedure stops after a pass without a merge. It does not search all partitions or minimize their class count. Rule synthesis follows merging and can still fail under its configured bounds.

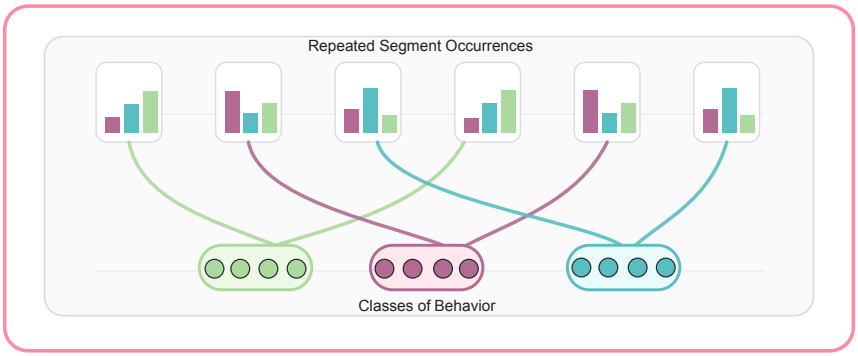


Figure 3: Grammar-aware merging groups signature feature examples from repeated segment occurrences into classes.

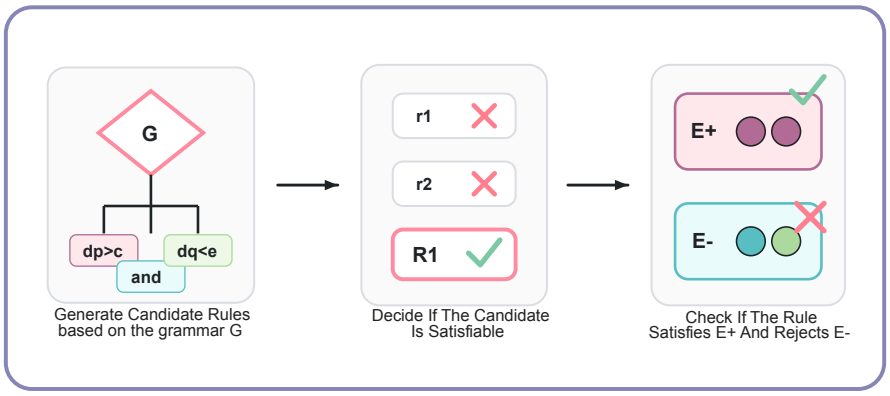


Figure 4: SyGuS uses the grammar to build candidate predicates and selects one that covers E^+ while rejecting E^-.

3.6. SyGuS Rule Synthesis

In a SyGuS problem, a grammar $\mathcal{G}$ defines the syntactic form of candidate expressions, a formula ψ defines the semantic specification, and the solution is an expression $e \in \mathcal{L}(\mathcal{G})$ satisfying ψ. Here, the variables of the expression are the named coordinates of $\phi(s)$, and the Boolean fragment of $\mathcal{G}$ generates predicates over segment features.
Rule synthesis constructs one rule for each segment class. For class E_i, the positive examples are

$$\mathrm{Pos}_i = \{\phi(s_h) \mid h \in E_i\}.$$

The negative examples are

$$\mathrm{Neg}_i = \{\phi(s_h) \mid h \in I \setminus E_i\}.$$

Figure 4 illustrates grammar-based predicate synthesis and the checks against positive and negative examples, denoted by E^+ and E^- in the figure.

SyGuS solves the specification

$$\exists e_i \in \mathcal{L}_{\mathsf{Bool}}(\mathcal{G}) : (\forall p \in \mathrm{Pos}_i : \mathrm{den}(e_i)(p) = \top) \wedge (\forall n \in \mathrm{Neg}_i : \mathrm{den}(e_i)(n) = \bot)\,,$$

where $\mathcal{L}_{\mathsf{Bool}}(\mathcal{G})$ is the Boolean fragment generated from the rule start symbol and $\mathrm{den}(e_i)$ is the predicate denoted by expression e_i. The variables of e_i are the named coordinates of the feature vector. The resulting rule is $R_i = \mathrm{den}(e_i)$.

