A Framework for Steady-State Detection in Process Mining

Alexander Kraus[a,b], Keyvan Amiri Elyasi[a], Adrian Rebmann[b], Sherri Hadian[b], Han van der Aa[c]

[a] *University of Mannheim, B6, 26, Mannheim, 68159, Germany*
[b] *SAP Signavio, SAP SE, Dietmar-Hopp-Allee 16, Walldorf, 69190, Germany*
[c] *University of Vienna, Währinger Str. 29, Vienna, 1090, Austria*

Abstract

Steady-state detection (SSD) is a crucial task in the analysis of complex and dynamic systems, as it enables the reliable assessment of system behavior by distinguishing between stable and unstable states. SSD techniques have been extensively studied and applied in various domains, including signal processing and industrial systems. However, their application within the information systems domain, particularly in process mining, has received little attention, even though business processes themselves can be regarded as complex socio-technical systems. In particular, event logs that capture the execution of business processes often contain data from both steady and non-steady states. Mixing up these states can significantly affect the accuracy and reliability of insights from common process mining tasks, such as process performance analysis and process discovery. To address this problem, we propose a dedicated SSD framework for process mining and demonstrate how differentiating between distinct process states can enhance the accuracy and reliability of process mining insights. The SSD framework takes an event log as input and identifies the existing steady and non-steady states along with their corresponding time periods. We evaluate the framework through two experiments: one assessing accuracy using simulated event logs and another demonstrating its impact on three key process mining tasks: process performance analysis, process discovery, and remaining time prediction.

Keywords: Process mining, Business process, Steady-state detection

1. Introduction

Business processes are often supported by information systems that record execution data in event logs, which are then used in process mining to extract data-driven insights [1]. However, these event logs often capture business processes executed in both steady and non-steady states. A *steady state* represents a period when a process operates in a stable and predictable manner, showing consistent behavior over time [2]. In contrast, a *non-steady state* reflects periods of change or instability, where the process experiences fluctuations and irregularities in response to varying conditions in its operating environment. These non-steady states can arise from factors such as increased case arrivals during peak seasons or reduced resource availability during holidays, causing the process to deviate from its usual operations and performance levels.

The distinction between steady and non-steady states of processes is crucial for various process mining tasks. Failing to distinguish between such states can, for instance, distort process performance insights, impact the process discovery insights, or hurt the accuracy of predictive process monitoring models. Recognizing the impact that fluctuations have on the behavior of complex and dynamic systems, the task of *Steady-State Detection* (SSD) aims to identify periods when a system operates in a steady state (or when it does not). Various techniques for this task have already been developed and tested in different application contexts, such as industrial systems [3] and signal processing [4]. However, their application in process mining has been largely overlooked, even though business processes are dynamic socio-technical systems [5] where SSD can significantly enhance the accuracy of process mining insights.

Therefore, in this paper, we propose a SSD framework for process mining and demonstrate how differentiating between distinct process states can enhance the accuracy and reliability of insights across several common process mining tasks. Our framework takes an event log as input and identifies the existing steady and non-steady states and when they occur. The framework is operationalized in three main steps. First, it extracts time series from event logs that capture the evolution of relevant process characteristics. Then, it detects steady and non-steady states using existing SSD techniques that work with time series. Finally, it groups the detected steady states based on the corresponding process behavior to distinguish between different steady-state groups, i.e., steady states with different characteristics. The

accuracy of our framework is evaluated through two experiments: one assessing its performance in a controlled environment using simulated event logs, and another demonstrating its impact on several downstream process mining tasks, specifically process performance analysis, process discovery, and remaining time prediction. Our findings confirm that the proposed framework effectively enables the application of SSD in process mining and highlight its potential to deliver more accurate and reliable insights into organizational operations.

This paper presents an extended and revised version of our earlier work [6], in which we introduced the first version of our framework. The current paper extends that work in three main directions. First, we enhance the framework's capabilities by introducing an additional analysis step (Step 3). This step enables the framework not only to distinguish between steady and non-steady states but also to determine whether the identified steady states belong to the same or to different steady-state groups, thereby supporting a more detailed analysis of existing steady states in downstream process mining tasks. Second, we substantially broaden the evaluation. We conduct a sensitivity analysis of the hyperparameter used in Step 1, incorporate evaluation of additional SSD techniques in Step 2, evaluate the accuracy of the newly added Step 3, and provide a misclassification analysis to highlight key challenges in detecting steady states in business processes. Finally, we explore further applications of the framework in process mining. In addition to the original task of remaining time prediction, we assess the framework's impact on process performance analysis and process discovery. We also extend the original remaining time prediction experiment by evaluating the use of the framework as a bucketing strategy and comparing it with established bucketing techniques in predictive process monitoring.

The remainder of this paper is organized as follows. Section 2 provides background and illustrates the importance of SSD in process mining. Section 3 presents the proposed SSD framework for business processes. Section 4 reports the evaluation experiments conducted to assess the accuracy of the framework, and Section 5 demonstrates its usefulness in downstream process mining tasks. Section 6 reflects on the strengths and limitations of our framework. Finally, Section 7 discusses the relationship between SSD and related problems in process mining, and Section 8 concludes the paper and outlines directions for future work.

2. Background and Problem Illustration

In this section, we present the preliminaries of process mining, introduce the concept of steady states, and explain its relevance to process mining.

Preliminaries. *Process mining* is a discipline at the intersection of data science and process science that focuses on the analysis and improvement of *business processes* by systematically leveraging *event logs* recorded by information systems [7].

Business process and its perspectives. A *business process* is a set of activities and constraints between them that are performed within an organizational and technical environment to realize a business goals [8]. A business process can be analyzed from several *perspectives* [9]. The *control-flow* perspective considers the sequencing and dependencies of activities. The *resource* perspective examines the involvement of individuals, systems, or departments. The *time* perspective focuses on temporal aspects such as bottlenecks, service levels, and case duration. The *data* perspective addresses additional event information, such as costs or product details.

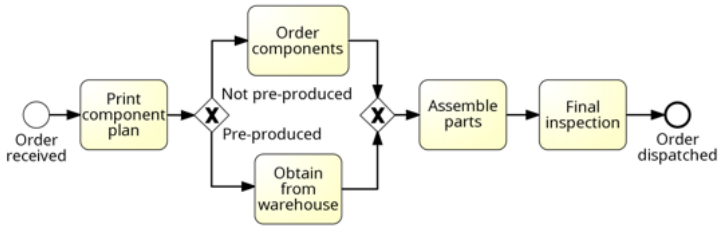


Figure 1: Model of a business process.

