Know Your Streams: On the Conceptualization, Characterization, and Generation of Intentional Event Streams

Andrea Maldonado*[1,4,5], Christian Imenkamp*[2], Hendrik Reiter[3], Thomas Seidl[4,5], Wilhelm Hasselbring[3], Martin Werner[2], and Agnes Koschmider[2]

[1] School of Engineering and Design, Technical University of Munich, Germany
{andrea.maldonado martin.werner}@tum.de
[2] University of Bayreuth, Bayreuth, Germany
{christian.imenkamp,agnes.koschmider}@uni-bayreuth.de
[3] Christian-Albrechts-University Kiel, Kiel, Germany
{hendrik.reiter,hasselbring}@email.uni-kiel.de
[4] Database Systems and Data Mining, Ludwig Maximilian University of Munich, Germany seidl@dbs.ifi.lmu.de
[5] Munich Center for Machine Learning, Munich, Germany

Abstract. The shift toward IoT-enabled, sensor-driven systems has transformed how operational data is generated, favoring continuous, real-time event streams (ES) over static event logs. This evolution presents new challenges for Streaming Process Mining (SPM), which must cope with out-of-order events, concurrent activities, incomplete cases, and concept drifts. Yet, the evaluation of SPM algorithms remains rooted in outdated practices, relying on static logs or artificially *streamified* data that fail to reflect the complexities of real-world streams. To address this gap, we first perform a comprehensive review of data stream literature to identify stream characteristics currently not reflected in the SPM community. Next, we use this information to extend the conceptual foundation for ES. Finally, we propose Stream of Intent, a prototype generator to produce ES with specific features. Our evaluation shows excellence in producing reproducible, intentional ES for targeted benchmarking and adaptive algorithm development in SPM.

Keywords: Algorithm Evaluation · Event Stream Features · Process Discovery · Conformance Checking · Benchmarking

1 Introduction

The rise of the Internet of Things (IoT), Industry 4.0, and distributed sensor systems has fundamentally changed how operational data is generated. Increasingly, organizations must deal not only with static collections of events, but also with Event Streams (ES) i.e., real-time sequences of events emitted by interconnected components and systems. Consider a manufacturing setting in which

* Equal contribution

multiple machines, equipped with sensors, continuously send status updates to a central system. Here, events arrive asynchronously, possibly out of order, and with limited knowledge about case completion. Moreover, such environments are subject to concept drifts, for instance, when a new machine is introduced or a production line is reconfigured, the underlying process behavior shifts, potentially invalidating previously learned process models. These conditions are typical in industrial environments and pose significant challenges for the practical application of process mining (PM).

Traditional PM assumes that event logs are complete, static, chronologically ordered by case, and collected retrospectively. These assumptions work well for postmortem analysis but do not capture the dynamic, real-time nature of ES environments. In contrast, Streaming Process Mining (SPM) aims to support real-time analysis by processing events on the fly. Although traditional PM can analyze and explain the reasons behind a factory's production failure after it has occurred, SPM aims to spot issues early enough to prevent them. However, existing approaches often rely on *streamified* (i.e., incremental replay of static) event logs for evaluation [5,4,2,7,6] or use comparable synthetic event logs generated by the CPN Tools [1] rather than real-time event streams. These approximations lack core properties of realistic ES, such as temporal disorder [3], indefinite case lifespans, and concurrent activities [5], which undermine their utility for comprehensive development and testing. Furthermore, [13] generates Out-of-Order streams by assigning some delayed ingestion times to an existing log.

