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]*Professor and Chair of Data Systems, Hasso Plattner Institute: Potsdam, Brandenburg*

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, but process discovery expects 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.

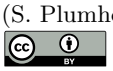
However, sensor data is not the kind of input that process discovery expects. Classical discovery requires an event log in which every event refers to a case, an *activity*, and a timestamp [3]. IoT devices supply only the timestamp: they emit continuous, multivariate streams with no activity names, 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]. Constructing this

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)



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, 5]. Supervised and expert-guided methods assume that the relevant activities are already known and merely need to be recognized [6, 7]. Recent unsupervised methods avoid this assumption but produce alphabets that cannot be inspected: cluster identifiers are opaque [8], LLM-generated labels are stochastic and not verifiable against the sensor data [9], and rule-based detectors can only fire on activities an expert already annotated [10]. 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.

This paper. We address this missing link directly. 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 [11]), and apply syntax-guided synthesis (SyGuS) [12] 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. 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 [7]. 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 objects used in the formalization (Section 2.1). Then, review related work (Section 2.2).

2.1. Preliminaries

We first introduce the basic objects 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. This paper assumes that preprocessing maps them into real-valued vectors. We use integer sample indices as the time base.

Definition 2.2 (Event data)**.** Let A be a finite activity alphabet. An activity is one element $a \in A$. An event is a tuple $e = (a, \tau^-, \tau^+)$, where a is an activity and τ^-, τ^+ are start and end timestamps. A trace is a finite sequence of events. 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 activity 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 [5, 13]. The taxonomy of van Zelst et al. [4] shows that most event-abstraction techniques are supervised and few address continuous data; supervised methods such as the behavioral activity patterns of Mannhardt et al. [14] require domain knowledge, and throughout, the activity vocabulary is provided or curated rather than synthesized from the 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 [15] to continuous industrial and smart-product streams [8, 16, 17]. These methods do describe each cluster by statistics, centroids, or characteristic shapes, so they are not wholly 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 [16, 17]. 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

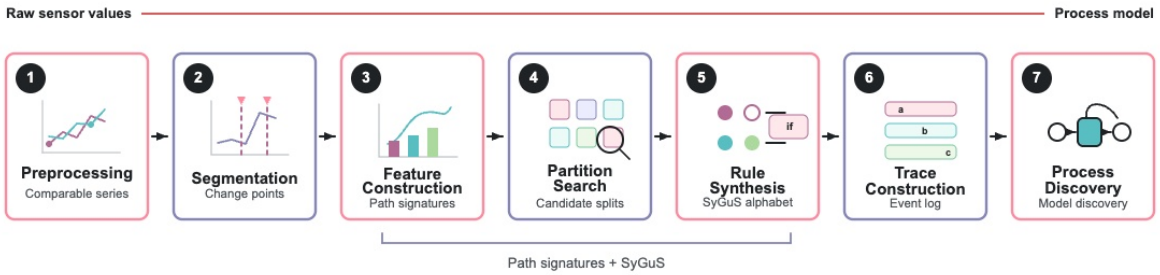


Figure 1: Overview of the different preprocessing, abstraction, synthesis, and discovery steps.

executions and derive reusable sensor patterns [6], later generating complex-event-processing rules for online detection [10], while Sensor2EventLog [7] 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 [9] 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. [18] 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 activity alphabet. We adopt Sensor2EventLog as our quantitative baseline, as it is the most recent and directly comparable, but require none of its teacher loop or domain rules. In short, no prior approach combines the unsupervised setting of clustering with the interpretable activity definitions of the rule-based methods, which is precisely the gap we address.

Building blocks: signatures and predicate synthesis. We represent each candidate segment of the sensor stream by path-signature features [11], an established feature-extraction method for multivariate time series [19] and sensor-based tasks [20]. Because each such feature has a concrete meaning (for example, 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 human-meaningful name. Constructing the alphabet is then a synthesis problem: syntax-guided synthesis [12] with SMT-based solvers [21, 22] and example-driven invariant learning [23] find predicates separating labeled examples. Whereas such predicate synthesis is normally used to explain or guard choices at decision points that already exist, we use it to construct the activity alphabet itself.