Business processes are commonly represented through process models defined in a specific modeling language. A widely used standard for this purpose is the Business Process Model and Notation (BPMN) [10]. Figure 1 shows the control-flow of a business process model expressed in BPMN. The model has a start event labeled "Order received", five activities, and an end event marked "Order dispatched". The activities are carried out in sequence, with a single decision point that allows for a choice between "Order components" and "Obtain from warehouse" after the first process activity.

Event log. An *event log* L is as a collection of events recorded by an information system. Each event $e \in L$ is represented as a tuple $e = (\text{caseID}, \text{activity}, \text{timestamp})$, where *caseID* identifies the case, *activity* specifies the executed

process step, and *timestamp* records when it occurred. In addition to the attributes caseID, timestamp, and activity, the event log can also record additional attributes such as resource information and cost. A *trace* σ is a sequence of events with the same caseID, ordered by timestamp. We denote by Σ_L the ordered set of all traces in L, sorted by the timestamp of their first event.

Table 1: Example of an event log.

Case ID	Timestamp	Activity	Resource	Cost
1	10-01-2026 14:17	Order received	System	0
1	11-01-2026 12:02	Print component plan	Robin	50
1	15-01-2026 13:06	Obtain from warehouse	Amir	200
1	20-01-2026 16:12	Assemble parts	Robin	100
1	22-01-2026 15:18	Final inspection	Omar	200
1	23-01-2026 10:01	Order dispatched	Omar	80
2	12-01-2026 15:34	Order received	System	0
2	13-01-2026 11:32	Print component plan	Robin	50
2	17-01-2026 12:12	Order components	Robin	80
2	18-01-2026 17:16	Assemble parts	Peter	200
2	22-01-2026 11:42	Final inspection	Omar	200
2	23-01-2026 11:33	Order dispatched	Omar	70
...	...	...	...	...

Table 1 shows a snapshot of an event log generated from the execution of the business process illustrated in Figure 1. Each row represents one event through its recorded attribute values. For example, the first event belongs to Case 1. It has the timestamp "10-01-2026 14:17" and refers to the activity "Order received," executed by a system without incurring cost. The trace for Case 1 consists of the first six rows, all sharing Case ID 1.

Process characteristics. Using information recorded in an event log, a business process can be described through *process characteristics*, that is, measurable properties that capture overall behavior of a process as a system over time. These characteristics are derived from an event log by analyzing multiple cases and their event attributes to capture aggregated behavior for a certain period. Established examples of such process characteristics are the number of arriving cases, active cases (workload), and completed cases over a certain period. Another example that reflects the time perspective is the average lead time, that is, the average duration between the start and completion of a case. The extraction and analysis of such process characteristics support process monitoring and enable the identification of steady and non-steady states of a business process.

Steady states and the SSD problem. A steady state refers to a condition in which the behavior of a system remains constant over time [2], making its behavior predictable and allowing for more precise and meaningful analysis. The study of steady states has a long history and has proven to be important in various fields, including mechanics [11], biology [12], and ecology [13].

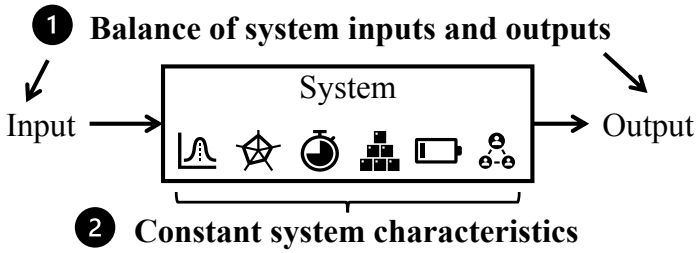


Figure 2: Two key properties of a steady state.

Steady state of a business process. In process mining, we define a business process to be in a steady state if its system-level behavior remains stable and consistent over time. As shown in Figure 2, a business process can be represented as a system with its inputs, internal characteristics, and outputs. The steady state of a business process can then be characterized by the following two properties:

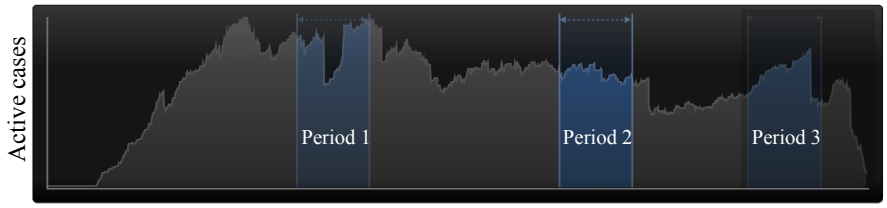
1. *Balance of system inputs and outputs*: A system in a steady state maintains a balance between input and output, ensuring that no significant fluctuation occurs over time. In the context of a business process, this means, e.g., that the ratio of incoming and completed cases remains consistent over time.
2. *Constant system characteristics*: The characteristics of a system in a steady state remain consistent. For a business process, this could mean that, e.g., the number of active cases remains stable.

It is important to note that when examining the system-level behavior of a process, we focus on process characteristics that provide a holistic description of its behavior that evolves over time, exhibiting notable fluctuation.

The SSD problem. In the context of process mining, we define the SSD problem as the task of detecting periods when system-level process behavior, derived from an event log, remains in a steady state.

Importance of SSD in process mining. To illustrate the importance of SSD in process mining, we examine how the performance of a business

process, measured by average and median lead times, can differ between steady and non-steady periods, and the implications this may have on a downstream process mining task. For this purpose, we use a real-life event log describing a permit application process at a municipality (BPIC2015-2) [14] as a running example in this section. The event log contains 44,354 events, capturing the execution of 832 cases over a period of approximately 5 years. During this period, the process exhibits an average lead time of 22.9 weeks, with a median lead time of 15.5 weeks. For simplicity, we focus on a single system-level process characteristic, namely the number of active cases, when examining the steady state of a business process.



			Steady State	
Measure	**Event log**	**Period 1**	**Period 2**	**Period 3**
Number of traces	832	23	23	25
Duration	56 months	5 months	5 months	5 months
Average lead time	22.9 weeks	10.7 weeks	5.6 weeks	12.1 weeks
Median lead time	15.5 weeks	11.3 weeks	3.6 weeks	13.9 weeks

Figure 3: Comparison of process performance between different periods.