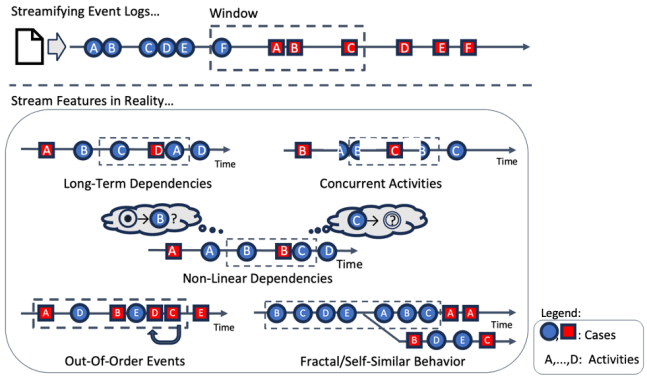


Fig. 1: Know Your Streams is the first framework to breach ES research with the complexities of real-world real-time processes.

To address this gap, we extend the formalization of ES that explicitly captures the distinguishing characteristics of real-world streaming data (Figure 1). Our structured scheme consolidates previously fragmented definitions of what constitutes an ES and identifies key dimensions that must be considered for

[1] http://cpntools.org

evaluation in a streaming context. We conducted this based on a comprehensive literature review of the general data stream community. Building on this basis, we propose a conceptual framework and a prototype generator that produces intentional ES.

In summary, our contributions are as follows: (1) We identify characteristics of ES from the general data stream community and compare them to current SPM approaches. (2) We expand the ES definition and derive a conceptual framework for ES. (3) We propose a prototype to generate ES with intentional characteristics.

2 The Need for Realistic Event Streams

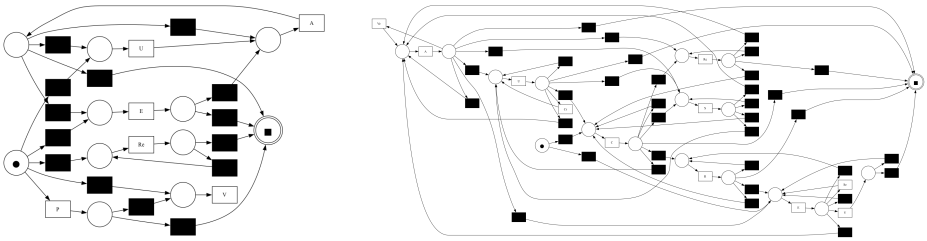
To motivate the need for realistic ES characteristics, we compare the process discovery results from a stream-based event log, a common approximation, with those from the same ES, which inherently includes the realistic characteristics identified in our work. Figure 2 shows the results. Both (a) and (b) required extensive manual preprocessing, a classical step in process mining, to discover the process model (e.g., restoring directly following relations, filtering out fractal behavior). This is because existing algorithms are not equipped to handle these characteristics. Consequently, such a time-consuming preprocessing is a prerequisite that is often not feasible in real-time settings. Instead of relying on extensive filtering and cleaning to convert streams into an "offline" format, we propose integrating realistic stream characteristics directly into algorithms to leverage them for valuable analysis.

In this experiment, we use (a) as to establish a baseline for the expected process model, which was created from a preprocessed version of the stream in (b). In contrast, (b) presents the result of the streaming heuristics miner [7]. Notably, even the streaming heuristics miner required additional preprocessing to discover the process model in (b), as its design does not inherently handle the structures present in a realistic stream. A comparison of the two results reveals a significant increase in silent transitions in ((b) = 36) compared to ((a) = 15), which results in a greater number of required connections. Furthermore, our analysis shows that the streaming heuristics miner tends to discover more loops when dealing with realistic streams, which is likely a consequence of their more complex and unpredictable nature.

The purpose of this experiment is not to diminish the value of established methods, but to highlight the opportunity to harness the benefits of true real-time analysis through targeted modeling of realistic stream characteristics.

3 Related Work

Data streams differ significantly from static datasets [1]. In particular, characteristics of real-time environments (i.e., continuous data arrival, limited resources, latency constraints, and unpredictable future data instances) pose unique challenges [11,9,14]. These challenges are inherent in the application side of the



(a) Heuristics miner using a stream-based event log.

(b) Streaming heuristics miner using an ES containing realistic ES characteristics.