Positioning. Our contribution differs along three axes: *supervision* (no labels, domain rules, or analyst decisions, unlike [14, 7, 10]); *interpretability* (each activity is an explicit interval predicate over named sensor features that can be checked on a single segment, unlike the opaque cluster identifiers of [15, 8] or free text LLM LLM labels [9]); and *representation* (activities are synthesized as rules fit to the data rather than supplied as a curated vocabulary). To our knowledge, this combination of an unsupervised setting, continuous sensor data, and an interpretable, rule-based activity alphabet is not offered by existing work.

3. Pipeline

This section presents the discovery procedure. The procedure follows the structure shown in Figure 1. It transforms raw sensor values into a process model through (1) preprocessing, (2) segmentation, (3) feature construction, (4) partition search, (5) rule synthesis, (6) trace

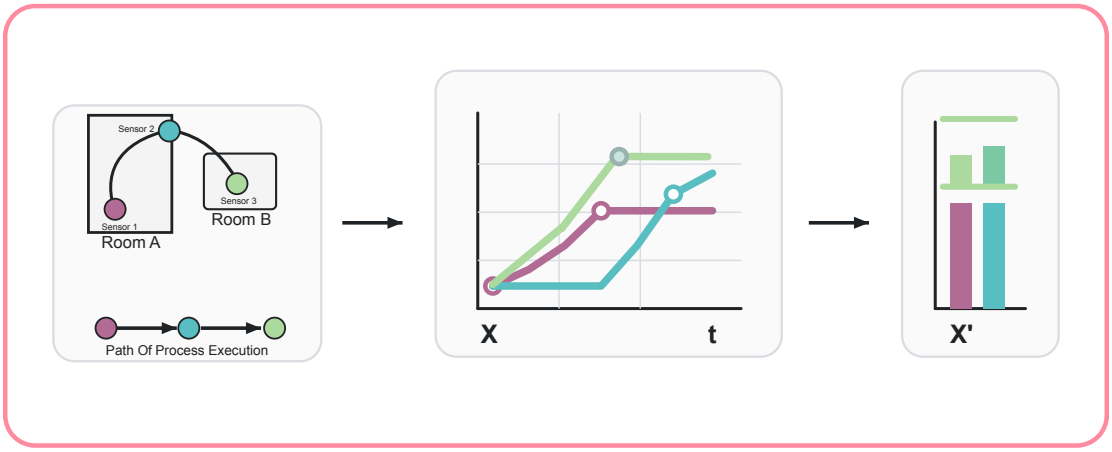


Figure 2: Connected sensors produce raw time series, which preprocessing maps to comparable features.

construction, and (7) process discovery. The procedure combines path signatures with SyGuS to construct the missing activity alphabet.

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 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 does not add domain knowledge. It only makes sensor variables comparable for later segmentation and synthesis. The mapping from connected sensor measurements to comparable feature channels is summarized in Figure 2.

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.