In this pipeline `Linear Real Arithmetic (LRA)` is used as the logic. The grammar permits $\top$ and conjunctions of atoms $c \le z_j$ or $z_j \le c$ over the d_ϕ profile features. The candidate constants

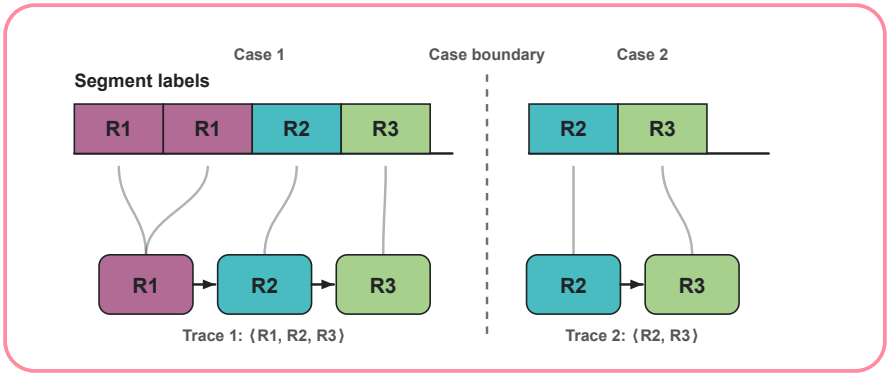


Figure 5: Consecutive equal rule labels collapse within each case. The case boundary is preserved, yielding two separate activity projections.

for feature j are its positive-class hull bounds expanded by the configured margin. The atom limit is configurable (i.e., $\max(1, 2d_\phi)$ by default and four in the current evaluation drivers). The constraints cover margin-expanded positive profiles and reject unexpanded negative profiles. Synthesis failure aborts the pipeline without reducing the margin or relaxing the atom limit.

In the running example, SyGuS can synthesize three rules. One rule covers positive position increments and near-zero pressure increments. One rule covers near-zero position increments and positive pressure increments. One rule covers negative position increments and near-zero pressure increments. These rules become the event alphabet $\mathcal{R}$.

3.7. Case Detection and Trace Construction

Trace construction assumes sequential cases within each sensor stream. Case boundaries must be supplied or detected by an explicit deterministic rule. The segmentation must preserve these boundaries. Concurrent or interleaved cases require a separate correlation method.

Each training segment receives its unique rule label. Consecutive equal labels collapse within each case. A maximal block $s_\ell, \ldots, s_r$ with label R_j becomes $(R_j, \tau_\ell, \tau_{r+1})$. Held-out segments can activate zero or multiple rules. These segments remain unresolved and reduce rule applicability.

Case definitions determine the interpretation of the discovered traces. These artificial boundaries can split executions or merge unrelated behavior. A recurring start rule would provide only a heuristic boundary without independent validation.

In Figure 5, the case boundary separates the activity projections $\langle R_1, R_2, R_3\rangle$ and $\langle R_2, R_3\rangle$. The two initial R_1 segments collapse into one event within the first case. The repeated pressure–left pair belongs to the second case and does not imply a within-case transition from R_3 back to R_2. A rework loop would instead require repetition within the same case, such as $\langle R_1, R_2, R_3, R_2, R_3\rangle$.

3.8. Process Discovery

Process discovery is applied to L_X. The input is now an event log over $A = \mathcal{R}$. Process discovery operates on activity projections, while event timestamps remain preserved in the log. Accordingly, any discovery algorithm that is compatible with this induced event-log representation can be employed. A concrete implementation may use the Inductive Miner [27].

This step produces the model component M of the solution. The model language $\mathcal{L}(M)$ describes permitted sequences of synthesized rules. The resulting model is exploratory because its activity semantics are derived from sensor behavior rather than from analyst labels.