Fig. 2: Comparing discovered process models (with dependency threshold = 0.85) including vs. omitting realistic ES characteristics from the same event source.

streaming environment. Most SPM algorithms analyze the data and control-flow of the stream. Given that, we will focus on literature describing characteristics of streams that make it challenging to detect patterns in streaming data.

Streams include complex patterns and dependencies. For instance, temporal dependencies illustrate the influence of past events on subsequent events [22,12]. Furthermore, long-term dependencies pose additional challenges, requiring algorithms to maintain and utilize historical information [22]. Non-linear dependencies further increase complexity. Future events often are the result of past non-linear rules rather than simple sequential relationships [10,1].

Moreover, realistic ES frequently experience out-of-order events. This disrupted the timely assumptions often required in traditional methods [22,11]. Additionally, fractal behavior, which is marked by repeating patterns across different scales, breaks common assumptions about data being independent and uniformly distributed. This makes tasks such as anomaly detection and forecasting more challenging [24,23].

However, SPM designs and evaluations are often based on artificially *streamified* event logs and adopt the foundations of classical process mining, ignoring the characteristics of real-time environments [8,19,20].

4 Event Streams

In this section, we consolidate the definition of event stream (ES), particularly focusing on extending the static atomic event notation to handle activities with duration to better capture the complexities of real-world processes and analyze ES characteristics, such as concurrency.

4.1 Event Stream Formalization

Typically, in PM, an *event* $e \in \mathcal{E} : c \times a \times t$ is an atomic tuple in event data. Each event contains at least three attributes: A *case id* $c \in C$ from the universe

of case identifiers $\mathcal{C}$, an *activity* $a \in \mathcal{A}$ from the universe of activities, $\mathcal{A}$ and a *timestamp* $t \in \mathbb{N}$. Further, the projection function π_x maps an event to its attribute x. Hence, $\pi_c((c_0, a_0, t_0)) = c_0$.

This definition is sufficient to model instantaneous events and many real-world scenarios. These activities naturally require both a start and an end timestamp. Consider a smart factory where machines autonomously perform tasks such as assembling parts, performing diagnostics, or transporting materials. These activities span time intervals: for example, assembling a component may begin at 14:03:10 and complete at 14:04:05. If such activities are represented using a single timestamp, critical temporal information is lost. This simplification limits the kinds of questions to be answered, like the average duration of an activity, the presence of waiting times between activities, or the number of concurrently running cases. These limitations clearly show that an ES model relying on only one timestamp per activity is inadequate for this category of applications.

To address this limitation, we introduce a formalism where each activity is represented by a pair of events: A *start event* marking the beginning of the activity and an *end event* marking its completion. Let $\mathcal{E}_s$ and $\mathcal{E}_e$ denote the sets of start and end events. We define $e_s = (c, a, t, \text{"}start\text{"}) \in \mathcal{E}_s$ and $e_e = (c, a, t, \text{"}end\text{"}) \in \mathcal{E}_e$. We model the *interval-based* ES as a function: $S^* : \mathbb{N} \to \mathcal{E}_s \cup \mathcal{E}_e$. An event stream is called temporally ordered if the following holds: $\forall i, j \in \mathbb{N} : i < j \implies \pi_t(S^*(i)) \leq \pi_t(S^*(j))$.

To map start events to their respective end events we introduce the function $\mu : \mathcal{E}_s \to \mathcal{E}_e$. This function has the following constraints for all $e_s \in \mathcal{E}_s$ and $e_e = \mu(e_s) \in \mathcal{E}_e$: $\pi_c(e_s) = \pi_c(e_e)$, $\pi_a(e_s) = \pi_a(e_e)$ and $\pi_t(e_s) \leq \pi_t(e_e)$.

Building on this formalism, we can now define properties within ES, such as the concurrency of events at a given timestamp t as $\text{concurrency}(t^*) = |\{e_s \in \mathcal{E}_s \mid \pi_t(e_s) \leq t^* \leq \pi_t(\mu(e_s))\}|$.

Another crucial aspect in real-world ES is the absence of explicit case identifiers [15]. Events often arrive without a clear reference to a process instance. Therefore, online correlation is an additional key challenge for SPM and stream generation.