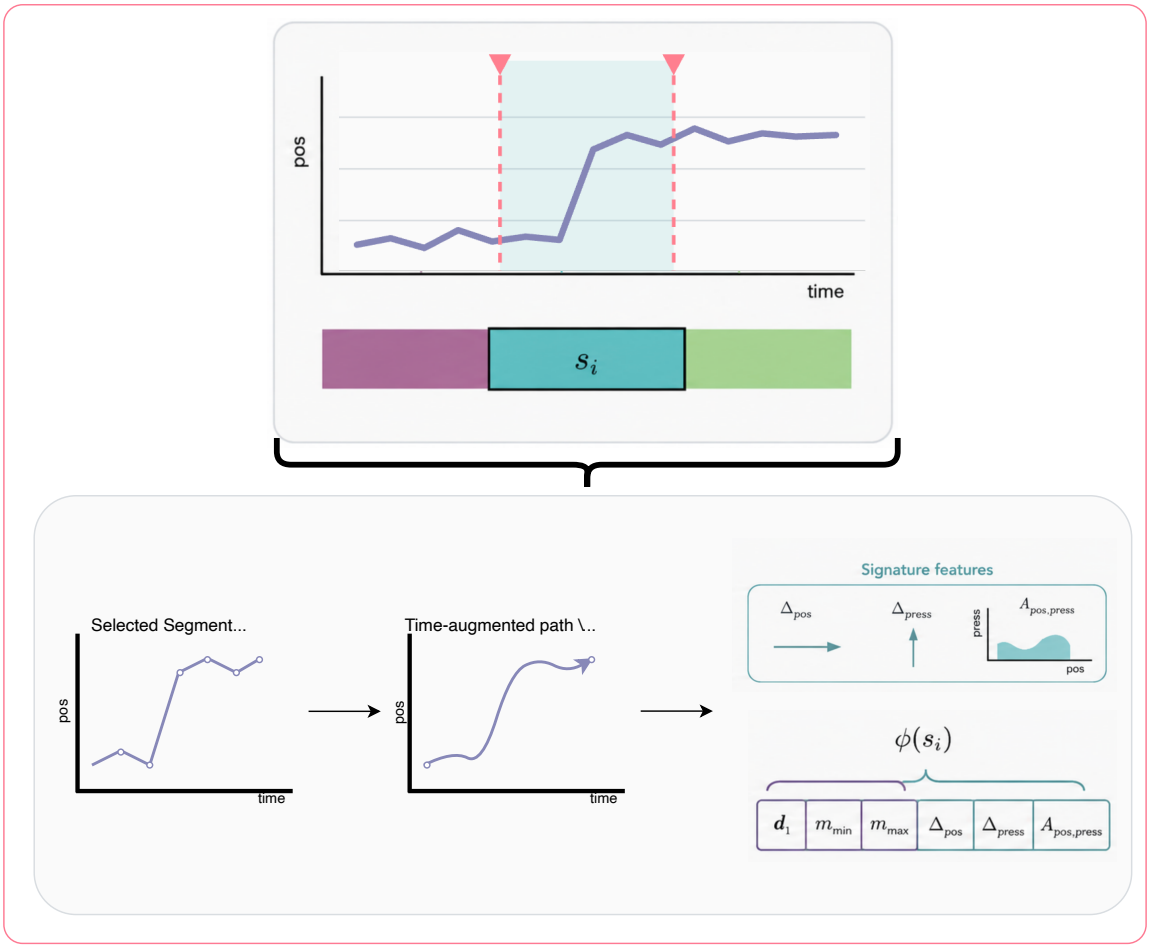


Figure 3: Change point detection highlights colored segment boundaries, and signature-aware abstraction extracts path features from each segment.

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 [24, 25]. 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 right side of Figure 3 highlights one candidate segment as a path and the feature vector extracted from it.

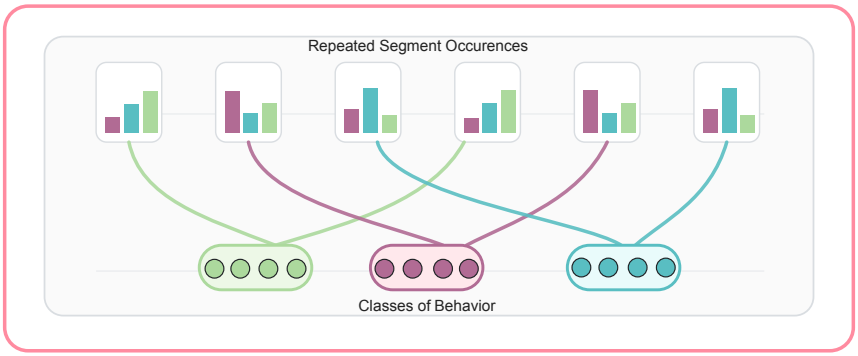


Figure 4: Grammar-aware merging groups signature feature examples from repeated segment occurrences into classes.

Following Chevyrev and Kormilitzin [11], 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 groups compatible segment occurrences into segment classes. Let $\mathbf{s} = (s_0, \ldots, s_{m-1})$ be the candidate segment occurrence sequence and let $I = \{0, \ldots, m-1\}$ be its occurrence indices. The merging step searches for a partition $\mathcal{E} = \{E_1, \ldots, E_k\}$ of I. Each class E_i should be explainable by one rule in the grammar $\mathcal{G}$.