For the two cases in Figure 5, Figure 6 permits either R_1 followed by R_2 and R_3, or R_2 followed by R_3. The silent transition τ skips R_1 for the second case. The model contains no loop from R_3 back to

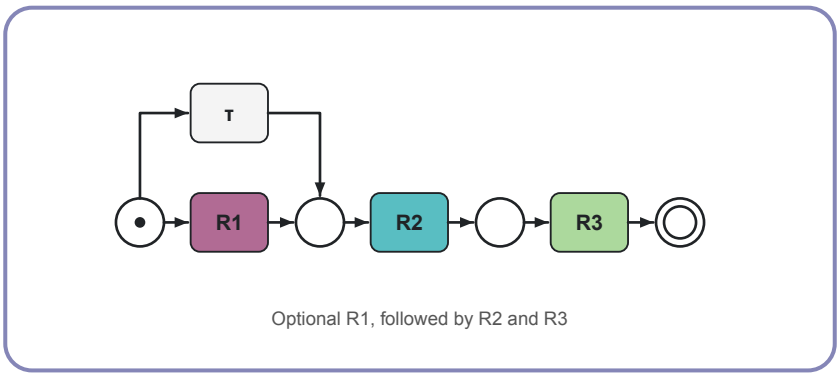


Figure 6: Illustrative process model for the two case projections in Figure 5. The silent transition τ makes R_1 optional; no rework loop is implied by repetition across cases.

Table 1
Benchmark setup for the Sensor2EventLog comparison.

Dataset	Batches	Train	Test	Sensors	Ref. states	Rules
PM_HACCP_PASTEURIZATION	968	581	387	8	6	6
SWaT.A4_A5_Jul_2019_P1	4	2	2	5	2	3

R_2, since the illustrated repetition occurs across a case boundary. Visible transitions are annotated with synthesized rule activities; τ denotes an unobservable transition and is not part of the event alphabet.

4. Evaluation

The evaluation provides a partial comparison with Sensor2EventLog [8]. Sensor2EventLog uses a teacher to select states and refine event rules. Our experiments use zero teacher iterations and zero expert event rules. Segmentation settings, feature choices, initial class counts, and case definitions remain prescribed inputs. Our experimental artifacts are available online.[2]

Sensor2EventLog's domain-specific thresholds are excluded from our pipeline. These thresholds define the baseline's expert rules (e.g., $Q_{in} > 0.3$ and $T > 70$). They explain the baseline's additional human guidance. They do not constrain our segmentation or rule construction. Reference-state labels support scoring after discovery.

HACCP provides 968 labeled batches. The split uses 581 training batches and 387 test batches with eight sensor columns. The corresponding segment counts are 5,551 and 3,699. The dataset's `state` column supplies the six names (i.e., Idle, Fill, HeatUp, Hold, Cool, and Discharge). The identifiers A01–A06 denote automatically constructed predicates. The reference names describe post-hoc matches and are not generated rule names.

SWaT provides only a small diagnostic sample. The selected P1 window spans 2019-07-20 04:35–06:50 UTC (i.e., 135 minutes). The workbook lacks the prepared batch identifiers used by Sensor2EventLog. Four equal contiguous pseudo-batches provide two training cases and two test cases with 15 segments each. The input contains five sensor columns and two workbook states. The held-out split contains only one reference state.

The result summaries record fixed pipeline parameters. The segmentation penalties are 6.0 for HACCP and 4.0 for SWaT. Both runs use a minimum segment size of 20 samples and signature depth $K = 1$. The rule margin is 2.0 in the scaled feature space. The feature limits are maximal (i.e., 40 for HACCP and 30 for SWaT). Depth one captures increments but excludes higher-order sensor interactions.

[2] https://github.com/chimenkamp/IoT-Only-Process-Discovery-

Table 2
Partial HACCP comparison. Ours averages six states; Sensor2EventLog averages four published rule–state pairs.

Approach	Rules	Iter.	Domain	Coverage	Precision	Effect.
Ours	6	0	0	0.749250	0.739081	0.700088
Sensor2EventLog	4	3	4	0.963000	0.970000	0.966309

Table 3
HACCP reference labels and best matching anonymous rules on the test split.