4.2 Event Stream Characteristics

To further build on the holistic definition of an ES, we discuss the characteristics derived from the literature (Section 3) and their translation from data streams into ES. For that, we use the extended ES formalization shown in Section 4.1. In the following section, we propose formalisms for temporal, nonlinear, and long-term dependencies, out-of-order events, and fractal process structures.

Event Observation and Temporal Reality. In real-time environments, it is essential to clearly distinguish between the time an event occurs in the real-world process (the **event timestamp**) and the time it is captured and received by the analysis system (the **arrival timestamp**).

We therefore extend the event tuple to $e = (c, a, t, \text{type}, t_{arrival})$, where $t \in \mathbb{N}$ is the event timestamp and $t_{arrival} \in \mathbb{N}$ is the arrival timestamp. The ES,

$S^* : \mathbb{N} \to \mathcal{E}_s \cup \mathcal{E}_e$, is a sequence ordered by arrival time, reflecting the order of observation.

$$\forall i, j \in \mathbb{N} : i < j \implies \pi_{t_{arrival}}(S^*(i)) \leq \pi_{t_{arrival}}(S^*(j))$$

This ordering allows us to formally define events that arrive out of their natural temporal sequence.

Out-of-Order Events An ES S^* is considered to contain **out-of-order events** if the sequence of event timestamps t is not monotonically non-decreasing concerning the arrival sequence. Formally, an out-of-order event is any event $e_j = S^*(j)$ in the position j for which a preceding event in the stream, $e_i = S^*(i)$ with a position i, where $i < j$, has a later event timestamp. Let X denote the set of relevant attributes used to compare events. An event $e_j = S^*(j)$ is Out-Of-Order if there exists a preceding event $e_i = S^*(i)$ with $i < j$ such that $\pi_t(e_j) < \pi_t(e_i)$ for the same projection on X. The displacement $\pi_t(e_i) - \pi_t(e_j)$ then quantifies the degree of disorder. To avoid ambiguity, we explicitly require that the ordering refers to the attribute t_{arrival}.

$$\exists i, j \in \mathbb{N} \left(i < j \ \wedge \ \forall x \in X : \ \pi_t(S^*(j, x)) < \pi_t(S^*(i, x)) \right)$$

The magnitude of this temporal displacement, $\pi_t(S^*(i)) - \pi_t(S^*(j))$, can be used as a measure of the stream's disorder.

Dependencies: Temporal, Long-Term, and Non-Linear The evolution of a process case is governed by dependencies between its events. To model this, we define the **history** of a case $c \in \mathcal{C}$ at time t, denoted $H(c, t)$, as the ordered sequence of all events belonging to that case up to time t.

$$H(c, t) = \langle e_1, e_2, \ldots, e_k \mid \pi_c(e_i) = c \wedge \pi_t(e_i) \leq t \rangle$$

The generation of subsequent events in the case is determined by a conceptual **transition function**, δ, which maps the case history to a new activity.

$$a_{k+1} = \delta(H(c, t))$$

This function δ encapsulates the underlying process logic:

- **Temporal Dependencies**: These are inherent, as δ operates on the event history. The time elapsed between events, $\pi_t(e_{k+1}) - \pi_t(e_k)$, is a direct consequence of the process modeled by δ.
- **Long-Term Dependencies**: These occur when the function δ is sensitive to events early in the history (e_i for $i \ll k$), meaning distant past events can influence current and future activities.
- **Non-Linear Dependencies**: The complexity of the process logic is embodied by δ. If δ represents simple sequential logic, the dependencies are linear. However, if δ models complex branching, parallelism, or context-aware decisions, it gives rise to non-linear dependencies.

δ can be instantiated as a simple mapping where the next activity depends only on the last activity, e.g., $\delta(\langle A \rangle) = B$, or as a context-aware rule that considers the entire history, e.g., $\delta(\langle A, C \rangle) = D$ if an earlier event C occurred.