The number of active cases, shown in Figure 3, indicates that the process was not steady throughout the recorded timeframe, with both stable and unstable periods of process behavior. For instance, in Period 1, spanning 5 months and involving 23 cases, the process shows instability, marked by a significant drop in active cases. The average lead time is 10.7 weeks, with a median of 11.3 weeks. Period 2, also 5 months long with 23 cases, is more stable, with fewer fluctuations in active cases. The average lead time is 5.6 weeks, and the median is 3.6 weeks. Lastly, Period 3 exhibits a rise and fall in active cases, indicating a non-steady state. It has an average lead time of 12.1 weeks and a median lead time of 13.9 weeks.

Overall, we observe that performance in the steady state (Period 2) is almost twice as high as in the other periods and about four times higher than the overall average. When performance differs strongly between steady

and non-steady states, SSD becomes an important pre-processing step for performance analysis in process mining. It allows for more accurate insights into how process performance varies between steady and non-steady states.

3. Steady-State Detection Framework for Process Mining

This section presents our proposed SSD framework, illustrated in Figure 4. The framework takes an event log as input and proceeds through three main steps:

1. *Time series extraction*: Different process characteristics are extracted from the event log as time series, representing their evolution over time.
2. *Steady-state detection*: The extracted time series are analyzed using existing SSD techniques to identify periods during which the process operates in a steady state.
3. *Steady-state grouping*: The detected steady states are subsequently grouped based on similar process behavior, enabling a more detailed analysis of these states in downstream process mining tasks.

As output, the framework identifies the steady and non-steady states, their group assignments, and the corresponding periods. This information can be used to extract state-specific event logs containing the relevant events or cases for each state, enabling more precise insight generation for various downstream process mining tasks, as demonstrated in Section 5.

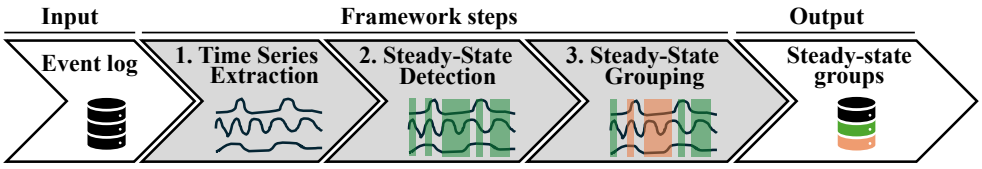


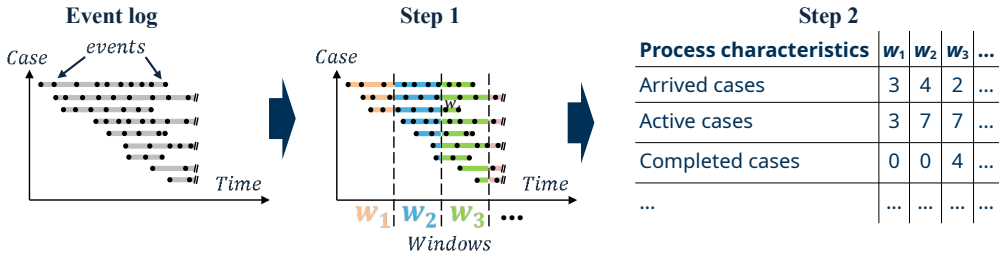
Figure 4: Overview of the main steps of our framework.

To ensure flexibility, our framework provides several configuration options at different stages, which can be used to achieve a more effective SSD in event logs. Since the optimal choice often depends on the characteristics of the data, we do not commit to a specific design choice. However, in our evaluation, we examine the impact of these configuration options on detection accuracy for a given data collection.

3.1. Time Series Extraction

In Step 1, we generate time series from an event log to capture the evolution of process characteristics relevant to SSD. Such transformations are widely used in process mining, for purposes including business process simulation [15], assessing process resilience [16], and evaluating process complexity [17]. Below, we outline the specifics of this step.

Figure 5 illustrates how process characteristics are derived as time series from an event log through *windowing* and *time series construction*.



Process characteristics	w_1	w_2	w_3	...
Arrived cases	3	4	2	...
Active cases	3	7	7	...
Completed cases	0	0	4	...
...	...	...	...	...

Figure 5: Time series extraction from an event log.

Windowing. Our approach takes an *event log* L and divides its entire timeframe into $n \in \mathbb{N}$ equally spaced *time windows* $W_l = \langle w_1, \ldots, w_n \rangle$, each with a fixed length l (e.g., a day or a week). Consequently, each event $e \in L$ is assigned to exactly one time window w_t, where $t \in \{1, \ldots, n\}$.

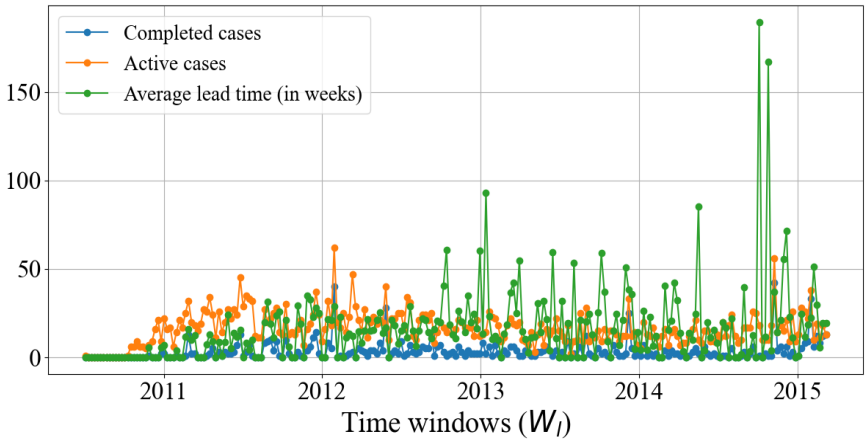


Figure 6: Outcome of the first framework step.

Time series construction. Next, we construct time series over $w_t \in W_l$ for different process characteristics. In our framework, we consider four process characteristics that are relevant for SSD and can be derived from a standard event log L: *the number of arrived cases* (arc), *the number of active cases* (ac), *the number of completed cases* (cc), and *the average lead time* (alt) of completed cases during a time window w_t. If the event log includes further information, such as resource details, additional process characteristics can be considered to enrich the process representation.

We use $y^f_{w_t} \in \mathbb{R}$ to denote the value of a characteristic $f \in F = \{ac, cc, alt\}$ during a time window w_t. For each feature, we concatenate these values into a *time series* $\{y^f_{w_t}\}^n_{t=1}$, which captures the evolution of f over the time windows in W_l. Figure 6 shows the outcome of this step with weekly windowing for a real-life event log[1].

3.2. Steady-State Detection

After extracting the time series, the next step is to identify periods during which the process operates in a steady state. This is carried out in two stages. First, at the time-series level, each process characteristic is analyzed individually to determine its steady or non-steady periods. Second, at the overall process level, the results from the individual time series are combined to form an integrated view of whether the process as a whole is in a steady or non-steady state.