The partition is abductive: the observations are the feature vectors $\phi(s_i)$, the theory is the grammar $\mathcal{G}$ and the hypotheses correspond to segment classes. The search space is constrained rather than spanning arbitrary partitions. Let $\sim_c$ be a symmetric compatibility relation over segment occurrences, fixed by the method from selected envelope and signature features. A candidate class E_i is compatible if all pairs of occurrence indices in E_i satisfy $\sim_c$. A partition is admissible if every class is compatible and, for each class, a Boolean expression generated by $\mathcal{G}$ can separate its examples from the examples of the other classes. The one-class partition is admissible only if all segment occurrences are pairwise compatible under $\sim_c$. The preferred explanation uses the fewest classes among admissible partitions.

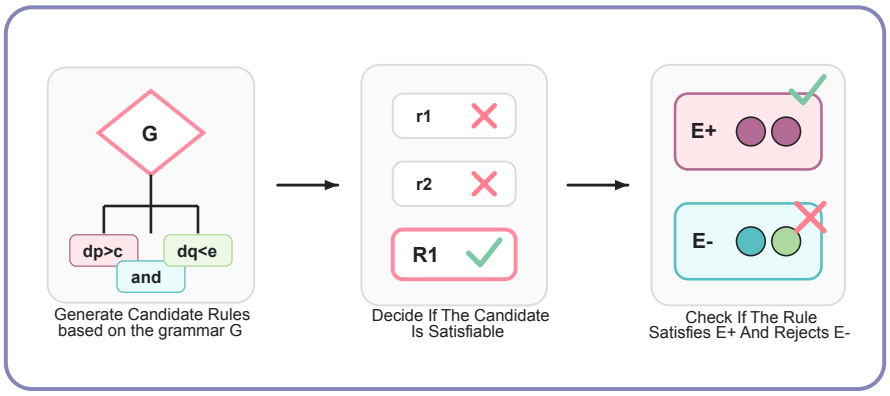


Figure 5: SyGuS uses the grammar to build candidate predicates and selects one that covers E^+ while rejecting E^-.

From an algorithmic perspective, this step constitutes a grammar-aware partition search. It may reuse the SyGuS solver from the subsequent stage as a feasibility oracle. The synthesis step that follows then produces a final rule for each selected class.

This step determines the size of the activity alphabet and introduces new activities only when necessitated by the feature space and the grammar. In the running example, the compatibility relation distinguishes three movement patterns, yielding three behavioral segment classes. Across repeated executions, segments with equivalent behavior are consistently assigned to the same class.

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

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

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 $R_i = \mathrm{den}(e_i)$. The grammar $\mathcal{G}$ can use interval predicates over named segment features. This keeps the rules interpretable. The semantic constraints are finite because they range over segments rather than timestamps; the search itself is finite when the grammar depth and candidate constants are bounded.

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 activity alphabet $\mathcal{R}$.

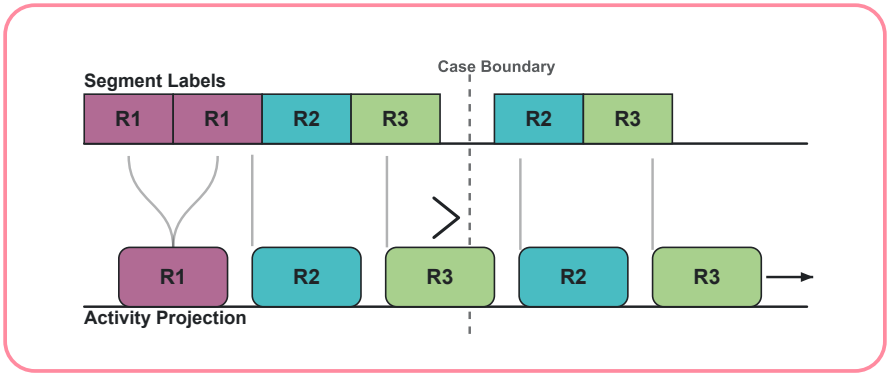


Figure 6: Rule activations are collapsed into a sensor-derived activity projection.