Hierarchical Structures and Self-Similarity Processes in the real world are often not isolated, but can be nested or can trigger other processes. We can model this with a **case containment relation**, denoted by the symbol $\prec$. The expression $c_j \prec c_i$ signifies that case c_j is a sub-process or a nested component of case c_i. This relation imposes a hierarchical, directed acyclic graph structure on the set of all cases $\mathcal{C}$.

A process may exhibit **fractal or self-similar behavior** if the underlying process logic repeats across different levels of this hierarchy. Let $\mathcal{P}(c)$ be the abstract process model for a case c (represented by its transition function δ).

Let $e_k \in c_i$ be the trigger event for case c_j. This establishes a temporal correlation in which the existence of c_j depends on e_k. The first subprocess event must occur after or at the same time as the triggering event: $\min(\{\pi_t(e)|\pi_c(e) = c_j\}) \geq \pi_t(e_k)$.

Self-similarity exists if, for a pair of cases $c_j \prec c_i$, their process models are structurally related by a transformation $\mathcal{T}$:

$$\mathcal{P}(c_j) \approx \mathcal{T}(\mathcal{P}(c_i))$$

The transformation $\mathcal{T}$ could represent various relationships, such as a simplification of the parent process, a scaling of its temporal properties, or an abstraction of its activities (as seen in Figure 1).

F may be realized as a scaling of temporal properties (e.g., multiplying all activity durations by a factor of two) or as a structural simplification (e.g., collapsing a parallel branch into a sequential pattern).

5 Stream of Intent: Our Prototype Generator

ES are fundamental for analyzing and understanding dynamic systems. Generating synthetic ES with specific, controllable characteristics is crucial for robust system testing, anomaly detection, and algorithm validation, especially when real-world data is scarce or sensitive. Our prototype, Stream of Intent, addresses this need by providing a novel approach to create synthetic ES with intentionally predefined features. It leverages the underlying simulation capabilities of the Distributed Event Factory (DEF) [21] and the feature-driven optimization framework of Generating Event Data with Intentional Features (GEDI) [18]. To define the features required by GEDI we utilize the characteristics presented in Section 4.1. We define a feature as a quantifiable characteristic with a numerical value.

This section details the architecture and method of Stream of Intent, explaining how it iteratively optimizes generator parameters to produce streams closely adhering to specified characteristics.

5.1 Our Approach: Stream of Intent Pipeline

The complete pipeline of Stream of Intent is conceptually illustrated in Fig. 3. The generation process begins by defining *static parameters* (e.g., window size,

and implicit/explicit time order) and *dynamic configuration parameters* that are subject to optimization. The core idea is an iterative optimization loop that ensures the generated streams adhere closely to predefined *target feature characteristics*. The pipeline consists of the following steps:

1. **Specify Stream Features and Targets:** The user defines specific goals for one or more stream features (e.g., target event frequency, specific concurrency patterns). These target values drive the optimization process, similarly to how GEDI defines a target feature vector g_k.
2. **Generate Process Model:** An initial process tree is generated using stochastic methods, such as those described in [16], along with an initial set of PTLG (Process Tree-based Log Generator) configuration parameters. This process tree serves as a high-level model of the desired process behavior.
3. **Process Tree to Markov Chain Conversion:** The generated process tree is converted into a Markov chain, translating its structural and behavioral properties into probabilistic state transitions for DEF's event simulation. The specifics of this conversion are elaborated in Section 5.2.
4. **Generate Stream (DEF Simulation):** The Markov chain derived from the process tree is then used by DEF to simulate a intentional ES. During this simulation, Stream of Intent also accounts for concurrent events, event lifecycles, and event durations, which are critical for realistic stream generation (see Section 4.1). DEF's underlying Markov chain model, as described in [21], allows for the bottom-up generation of events.
5. **Feature Extraction and Evaluation:** The simulated ES is intercepted and batched using a tumbling window approach (ensuring no overlap between windows). Predefined stream features are then extracted from each window and evaluated against the initial target feature values.
6. **Hyperparameter Optimization (Bayesian Optimization):** A Bayesian optimization procedure iteratively adjusts the generator parameters based on the evaluation results from the previous step. This optimization loop continues through the generation and evaluation stages until the desired feature specifications are met or a maximum number of optimization attempts is reached, mirroring GEDI's formal optimization problem $G(g_k) = \min_{\lambda \in \Lambda} d(f_e(A_\lambda), g_k)$.