SSD at time series level. For each time series $\{y^f_{w_t}\}^n_{t=1}$, we derive a corresponding *binary time series* $\{p^f_{w_t}\}^n_{t=1}$, with $p^f_{w_t} \in \{0, 1\}$ for each time window w_t using an existing SSD technique. This binary time series shows whether the process characteristic is in a steady state during w_t, with $p^f_{w_t} = 1$ for steady and $p^f_{w_t} = 0$ for non-steady.

To obtain binary time series $p^f_{w_t\,t=1}{}^n$, several options are available. Since the SSD problem has been studied in many domains, our framework offers several commonly used techniques as selectable options:

- RW (*Rolling Window*) [18]: Detects steady states by comparing short- and long-term rolling averages. A steady state is identified when the short-term mean deviates from the long-term mean by less than a threshold based on the long-term standard deviation.

[1] In this example, we use the the BPIC2015-2 event log, available at https://data.4tu.nl/collections/BPI_Challenge_2015/5065424/1.

- CS (*Cumulative Sum*) [19]: Monitors cumulative increases and decreases in the signal and flags transitions when deviations exceed a threshold. A drift correction term reduces false detections from gradual trends.
- VR (*Variance Ratio*) [20]: Compares long-term and short-term exponentially weighted measures of variability. A steady state is detected when their ratio remains below a specified threshold.
- EDP (*ED-Pelt with Transitions*) [21]: Segments the time series into statistically homogeneous regions using the Energy Distance Pruned Exact Linear Time (ED-PELT) change point detection algorithm.
- NBD (*Noise Band Differential*) [22]: Identifies steady and non-steady states by analyzing noise-band amplitude rather than relying on predefined models, adapting automatically to signal characteristics.
- VMA (*Variable Moving Average*): Computes a moving average that adapts to local signal variability. Low variability indicates steadiness, while high variability signifies transience.
- SL (*Slope-based Detection*): Fits linear regressions over sliding windows to estimate local trends. A steady state is detected when the slope is not statistically different from zero.
- PSL (*Probabilistic Slope-based Detection*): Extends SL by probabilistically aggregating slope significance results across windows, yielding a continuous steadiness probability for each time point.
- CST (*Change Point Detection with Stationarity Tests*): Identifies steady and non-steady states using change-point detection technique followed by stationarity testing. Given that time series stationarity has some conceptual overlaps with steady state detection, we propose this method to evaluate how well stationarity testing performs.

We recommend PSL as the default option based on the evaluation results discussed in Section 4. Beyond these techniques, the framework is compatible with any SSD technique that takes a real-valued time series as input and produces a binary time series.

SSD at process level. After performing SSD per process characteristic, we next aggregate the information from the binary time series to determine if indeed the entire process can be considered to be in a steady state during a time window w_t. To provide flexibility, our framework supports several straightforward aggregation options, but it is not limited to these:

- *Single-source aggregation* classifies a time window w_t as a steady-state

period if $p^f_{w_t} = 1$ holds for at least one process characteristic f.

- *Majority-based aggregation* deems a time window w_t as a steady-state period if $p^f_{w_t} = 1$ holds for at least 50% of the process characteristics.
- *Consensus-based aggregation* considers a time window w_t as a steady-state period if $p^f_{w_t} = 1$ for all process characteristics.
- *Kernel-based aggregation* extends the three options above by providing a smoother aggregation of $p^f_{w_t}$ from all process characteristics.

The kernel-based option first aggregates insights from different process characteristics by calculating the average value across all binary time series $\{p^f_{w_t}\}^n_{t=1}$ for each time window. Then, it applies a Gaussian filter [23] with a kernel of three standard deviation to smooth the curve and reduce fluctuations. After smoothing, the time series is rescaled using Min-Max normalization to ensure that the values lie between 0 and 1. As outcome, we obtain a *steady-state probability curve* as a time series $\{P_{w_t}\}^n_{t=1}$ with $P_{w_t} \in [0, 1], \forall w_t$. The steady-state probability curve represents the likelihood of a process to be in a steady state during the time window w_t. Finally, to identify the periods of steady states of a business process, we compare the values of the steady-state probability curve with a *consensus threshold* $\tau \in [0, 1]$. If $P_{w_t} \geq \tau$, the time window w_t is considered to be part of a steady state; otherwise, as non-steady. Figure 7 illustrates the detected steady states for a given steady-state probability curve (blue curved line), assuming a consensus threshold $\tau = 0.7$.

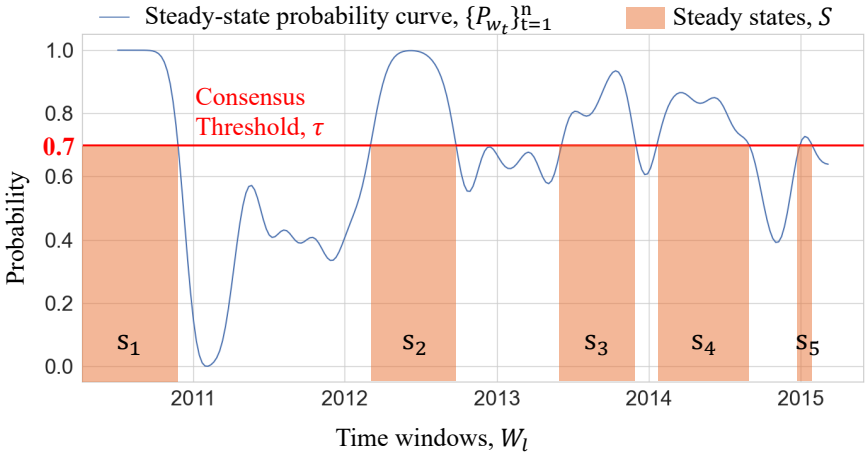


Figure 7: Steady-state detection using kernel-based aggregation.

After aggregation, each time window in W_l is labeled as steady or non-steady. We define a *steady state*, denoted s_i, as a sequence of consecutive

steady-state time windows from W_l. All detected steady states are combined into a sequence of steady states $S = \langle s_1, \ldots, s_d \rangle$. In the example in Figure 7, this results in five steady states, each covering a distinct range of consecutive steady-state time windows in W_l.

3.3. Steady-State Grouping

In the final step, we combine the detected steady states into steady-state groups. A *steady-state group* consists of one or more steady states that exhibit similar behavior in the process characteristics derived in Step 1 during the corresponding steady-state periods identified in Step 2. This approach enables us to distinguish not only between steady and non-steady periods, but also between steady periods that belong to different steady-state groups. These groups can then be analyzed separately using state-specific event logs in subsequent process mining tasks.