State	Rule	State sup.	Rule sup.	Overlap	Coverage	Precision	Effect.
Idle	A03	127391	79225	56566	0.444035	0.713992	0.563060
Fill	A01	127579	63680	63445	0.497300	0.996310	0.703892
HeatUp	A05	242321	188866	188611	0.778352	0.998650	0.881647
Hold	A02	346526	651618	346318	0.999400	0.531474	0.728804
Cool	A04	243033	189360	189131	0.778211	0.998791	0.881629
Discharge	A02	127469	651618	127240	0.998203	0.195268	0.441494

Rule metrics quantify agreement with reference states. The set T_y contains timestamps with state y. The value $r_t^{(m)} \in \{0, 1\}$ indicates activation of rule R_m at timestamp t. Coverage measures the captured fraction of a state. Precision measures the matching fraction of a rule's activations. Effectiveness is their geometric mean.

$$\mathrm{Cov}(y, R_m) = \frac{\sum_{t \in T_y} r_t^{(m)}}{|T_y| + \epsilon}, \quad \mathrm{Prec}(y, R_m) = \frac{\sum_{t \in T_y} r_t^{(m)}}{\sum_{t=1}^{T} r_t^{(m)} + \epsilon}.$$

$$\mathrm{Eff}(y, R_m) = \sqrt{\mathrm{Cov}(y, R_m) \cdot \mathrm{Prec}(y, R_m)}.$$

The best rule for each state maximizes effectiveness. Ties use coverage and then precision and overlap. Mean scores average the selected rules over reference states. State explainability instead selects the maximum coverage for each state.

Timestamp accuracy uses a mapping learned after discovery. Each rule receives its majority training state for evaluation. The fixed mapping then scores test timestamps. HACCP accuracy is 0.729043. SWaT accuracy uses the workbook's P1_STATE labels.

The HACCP comparison uses different averaging sets. Our scores average the best rules for all six reference states. The baseline coverage and precision average four published pairs (i.e., Fill, Hold, HeatUp, and Cool) from Sensor2EventLog Tables 2 and 4. We recompute baseline effectiveness as the mean of the four geometric means. The baseline values are publication-derived and do not come from a local rerun. Their larger averages do not isolate the effect of teacher guidance.

Rule A02 does not distinguish Hold from Discharge. Both reference states select A02 as their best matching rule. Separate dataset annotations support two reference states but do not establish two discovered event types. Repeated A02 occurrences can occupy different trace positions. Their shared predicate alone cannot recover the semantic distinction.

HACCP rule applicability reaches 0.999730 on test segments. Exactly 3,698 of 3,699 test segments receive one rule label (i.e., 99.9730%). The implemented conformance check yields fitness of 0.999036 on the induced test traces. This value measures acceptance of rule traces and does not validate their activity semantics.

SWaT precision requires a narrow interpretation. The best rule achieves precision of 1.000000 on the sole held-out reference state. This split cannot test discrimination between states. The rule covers only 0.561759 of that state's timestamps. Exactly 13 of 15 test segments receive a unique rule label (i.e., 86.6667%). The induced model's test fitness is 0.625000.

The SWaT baselines come from Sensor2EventLog Table 5. The published accuracies are 0.760000 for HMM and 0.940000 for K-means after teacher refinement. The table reports two iterations and two

Table 4
Preliminary SWaT diagnostic versus published baselines. Dashes denote unavailable comparison metrics.

Approach	Acc.	Coverage	Precision	Effect.	Applic.	Iter.	Domain
Ours	0.805089	0.561759	1.000000	0.749506	0.866667	0	0
Sensor2EventLog HMM	0.760000	–	–	–	–	2	2
Sensor2EventLog K-means	0.940000	–	–	–	–	2	2

rules for each learner. The publication evaluates one day of normal operation with different case and state definitions. Our 135-minute pseudo-batch diagnostic is not a reproduction or evidence of superior generalization.