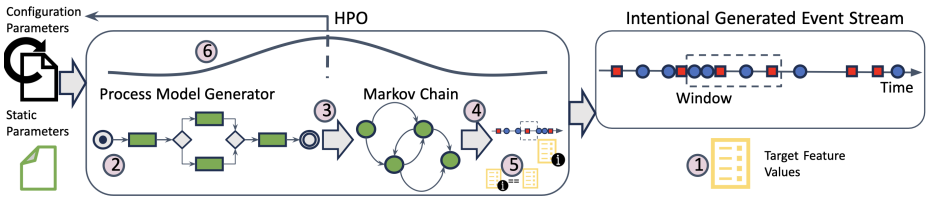


Fig. 3: Stream of Intent generates ES with intentional features, optimizing configuration parameters.

Upon conclusion of the optimization, Stream of Intent produces a static definition file compatible with DEF. This definition can be used to replay the stream with the optimized features. Furthermore, DEF's capability to combine multiple definitions allows for the simulation of concept drift in streams, enabling transitions (gradual or sudden) from one set of features to another.

5.2 From Process Tree to Markov Chain

The Stream of Intent pipeline starts with process trees, but its core simulation engine, DEF, operates on probabilistic Markov chain models. Thus, a critical step is converting the high-level process tree into a precise Markov chain representation. This transformation translates the tree's structural and behavioral properties into granular, probabilistic state transitions directly usable by DEF.

Our method adapts the approach by [17] for converting an observed event log into a first-order Markov chain. To bridge the gap, Stream of Intent first generates a representative sample set of traces from the process tree using its inherent log generation capabilities (e.g., as provided by the PTLG framework [16]). From this generated sample log L, a mapping structure C records activity transition frequencies. For each trace, transitions (from an implicit start state s_{start}, between consecutive activities (a_i, a_{i+1}), and to an implicit end state s_{end} are counted. These accumulated frequencies in C are then normalized to define the complete transition matrix of the Markov chain. This precisely constructed Markov chain is then supplied to DEF, enabling the simulation of ES whose probabilistic behavior accurately reflects the control-flow specified by the original process tree.

5.3 Experimental Results

Similarly to [18] we investigate two questions with our experiments: (E1) How well does Stream of Intent produce ES with specific feature values? (feasibility) (E2) How well do the commonly used event logs represent the feature space covered by Stream of Intent? (representativeness) Last, we also assess the utility of our generated ES and identify opportunities for further online streaming process mining algorithms developments. Our generated data[2] , implementation and evaluation results[3] are publicly available.

We first evaluate the **feasibility** of ES generation by analyzing how closely our streams reach predefined target values (i.e., [0, 0.1, 0.5, 0.7, 1.0]). We conducted 250 experiments covering all pairwise combinations of five features with five target values (i.e., $\binom{5}{2} = 10 \rightarrow 5 \cdot 5 = 25 \rightarrow 10 \times 25 = 250$). Results are shown in Figure 4. Yellow indicates minimal deviation from the target, while dark purple indicates maximal. Warmer colors denote easier-to-match features; cooler colors, harder ones. On the diagonal, we show feasible value ranges for individual features. The upper triangle demonstrates the average deviation for the

[2] https://doi.org/10.5281/zenodo.16685512
[3] https://github.com/andreamalhera/gedi_streams/tree/icpm25