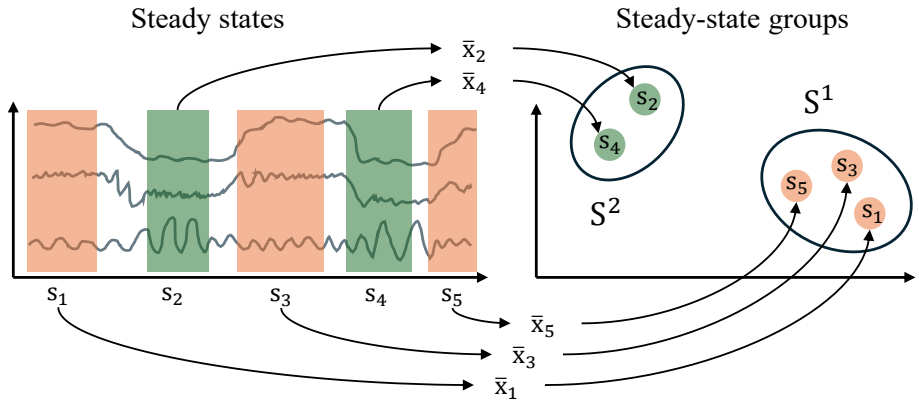


Figure 8: Steady-state grouping.

To achieve this, we proceed in two stages, as illustrated in Figure 8. First, we extract descriptive feature vectors $\overline{x}_i$ from the process characteristics associated with the periods corresponding to the detected steady states. Next, we apply clustering to these vectors in order to group them and thereby identify which steady states exhibit similar process behavior and should be assigned to the same steady-state group.

Feature extraction. For each steady state s_i, we use the values of the process characteristics obtained in Step 1 to derive numerical features that form an multidimensional vector summarizing the process behavior during s_i. The numerical features in the vectors can be derived in various ways.

In our framework, we provide two options for feature extraction: calculating the *mean* of each process characteristic, or combining both the *mean* and *standard deviation* to obtain a richer representation. Other aggregation techniques can also be integrated, as long as they yield values that can be consistently combined into a feature vector. For example, in Figure 8, we compute for each steady-state period s_i the mean of every process characteristic observed in that period. These values form the vector $\overline{x}_i$ that represents the steady state s_i.

Clustering. After feature extraction, we cluster the steady states based on their feature vectors to form *steady-state groups*, denoted SS_g. Each group contains steady states that correspond to periods in which the process exhibits similar behavior. Formally, each steady state s_i is assigned to a group g, written as $s_i^{(g)}$, and the steady states in the same group form $SS_g = \langle s_1^{(g)}, \ldots, s_{m_g}^{(g)} \rangle$. All remaining windows that are not assigned to any steady state constitute the *non-steady-state group* SS_0.

Since the number of groups is unknown, we use clustering techniques that do not require predefining the number of clusters. The framework supports affinity propagation [24] and k-means [25] with automatic cluster selection using the Elbow [26] and Silhouette methods [27]. Other clustering methods without predefined cluster numbers can also be integrated. When only a few steady states are detected, it is advisable to focus on key characteristics, such as the number of active cases and the average lead time, to avoid unstable clustering results.

Output. As output, our framework returns the *set of detected steady-state groups*, defined as $SS = \{SS_0\} \cup \bigcup_{g=1}^{k} SS_g$, where k is the number of groups, SS_g denotes steady-state group g, and SS_0 contains all non-steady-state time windows from W_l. Each time window from W_l belongs to exactly one SS_g.

State-specific event log extraction. Depending on the goal of the downstream process mining tasks, users can extract the events or traces associated with each steady-state group to obtain a *state-specific event log*, denoted L_{SS_g}. Such a log can be created, for example, by selecting all events whose timestamps fall within the time windows of the steady states in group g and combining them for further analysis.

4. Evaluation

This section describes the experiments we conducted to assess the accuracy of our framework in detecting groups of steady and non-steady states.

For this, we use a synthetic collection of event logs with known gold standard indicating when a process is in a steady or non-steady state. The experimental setup is presented in Section 4.1, followed by the evaluation results in Section 4.2. To ensure reproducibility, all data, implementation details, configurations, and raw results are available in a public repository[2].

4.1. Setup

Data collection. To evaluate our work, we need event logs with known periods of steady and non-steady states. Since no suitable public collection exists, we create our own data collection. For this purpose, we simulate a business process and define several scenarios that produce event logs containing both steady and non-steady periods.

Simulation approach. We generated data by simulating the process that is represented by the BPMN diagram in Figure 9. The process consists of ten activities and includes various control-flow patterns such as skipped steps, decision points, and parallelism. The simulation was conducted using AgentSimulator [28], an open-source tool for discrete event simulation. As the original purpose of the tool is to discover and simulate process models from event data automatically, we had to extend its functionality to simulate the desired process model of interest and different simulation scenarios with varying model parameters[3].

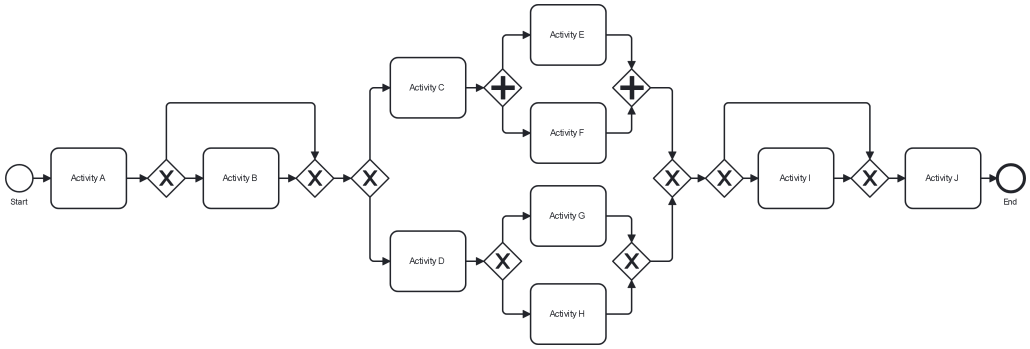


Figure 9: Simulated business process.

[2]Project repository: https://gitlab.uni-mannheim.de/processanalytics/ssd.
[3]Adjusted AgentSimulator: https://github.com/akprojectshub/AgentSimulator_adj.

Simulated scenarios. To generate our collection of event logs with varying numbers of steady and non-steady states, we designed six simulation scenarios of increasing complexity, as illustrated in Figure 10.