3.7. Case Detection and Trace Construction

Trace construction converts rule activations into traces. Each segment occurrence s_i receives the unique label $\lambda(i) = R_j$ for which $R_j(\phi(s_i)) = \top$. Known case boundaries split the segment occurrence sequence before collapse. They act as hard constraints: if a known case boundary falls within a candidate segment, the segment must be split, or the segmentation procedure must be constrained to respect this boundary. Consecutive repetitions of the same rule are then collapsed only within each case. A maximal case-local block $s_\ell, \ldots, s_r$ with label R_j becomes one timestamped event $(R_j, \tau_\ell, \tau_{r+1})$.

When case boundaries are not given, recurrence of a start-candidate rule can serve as an exploratory case heuristic. The start candidate may be the first observed rule in the stream or a rule selected by a fixed frequency or periodicity heuristic. The choice uses only the synthesized rule sequence and should therefore be treated as an explicit assumption of the induced log. Later evaluation must test whether this heuristic matches meaningful process instances.

In the running example, the activity projection of the segment labels is $\langle R_1, R_2, R_3 \rangle$. If the same process instance repeats the pressure and left-movement pair as rework, the projected trace may become $\langle R_1, R_2, R_3, R_2, R_3 \rangle$. If the repetition belongs to a different process instance, it becomes part of another trace. Such traces reveal loops to process discovery only when repetitions occur within the same case.

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 [26].

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 running example, a sequential model permits R_1 followed by R_2 followed by R_3. If a case includes the pressure–left pair as a rework loop, the discovered model can capture this behavior. Visible transitions are annotated with synthesized rule activities, while structural or silent transitions may remain unlabeled.

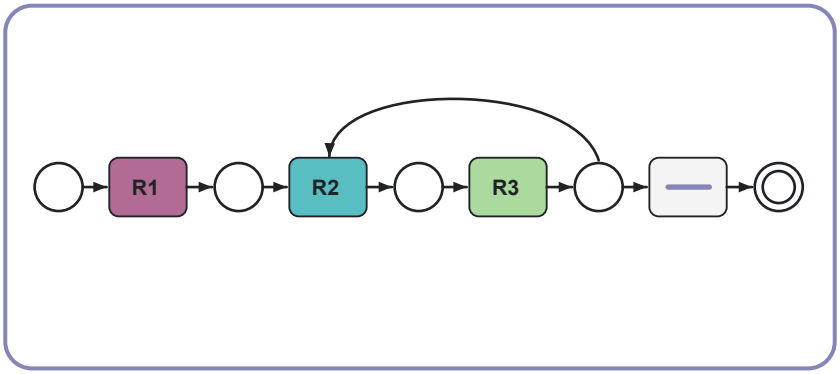


Figure 7: Process discovery produces a model over synthesized rule activities.

Table 1
Benchmark setup for the Sensor2EventLog comparison.

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

4. Evaluation

The evaluation compares our single-shot pipeline against Sensor2EventLog [7]. Sensor2EventLog uses a machine-teaching loop. A human teacher selects process states, adds event-rule features, reviews diagnostics, and refines rules. Our benchmark removes this teacher loop. Our benchmark uses zero teacher iterations, zero domain rules, zero rule additions, and zero rule edits. The evaluation results can be found here [1]

The benchmark follows a knowledge-agnostic protocol. The pipeline receives raw sensor columns and fixed domain-independent parameters and the pipeline does not receive activity names during discovery. The pipeline does not receive Sensor2EventLog thresholds, i.e., no $Q_{in} > \tau_Q$, no $T > 70$, no $\text{LIT101}_{\text{diff\_smooth}} < 0$, and no $\text{LIT101}_{\text{stability}} > 0.8$. Reference states are employed only after the discovery phase, where they are used to score the discovered rules according to the metrics defined in Sensor2EventLog. Table 1 summarizes the benchmark setup.