Metric coverage remains incomplete across methods. Our artifacts provide rule metrics, mapped accuracy, applicability, and model fitness for both datasets. The HACCP baseline provides the four selected rule pairs but lacks a matched six-state rule average. The SWaT baseline provides accuracy and a state-explainability lower bound (i.e., above 0.78). It lacks comparable aggregate rule metrics and applicability. The tables therefore support only a partial comparison.

5. Discussion

Explicit predicates make the event abstraction inspectable. HACCP rules label almost all test segments but merge some reference distinctions. SWaT provides preliminary evidence from four pseudo-batches and 30 segments. These results support diagnostic use of the rules. They do not establish a unique activity vocabulary or generalization across operating conditions.

The abstraction remains conditional on human design choices. Segmentation, features, grammar, initial class counts, and case boundaries determine the resulting event types. Environmental responses can satisfy the same predicates as process activities. Domain interpretation therefore remains a separate task. High process-model fitness only establishes agreement with the induced rule traces.

Acknowledgment: This work received funding from the Deutsche Forschungsgemeinschaft (DFG), grant 496119880.

Declaration on Generative AI: During the preparation of this work, the authors used LanguageTool/Grammarly for grammar and spelling checking. Codex was used to debug the implementation. After using these tools, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] C. Janiesch, A. Koschmider, M. Mecella, B. Weber, A. Burattin, C. Di Ciccio, G. Fortino, A. Gal, U. Kannengiesser, F. Leotta, F. Mannhardt, A. Marrella, J. Mendling, A. Oberweis, M. Reichert, S. Rinderle-Ma, E. Serral, W. Song, J. Su, V. Torres, M. Weidlich, M. Weske, L. Zhang, The internet of things meets business process management: A manifesto, IEEE Systems, Man, and Cybernetics Magazine 6 (2020) 34–44. doi:10.1109/MSMC.2020.3003135.

[2] W. M. P. van der Aalst, Process Mining: Data Science in Action, 2nd ed., Springer, 2016.

[3] W. van der Aalst, Process mining: Overview and opportunities, ACM Trans. Manage. Inf. Syst. 3 (2012). URL: https://doi.org/10.1145/2229156.2229157. doi:10.1145/2229156.2229157.

[4] S. J. van Zelst, F. Mannhardt, M. de Leoni, A. Koschmider, Event abstraction in process mining: literature review and taxonomy, Granular Computing 6 (2021) 719–736.

[5] K. Diba, K. Batoulis, M. Weidlich, M. Weske, Extraction, correlation, and abstraction of event data for process mining, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 10 (2020) e1346.