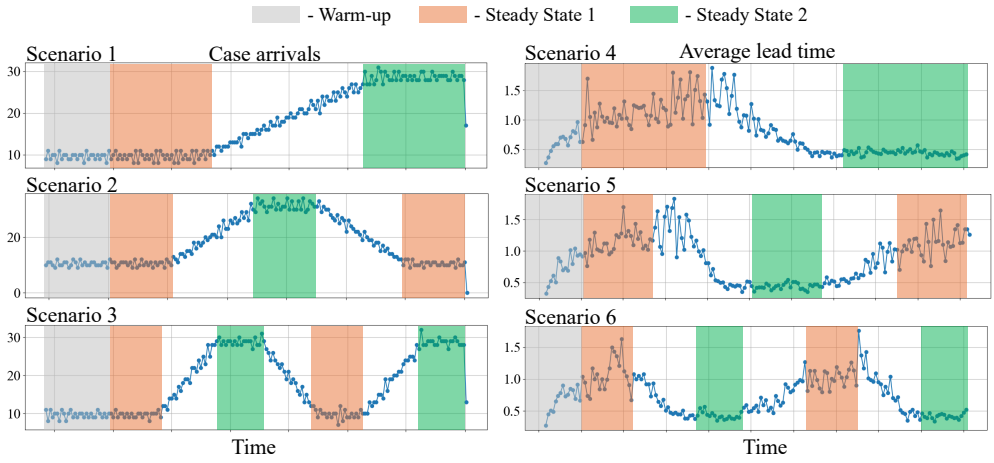


Figure 10: Simulated number of arrived cases for each scenario.

In the first three scenarios, we varied the case arrival rate to obtain different steady states. Specifically, in Scenario 1, we started with a fixed arrival rate to simulate operations during the first steady state and then linearly increased it, resulting in a second steady state with three times higher arrival rate. This setup produces a single transition period, representing one non-steady state. In Scenarios 2 and 3, we increased the number of transitions to two and three, respectively, creating a repeating pattern of two distinct steady states separated by non-steady states. The detection complexity increases accordingly, as the total number of traces and covered periods remain constant. This gradual increase in complexity enables a systematic evaluation of the framework's ability to detect multiple steady and non-steady states.

In the remaining three scenarios, we varied the number of available resources. Compared to the first three scenarios, this leads to more complex behavior, since changes in resource availability affect process characteristics in a less linear way, as shown in Figure 10 by the visualization of the average lead time. Similar to the first three scenarios, we defined different patterns in terms of steady and non-steady states.

Scenario parameters. The specific parameters that differ between the scenarios are listed in Table 2. In the first three scenarios, only the arrival rates are varied, while the number of available resources is fixed at 10. All resources

Table 2: Simulation parameters for each scenario.

Scenario ID	Arrival rate	Task duration	Available resources
	(cases/week)	*(hours)*	*(number)*
1	10–30	Uniform($[3, 5]$)	10
2	10–30–10	Uniform($[3, 5]$)	10
3	10–30–10–30	Uniform($[3, 5]$)	10
4	20	Uniform($[8, 9]$)	10–12
5	20	Uniform($[8, 9]$)	10–12–10
6	20	Uniform($[8, 9]$)	10–12–10–12

are assumed to be cross-trained and capable of performing any activity in the process once they become available. For simplicity, the process is assumed to operate continuously (24/7). The execution time of each activity follows a uniform distribution between 3 and 5 hours.

In Scenarios 4–6, we fix the number of arriving cases per week and vary the number of available resources by 20%, switching between 10 and 12 resources. The execution time for each activity is increased to ensure that the system experiences longer lead times when 10 resources are available compared to when 12 resources are available.

Each event log contains 3,300 traces, with the first 300 traces used as a warm-up period and therefore excluded from the evaluation. To ensure a robust evaluation, we generated 30 event logs for each scenario, resulting in a total of 180 event logs. Finally, each event log includes an equal number of steady and non-steady time windows, ensuring that both classes are balanced.

Framework configurations. Next, we discuss the framework configurations applied at each step and evaluated in this experiment:

Step 1. In Step 1 of our framework, we evaluate window sizes between 3 and 14 days. Shorter windows produce time series with many zero values, and longer windows yield too few data points for the SSD techniques used in Step 2. We consider four process characteristics: the number of arrived cases (arc), the number of active cases (arc), the number of completed cases (cc), and the average lead time of completed cases (alt), that is, $f \in F = \{\text{arc}, \text{ac}, \text{cc}, \text{alt}\}$.

Step 2. At the time series level, we evaluate all nine techniques discussed in Section 3.2: Rolling Window (RW), Cumulative Sum (CS), Variance Filter

Table 3: Tested parameter settings for SSD techniques in Step 2.

Technique	**Settings**
RW	$I_{long} = \{\mathbf{20}, 30, 40, 50, 60\}$, $I_{short} = \{1, 2, 3, \mathbf{5}, 10\}$, Threshold $= \{0.1, 0.25, 0.5, \mathbf{1.0}\}$
CS	Predict $= \{True, \mathbf{False}\}$, $\alpha = \{\mathbf{0.02}, 0.04, \ldots, 1.00\}$
VR	Ratio $= \{\mathbf{2}, 5, 10\}$, $\alpha_1 = \{\mathbf{0.1}, 0.3, 0.5\}$, $\alpha_2 = \{0.2, 0.35, \mathbf{0.5}\}$, $\alpha_3 = \{0.01, 0.05, \mathbf{0.1}\}$
EDP	Variance $= \{300, 350, \mathbf{400}, \ldots, 750\}$, Slope $= \{-5, 1, 5, 10, 20, 30, 40, 50, 60, \mathbf{70}\}$
NBD	Data portion $= \{0.01, 0.02, 0.03, 0.04, \mathbf{0.05}, \ldots, 0.10\}$, Neighbors $= \{1, 3, 5, 7, 10, 13, \mathbf{15}, 17, 20, 25\}$
VMA	Window $= \{1, 2, \ldots, \mathbf{7}, \ldots, 100\}$
SL	Window $= \{10, 15, 20, \mathbf{25}, 30\}$, Step $= \{1, 3, \mathbf{5}, 7, 10\}$, $\alpha = \{0.01, \mathbf{0.05}, 0.1, 0.5\}$
PSL	Window $= \{10, \mathbf{20}, 30\}$, $\alpha = \{\mathbf{0.01}, 0.25, 0.05\}$, Step $= \{\mathbf{1}, 5, 10\}$, Threshold $= \{0.2, \mathbf{0.5}, 0.7\}$
CST	Window $= \{\mathbf{20}, 30, 40, 50, 60\}$, n-tests $= \{2, \mathbf{3}, 4, 5\}$, Consensus $= \{\mathbf{0.2}, 0.4, 0.5, 0.6, 0.8\}$.