The evaluation uses the same dataset targets as Sensor2EventLog. The HACCP dataset contains 968 batches. The HACCP split uses 581 training batches and 387 test batches. The HACCP input contains 8 sensor columns and 6 reference states. The SWaT P1 run uses the normal-operation window from 2019-07-20T04:35:00+00:00 to 2019-07-20T06:50:00+00:00. The SWaT P1 formatting uses 4 equal contiguous pseudo-batches, i.e., 2 training pseudo-batches and 2 test pseudo-batches. The local SWaT workbook contains a `P1_STATE` column. This column is used only after discovery to compute anonymous state-level diagnostics.

The coverage metrics follow Sensor2EventLog. Let T_y be the timestamp set with reference state y. Let $r_t^{(m)} \in \{0, 1\}$ be the activation of rule R_m at timestamp t. Coverage measures the share of state y captured by rule R_m. Precision measures the share of activations of rule R_m that occur inside state y. Effectiveness is the geometric mean of coverage and precision.

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

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

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

Table 2
HACCP comparison with Sensor2EventLog rule metrics.

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
Best matching HACCP rule for each reference state 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

The reported state score uses the best matching discovered rule for each reference state. The reported mean score averages these best-rule scores over the reference states.

The state-explainability metric also follows Sensor2EventLog. State explainability for one state equals the maximum coverage achieved by any rule for that state. This value is computable only when reference-state labels are present. When timestamp accuracy is reported, each discovered rule is mapped to its majority reference state on the training split and this mapping is then applied to timestamp-level rule activations on the test split. Table 2 reports the aggregate HACCP comparison.

Our pipeline discovers 6 activities from 5,551 training segments. It evaluates on 3,699 test segments resulting in a mean test coverage of 0.749250, a mean test precision of 0.739081 and a mean test effectiveness of 0.700088.

After teacher refinement, Sensor2EventLog achieves superior HACCP rule metrics, with a mean coverage of 0.963000, mean precision of 0.970000, and mean effectiveness of 0.966309. This represents gains of 0.213750 in coverage, 0.230919 in precision, and 0.266222 in effectiveness relative to our method. Such differences are anticipated given the benchmark design: Sensor2EventLog leverages three teacher iterations and four domain rules, while our approach relies on neither. Table 3 reports the per-state HACCP scores.

The HACCP state scores show the main abstraction trade-off. The labels A01–A06 identify discovered interval predicates.

For example, a discovered rule of the form $z(\mathrm{end}(Q_{in})) \leq 2 \wedge z(\mathrm{sig}(T)) \in [-2.5, 2.62]$ describes segments where the final value of the Q_{in} sensor is not unusually high and the temperature-signature feature remains within a broad normal range.

The rules for HeatUp and Cool are highly precise. The HeatUp rule achieves 0.778352 coverage and 0.998650 precision, while the Cool rule reaches 0.778211 coverage and 0.998791 precision. In contrast, the Hold and Discharge states share rule A02 as their best rule. Although this shared rule provides high coverage for both states, it yields low precision for Discharge (0.998203 coverage and 0.195268 precision).

The induced HACCP log is almost fully labeled by the discovered rules, with a test segment rule applicability of 0.999730. This indicates that 99.9730% of test segments receive exactly one rule label, leaving only 0.000270 unlabeled.The held-out process model achieves a fitness of 0.999036 under the implemented conformance check, demonstrating that the induced HACCP traces are almost entirely accepted by the discovered model. Table 4 reports the SWaT values supported by the local artifacts.

The local SWaT workbook contains raw P1 sensor streams, controller/status columns, and

Table 4
SWaT P1 comparison with the values supported by the local artifacts.

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

a `P1_STATE` column. It does not contain the prepared `batch_id` column used by the public Sensor2EventLog implementation. Therefore, the run uses equal contiguous pseudo-batches for process mining formatting and `P1_STATE` only after discovery for scoring.

In the SWaT setting, our approach discovers three activities from 15 training segments and evaluates on 15 test segments. The post-hoc train-mapped timestamp accuracy is 0.805089. On the held-out SWaT state present in the test split, the best rule achieves 0.561759 coverage, 1.000000 precision, and 0.749506 effectiveness. Test applicability reaches 0.866667, while the held-out process-model fitness is 0.625000.