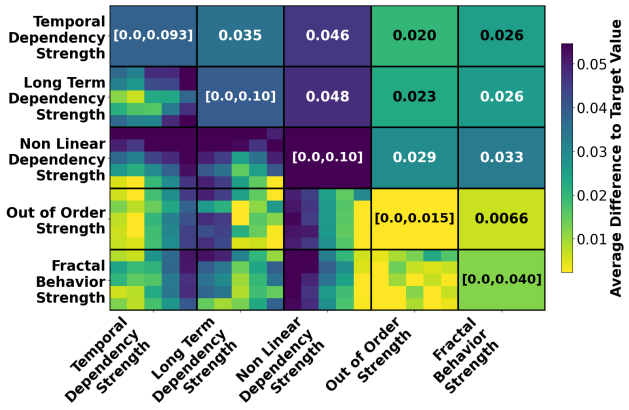


Fig. 4: In the diagonal, we present ranges with low distance results; above the diagonal, average distances; and below it, distances comparing targets to generated event stream feature values for each combination of features.

pair of features. The lower triangle displays a grid 5×5 of target combinations. For instance, *Non Linear Dependency* combined with *Temporal Dependency* or *Long Term Dependency* is harder to match when trying to achieve higher target values for both. Similarly, *Fractal Behavior* requires *Non-Linear Dependencies*, given the high deviation when the target for *Non Linear Dependencies* is set to zero. These results reveal the feasible regions for each feature and their pairwise interactions. This highlights constraints and dependencies that govern feasible targets in the ES generation space.

To answer our second evaluation question on **representativeness**, we compare the 250 ES produced for (E1) with event logs commonly used to evaluate SPM algorithms. For that, we utilize the principal components (i.e., linear dimensionality reduction technique) to compare the achievable feature space of Stream of Intent with the aforementioned event logs. The results of this experiment can be found in Figure 5, where the orange hull represents the feature space from the ES and blue from the event log. The first observation is that no event log archives PC2 values higher than two. Some features are simply absent.

Thus, answering E2, investigated event logs do not contain most characteristics of realistic ES and thus undermine their applicability to evaluate SPM tasks.

To further assess the **utility** of Stream of Intent, we conducted numerous experiments using established SPM algorithms [5,4,2,7,6]. from section 1. Although these algorithms are valid and useful in multiple scenarios, their assumptions do not align with the characteristics defined in section 4.2. Given that, we encourage the development of novel streaming discovery approaches that can natively handle the complexities of realistic, concurrent ES.

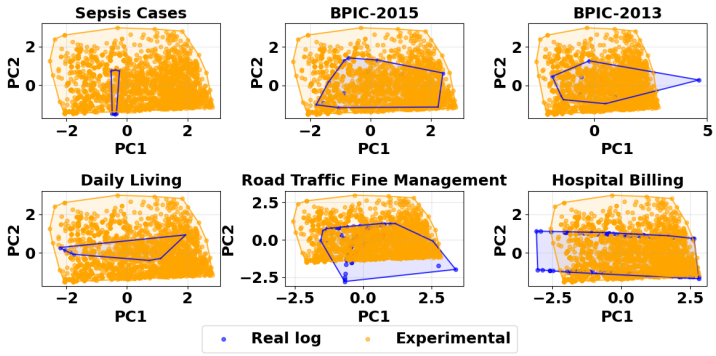


Fig. 5: Principal-component map contrasting generated ES (orange) with benchmark event logs (blue).

6 Discussion and Conclusion

In this paper, we address the gap between the SPM and data stream communities by identifying five key features from the literature that characterize realistic streams. As shown in section 2, including these features is crucial for real-time analysis, which existing methods often hinder through manual preprocessing. We expand the classical ES definition, propose a prototype generator, and show that Stream of Intent produces streams closely aligned with target features.

Threads To Validity: As no real-world streaming ES are publicly available, we could not compare our generated streams directly to such data. The selected features are reduced but cover a broad perspective in the literature [3,5]. We also note a selection bias in the evaluated algorithms, as their assumptions differ and affect applicability.

In future work: We intend to further work on the intentional generation of event streams with controllable and explainable features selection.