Note: For EDP and PSL, 81 parameter settings were tested; for all other techniques, 100 settings were evaluated. The highlighted values indicate the parameter configurations that yielded the best results in Experiment 1.

(VR), ED Pelt with Transitions (EDP), Noise-Based Differential (NBD), Variable Moving Average (VMA), Slope-Based Detection (SL), Probability Slope-Based Detection (PSL), and change point detection with stationarity tests (CST). For each technique, we test up to 100 parameter combinations, summarized in Table 3.

At the process level, we consider four aggregation techniques: single-source aggregation, majority-based aggregation, consensus-based aggregation, and kernel-based aggregation. For the kernel-based approach, we use Gaussian kernels with standard deviations of 2, 3, and 4. In addition to the aggregation-based SSD, we also consider results obtained without aggregation, that is, when the decision is based solely on one of the process characteristics $f \in F$.

Step 3. We consider three clustering approaches. The first is based on Affinity Propagation (AP), the second on adaptive k-means using the elbow method (KM-E), and the third on adaptive k-means guided by the silhouette score

(KM-S). For each clustering approach, we evaluate two feature extraction options: mean, and mean combined with standard deviation.

In total, we considered 51,720 different framework configurations for each of the 180 event logs, resulting in 9,309,600 framework executions.

Evaluation measures. To evaluate the accuracy of our framework in classifying time windows across multiple states (non-steady state 0, steady state 1, and steady state 2), we use the Matthews correlation coefficient (MCC) [29]. This metric is suitable for multiclass classification since it considers all entries of the confusion matrix and provides a balanced assessment across classes. It is defined as

$$\text{MCC} = \frac{cs - \vec{t} \cdot \vec{p}}{\sqrt{(s^2 - \vec{t} \cdot \vec{t})(s^2 - \vec{p} \cdot \vec{p})}}, \tag{1}$$

where C is the confusion matrix and:

- $c = \sum_k C_{kk}$ is the number of correctly predicted time windows,
- $s = \sum_i \sum_j C_{ij}$ is the total number of time windows,
- $\vec{t}$ is the vector of true class counts with $t_k = \sum_i C_{ik}$,
- $\vec{p}$ is the vector of predicted class counts with $p_k = \sum_i C_{ki}$.

The MCC ranges from -1 to $+1$, where $+1$ indicates perfect classification, 0 corresponds to random prediction, and negative values indicate disagreement between predictions and true labels.

In addition to MCC, we report class-specific performance measures in Section 4.2.4. For each group of steady and non-steady states, we compute precision, recall, F1-score, and specificity. For a group g, these measures are defined as:

$$\text{Precision}_g = \frac{TP_g}{TP_g + FP_g}, \qquad \text{F1}_g = 2 \cdot \frac{\text{Precision}_g \cdot \text{Recall}_g}{\text{Precision}_g + \text{Recall}_g},$$

$$\text{Recall}_g = \frac{TP_g}{TP_g + FN_g}, \qquad \text{Specificity}_g = \frac{TN_g}{TN_g + FP_g}.$$

Here, TP_g, FP_g, FN_g, and TN_g denote the true positives, false positives, false negatives, and true negatives for class g, respectively. All four measures range from 0 to 1, with higher values indicating better classification performance for the corresponding state.

4.2. Results

To assess the overall accuracy of the proposed framework and the effect of the tested configurations, we present the evaluation results step by step.

First, in Section 4.2.1, we assess the sensitivity of the framework to the window size parameter used in Step 1. Next, in Section 4.2.2, we examine the impact of different SSD techniques applied in Step 2 to detect steady-state periods at both the time series and process levels. Then, in Section 4.2.3, we analyze the impact of different clustering options in Step 3. Finally, in Section 4.2.4, we examine the types and extent of the observed misclassifications and discuss possible factors that may have caused them.

During each step of the analysis, we assess the influence of a specific parameter on the framework's overall accuracy while keeping the parameters of the other steps fixed at their optimal values. This ensures a fair comparison of each configuration within the same step. In addition, we report the results separately for all simulation scenarios to observe how different levels of scenario complexity affect the overall accuracy.

4.2.1. Accuracy of Step 1: Sensitivity analysis

First, we examine the sensitivity of the framework with respect to its hyperparameter used in Step 1. Specifically, we assess how the window size chosen in Step 1 affects the overall accuracy of the framework, under the assumption that we use the best-performing framework configuration in other framework steps, obtained for a 7-day window.

Table 4 reports the results for the tested window sizes across all simulation scenarios. The framework reaches its highest accuracy of 0.67 with a 7-day window, confirming this setting as optimal for the simulated steady-state scenarios. Accuracy decreases for both shorter and longer windows: the decline is moderate at first but becomes substantial, dropping by about half when moving from 7 to 3 or 14 days. Short windows do not capture stable behavior and produce time series dominated by zero values, while long windows smooth out relevant variations. Both effects reduce the ability of SSD techniques to detect steady states.

Findings. The results show that window size has a clear impact on accuracy, a known challenge in window-based time series extraction in process mining. However, in practice, testing several window sizes and selecting those that best capture variation in process characteristics can mitigate this issue. In addition, varying the window size allows analysis of steady states at different levels of temporal granularity.

Table 4: Sensitivity analysis.

Configuration	Simulated scenario						Avg ± Std
	S1	S2	S3	S4	S5	S6	
3 days	0.02	0.41	0.58	0.38	0.31	0.32	0.34 ± 0.23
4 days	0.37	0.89	**0.62**	0.44	0.43	0.37	0.52 ± 0.24
5 days	0.74	**0.95**	0.61	0.54	0.56	0.42	0.64 ± 0.18
6 days	0.95	0.90	0.59	0.53	0.55	**0.43**	0.66 ± 0.22
7 days	**0.97**	0.87	0.59	0.55	**0.64**	0.40	$\mathbf{0.67 \pm 0.22}$
8 days	0.96	0.79	0.46	**0.56**	0.62	0.24	0.61 ± 0.26
9 days	0.94	0.71	0.39	0.55	0.52	0.25	0.56 ± 0.26
10 days	0.92	0.54	0.36	0.51	0.40	0.14	0.48 ± 0.27
11 days	0.88	0.37	0.32	0.47	0.28	0.11	0.41 ± 0.27
12 days	0.87	0.35	0.32	0.47	0.15	0.12	0.38 ± 0.28
13 days	0.80	0.35	0.27	0.43	0.14	0.09	0.35 ± 0.26
14 days	0.71	0.32	0.17	0.44	0.12	0.05	0.30 ± 0.26
Avg.	0.76	0.62	0.44	0.49	0.39	0.24	0.49 ± 0.28

Note: The highlighted MCC values indicate the best result in each column.