Sensor2EventLog reports SWaT accuracy after teacher refinement. In the HMM configuration, accuracy improves from a baseline of 0.580000 to a final value of 0.760000, while the K-means configuration increases from 0.700000 to 0.940000. Both configurations employ two teacher iterations and two domain rules. Additionally, Sensor2EventLog reports that all SWaT states exceed a state explainability of 0.780000 after refinement. In comparison, our SWaT diagnostic accuracy is 0.045089 higher than the reported HMM result and 0.134911 lower than the reported K-means result. These values should not be interpreted as a direct reproduction of the Sensor2EventLog SWaT setup, as our local run relies on pseudo-batches and anonymous workbook states.

The benchmark supports a conservative interpretation. The HACCP results demonstrate that a single-shot rule alphabet can recover meaningful state-related structure without relying on domain rules. At the same time, they show that the human-guided Sensor2EventLog loop achieves substantially higher rule metrics under the same label-based evaluation. The SWaT results further indicate that a single-shot rule alphabet can recover state-related structure in a second sensor system, although the K-means-based Sensor2EventLog configuration still attains higher accuracy after teacher refinement. Accordingly, this comparison should be understood as a fair benchmark of a knowledge-agnostic setting, rather than as a claim that single-shot discovery can replace teacher-guided refinement.

5. Discussion

Summary: The evaluation demonstrates the feasibility of a single-shot abstraction from raw sensor values. In the HACCP setting, the approach discovers six sensor-derived activities without the use of domain rules. These activities label 99.9730% of the HACCP test segments. The best-rule HACCP scores reach 0.749250 mean coverage, 0.739081 mean precision, and 0.700088 mean effectiveness. The held-out HACCP process-model fitness is 0.999036. These values demonstrate that the pipeline recovers substantial state and control-flow structure before any human refinement.

The benchmark also confirms the value of domain knowledge. Sensor2EventLog reaches 0.963000 mean coverage, 0.970000 mean precision, and 0.966309 mean effectiveness on HACCP. This result is higher by 0.213750 coverage points, 0.230919 precision points, and 0.266222 effectiveness points. The difference follows from the setting because Sensor2EventLog uses 3 teacher iterations and 4 domain rules.

The SWaT run extends the evidence beyond HACCP. It discovers 3 activities without domain rules and labels 86.6667% of held-out test segments. Using the workbook's `P1_STATE` column

only after discovery, the run reaches 0.805089 train-mapped timestamp accuracy, 0.561759 coverage, 1.000000 precision, and 0.749506 effectiveness on the held-out state present in the test split. This diagnostic accuracy exceeds the 0.760000 reported for the Sensor2EventLog HMM configuration but falls short of the 0.940000 achieved by the K-means configuration. Accordingly, our result should be interpreted as a knowledge-agnostic baseline rather than as a substitute for teacher-guided eventization.

Limitations The evaluation uses reference states only after the discovery phase. This preserves the knowledge-agnostic setting but implies that semantic interpretation remains post hoc. The discovered activity identifiers correspond to interval predicates rather than human-interpretable activity names. For instance, the HACCP results indicate that Hold and Discharge share the same best rule. While this is informative as an abstraction diagnostic, it also highlights that formally valid rules may merge states that a domain expert would prefer to keep distinct.

Finally, process-model fitness is evaluated on the event log induced by the synthesized rules. High fitness therefore indicates that the discovered control-flow model accepts the induced rule traces, rather than that the abstraction captures all physically meaningful state distinctions. Accordingly, rule metrics, model fitness, and domain interpretation should be considered jointly.

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] 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.

[6] R. Seiger, M. Franceschetti, B. Weber, An interactive method for detection of process activity executions from iot data, Future Internet 15 (2023) 77.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] I. Chevyrev, A. Kormilitzin, A Primer on the Signature Method in Machine Learn-

ing, 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] J. Morrill, A. Fermanian, P. Kidger, T. Lyons, A generalised signature method for multivariate time series feature extraction, arXiv preprint arXiv:2006.00873 (2020).

[20] A. Fermanian, Embedding and learning with signatures, Computational Statistics & Data Analysis 157 (2021) 107148.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.