We would like to conclude by emphasizing that we do not intend to diminish the value of established methods for SPM, but rather to encourage the community to explore new directions by leveraging more realistic event streams, thereby enhancing algorithm robustness and applicability to real-world scenarios.

References

1. Aggarwal, C.C., Yu, P.S., Han, J., Wang, J.: A framework for clustering evolving data streams. pp. 81–92. Elsevier
2. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowledge and Information Systems **59**(2), 251–284 (May 2019)
3. Awad, A., Weidlich, M., Sakr, S.: Process mining over unordered event streams. pp. 81–88 (2020)

4. Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online Discovery of Declarative Process Models from Event Streams. IEEE Transactions on Services Computing **8**(6), 833–846 (Nov 2015), conference Name: IEEE Transactions on Services Computing
5. Burattin, A., López, H.A., Starklit, L.: Uncovering Change: A Streaming Approach for Declarative Processes. pp. 158–170. Springer Nature Switzerland, Cham (2023)
6. Burattin, A., Sperduti, A., van der Aalst, W.M.P.: Control-flow discovery from event streams. pp. 2420–2427 (Jul 2014), iSSN: 1941-0026
7. Burattin, A., Sperduti, A., Aalst, W.M.P.v.d.: Heuristics Miners for Streaming Event Data (Dec 2012), arXiv:1212.6383
8. Burattin, A., Sperduti, A., van der Aalst, W.M.: Heuristics miners for streaming event data. arXiv preprint arXiv:1212.6383 (2012)
9. Chakrabarti, A., Cormode, G., Mcgregor, A.: A near-optimal algorithm for estimating the entropy of a stream **6**(3), 51:1–51:21
10. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey **41**(3), 15:1–15:58
11. Cormode, G.: References for data stream algorithms
12. Cuomo, J., Homayouni, H., Ray, I., Ghosh, S.: Detecting temporal dependencies in data
13. Grulich, e.a.: Generating reproducible out-of-order data streams. In: Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems. p. 256–257. DEBS '19, Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3328905.3332511
14. Haug, J., Tramountani, E., Kasneci, G.: Standardized evaluation of machine learning methods for evolving data streams
15. Helal, I.M., Awad, A.: Online correlation for unlabeled process events: A flexible cep-based approach. Information Systems **108**, 102031 (2022), https://www.sciencedirect.com/science/article/pii/S0306437922000333
16. Jouck, T., Depaire, B.: Ptandloggenerator: A generator for artificial event data (2016)
17. Kalenkova, A., Mitchell, L., Roughan, M.: Performance analysis: Discovering semi-markov models from event logs. IEEE Access **13**, 38035–38053 (2025)
18. Maldonado, A., Frey, C.M.M., Tavares, G.M., Rehwald, N., Seidl, T.: Gedi: Generating event data with intentional features for benchmarking process mining. pp. 221–237. Springer Nature Switzerland, Cham (2024)
19. Nagy, Z., Werner-Stark, A.: A multi-perspective online conformance checking technique. pp. 172–176 (03 2020)
20. Raun, K., Nielsen, M., Burattin, A., Awad, A.: C-3pa: Streaming conformance, confidence and completeness in prefix-alignments. pp. 437–453. Springer Nature Switzerland, Cham (2023)
21. Reiter, H., Imenkamp, C., Koschmider, A., Hasselbring, W.: Distributed Event Factory: A Tool for Generating Event Streams on Distributed Data Sources
22. Song, Y., Lu, J., Lu, H., Zhang, G.: Learning data streams with changing distributions and temporal dependency **34**(8), 3952–3965
23. de Sousa, E.P.M., Traina, A.J.M., Traina, C., Faloutsos, C.: Measuring evolving data streams' behavior through their intrinsic dimension **25**(1), 33–60
24. Zhang, M., Huang, S., Chen, S., Wei, C.T., Hu, J., Ji, K., Khan, I., Zhang, M.: Research on soft error detection method for self-similar data streams based on structure mapping