[6] E. Brzychczy, M. Aleknonytė-Resch, D. Janssen, A. Koschmider, Process mining on sensor data: a review of related works: E. brzychczy et al., Knowledge and Information Systems 67 (2025) 4915–4948.
[7] R. Seiger, M. Franceschetti, B. Weber, An interactive method for detection of process activity executions from iot data, Future Internet 15 (2023) 77.
[8] A. Moradbeikie, I. M. Grigore, S. I. Lopes, S. Barbon Junior, Sensor2eventlog: Bridging continuous iot data and process mining through eventization, in: L. Fuentes, P. Plebani, C. Combi, H. Reijers (Eds.), Advanced Information Systems Engineering, Springer Nature Switzerland, Cham, 2026, pp. 177–194.
[9] E. J. Husom, A. Goknil, F. Mannhardt, S. Tverdal, S. Sen, P. Nguyen, Unsupervised learning and process analysis for sensor data validation in the iiot, ACM Transactions on Internet Technology 26 (2026) 1–34.
[10] M. Shirali, M. F. Sani, Z. Ahmadi, E. Serral, Llm-based event abstraction and integration for iot-sourced logs, in: International Conference on Business Process Management, Springer, 2024, pp. 138–149.
[11] R. Seiger, A. F. Kurz, M. Franceschetti, Online detection of process activity executions from iot sensors using generated event processing services, Future Generation Computer Systems 174 (2026) 107987.
[12] I. Chevyrev, A. Kormilitzin, A Primer on the Signature Method in Machine Learning, Springer Nature Switzerland, Cham, 2026, pp. 3–64. URL: https://doi.org/10.1007/978-3-031-97239-3_1. doi:10.1007/978-3-031-97239-3_1.
[13] R. Alur, R. Bodik, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, A. Udupa, Syntax-guided synthesis, in: 2013 Formal Methods in Computer-Aided Design, 2013, pp. 1–8. doi:10.1109/FMCAD.2013.6679385.
[14] Y. Bertrand, B. Van den Abbeele, S. Veneruso, F. Leotta, M. Mecella, E. Serral, A survey on the application of process mining to smart spaces data, in: M. Montali, A. Senderovich, M. Weidlich (Eds.), Process Mining Workshops, Springer Nature Switzerland, Cham, 2023, pp. 57–70.
[15] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, P. J. Toussaint, From low-level events to activities - a pattern-based approach, in: M. La Rosa, P. Loos, O. Pastor (Eds.), Business Process Management, Springer International Publishing, Cham, 2016, pp. 125–141.
[16] D. Janssen, F. Mannhardt, A. Koschmider, S. J. van Zelst, Process model discovery from sensor event data, in: S. Leemans, H. Leopold (Eds.), Process Mining Workshops, Springer International Publishing, Cham, 2021, pp. 69–81.
[17] M. Mayr, S. Luftensteiner, G. C. Chasparis, Abstracting process mining event logs from process-state data to monitor control-flow of industrial manufacturing processes, Procedia Computer Science 200 (2022) 1442–1450.
[18] M. L. Van Eck, N. Sidorova, W. M. Van der Aalst, Enabling process mining on sensor data from smart products, in: 2016 IEEE tenth international conference on research challenges in information science (RCIS), IEEE, 2016, pp. 1–12.
[19] K. Nadim, A. Ragab, M.-S. Ouali, Data-driven dynamic causality analysis of industrial systems using interpretable machine learning and process mining, Journal of Intelligent Manufacturing 34 (2023) 57–83.
[20] J. Morrill, A. Fermanian, P. Kidger, T. Lyons, A generalised signature method for multivariate time series feature extraction, arXiv preprint arXiv:2006.00873 (2020).
[21] A. Fermanian, Embedding and learning with signatures, Computational Statistics & Data Analysis 157 (2021) 107148.
[22] A. Reynolds, H. Barbosa, A. Nötzli, C. Barrett, C. Tinelli, cvc4sy: Smart and fast term enumeration for syntax-guided synthesis, in: I. Dillig, S. Tasiran (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2019, pp. 74–83.
[23] L. de Moura, N. Bjørner, Z3: An efficient smt solver, in: C. R. Ramakrishnan, J. Rehof (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 337–340.

[24] P. Garg, D. Neider, P. Madhusudan, D. Roth, Learning invariants using decision trees and implication counterexamples, in: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 499–512. URL: https://doi.org/10.1145/2837614.2837664. doi:10.1145/2837614.2837664.

[25] R. Killick, P. Fearnhead, I. A. Eckley, Optimal detection of changepoints with a linear computational cost, Journal of the American Statistical Association 107 (2012) 1590–1598. URL: https://doi.org/10.1080/01621459.2012.737745. doi:10.1080/01621459.2012.737745. arXiv:https://doi.org/10.1080/01621459.2012.737745.

[26] C. Truong, L. Oudre, N. Vayatis, Selective review of offline change point detection methods, Signal Processing 167 (2020) 107299. URL: https://www.sciencedirect.com/science/article/pii/S0165168419303494. doi:https://doi.org/10.1016/j.sigpro.2019.107299.

[27] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering block-structured process models from event logs containing infrequent behaviour, in: N. Lohmann, M. Song, P. Wohed (Eds.), Business Process Management Workshops, Springer International Publishing, Cham, 2014, pp. 66–78.