4.2.2. Accuracy of Step 2: Steady-State Detection

This section presents the impact of the nine tested SSD techniques that can be used to instantiate Step 2. We discuss the results in two parts. First, we evaluate the accuracy of the SSD at the time series level. Then, we analyze the process-level accuracy, examining the impact of different aggregation options.

Part A: Accuracy at time series level. Table 5 reports the framework's accuracy at the time-series level, broken down by SSD technique and simulation scenario. For each technique, we apply the best-performing configuration, highlighted in bold in Table 3, that yields the highest overall framework accuracy.

Across all scenarios, the probabilistic slope-based technique (PSL) achieves the highest average accuracy with an MCC of 0.67, followed by the non-probabilistic slope-based method (SL) with 0.62. Their strong performance reflects the linear transitions used in the simulated scenarios, which align well with the assumptions of these methods. Although SL slightly outperforms PSL in Scenarios 2 and 3, the probabilistic variant is more accurate in the

Table 5: Accuracy of Step 2 (Part A).

Configuration	Simulated scenario						Avg ± Std
	S1	S2	S3	S4	S5	S6	
RW	0.87	0.77	0.48	0.44	0.45	0.26	0.54 ± 0.24
CS	0.18	0.04	0.06	0.31	0.38	0.16	0.19 ± 0.21
VR	0.83	0.62	0.39	0.42	0.39	0.10	0.46 ± 0.24
EDP	0.87	0.78	0.65	0.50	0.05	0.05	0.48 ± 0.39
NBD	0.13	0.08	0.03	0.25	0.11	0.09	0.12 ± 0.19
VMA	0.56	0.67	0.36	0.50	0.54	0.38	0.50 ± 0.20
SL	0.87	**0.88**	**0.66**	0.46	0.52	0.36	0.62 ± 0.21
PSL	**0.97**	0.87	0.59	**0.55**	**0.64**	**0.40**	$\mathbf{0.67 \pm 0.22}$
CST	0.73	0.43	0.48	0.28	0.15	0.06	0.35 ± 0.30

Note: The highlighted MCC values indicate the best result in each column.

more complex Scenarios 4–6. The rolling-window (RW) and exponential-decay prediction (EDP) techniques perform well in simpler scenarios but their accuracy drops as complexity increases, yielding average MCCs of 0.54 and 0.48. Noise-based detection (NBD) performs the weakest, with an average MCC of 0.12. Notably, only two techniques, i.e., CS and NBD, show a slight increase in MCC in the more complex Scenarios 4-6 compared to Scenarios 1-3. However, their overall performance across all scenarios remains lower than that of the other techniques.

Regarding accuracy variation, all techniques exhibit a similar overall standard deviation. This consistency stems from a gradual decline in accuracy with increasing scenario ID, reflecting the expected impact of rising process complexity.

Findings. These results lead to the following two key findings. First, existing SSD techniques from other domains can detect steady states in business processes reasonably well in simple scenarios. Second, their accuracy decreases in more complex scenarios due to nonlinear and time-dependent relations in process behavior, indicating clear room for improvement.

Part B: Accuracy at process level. Table 6 shows the framework's accuracy at the process level, differentiated by the applied aggregation options and simulated scenarios.

Configurations with aggregation generally achieve higher accuracy than

Table 6: Accuracy of Step 2 (Part B).

Configuration	Simulated scenario						Avg ± Std
	S1	S2	S3	S4	S5	S6	
Without aggregation							
Arrived cases	**0.98**	0.90	**0.62**	0.00	0.00	0.00	0.42 ± 0.43
Active cases	0.96	0.89	0.60	0.50	0.39	0.36	0.62 ± 0.26
Compl. cases	0.94	**0.93**	0.59	0.00	0.00	0.00	0.41 ± 0.43
Avg. lead time	0.00	0.01	0.00	**0.55**	0.63	0.40	0.27 ± 0.29
With aggregation							
Single	0.00	0.01	0.00	0.00	0.00	0.00	0.00 ± 0.02
Majority	**0.98**	0.91	**0.62**	0.51	0.39	0.37	0.63 ± 0.26
Consensus	0.97	0.87	0.59	**0.55**	**0.64**	0.40	$\mathbf{0.67 \pm 0.22}$
Kernel 2	**0.98**	0.89	0.59	0.52	0.44	**0.41**	0.64 ± 0.24
Kernel 3	**0.98**	0.88	0.59	0.52	0.44	**0.41**	0.64 ± 0.24
Kernel 4	**0.98**	0.87	0.61	**0.55**	0.43	**0.41**	0.64 ± 0.23

Note: The highlighted MCC values indicate the best result in each column.

those without it. The Consensus method performs best with an average MCC of 0.67, followed by Kernel 2–4 (0.64) and Majority voting (0.63). The weakest results are obtained with the Single approach, where a steady state is detected as soon as any individual characteristic indicates steadiness.

Among the non-aggregated configurations, two findings stand out. First, the number of active cases is the only characteristic that performs consistently across all scenarios, reaching an accuracy of 0.62, which is comparable to the aggregation-based methods. Second, different characteristics are informative in different scenarios. In Scenarios 1–3, arrived, active, and completed cases all serve as reliable indicators. This is expected, as in these scenarios we vary the arrival rate, which often influences active and completed cases after some delay, reflecting a lagged correlation. However, arrived and completed cases fail to generalize to the more complex Scenarios 4–6, whereas active cases remain moderately informative. In contrast, average lead time, which is less correlated with the arrival rate than active or completed cases, performs poorly in the simpler scenarios but becomes the most informative characteristic in Scenarios 4–6. This finding highlights the importance of considering complementary process characteristics that capture other rele-

vant aspects of the process.

Findings. These results lead to two insights. First, accurate steady-state detection benefits from aggregating multiple process characteristics rather than relying on a single one. Second, the selected characteristics should be complementary, that is, they should describe the process from different perspectives (such as case, time, resources, etc.) to improve detection robustness.

4.2.3. Accuracy of Step 3: Steady-State Grouping

Next, we evaluate the accuracy of the clustering options used in Step 3. We assess this in two ways: first, based on the actual results from Step 2, and second, under the assumption of perfect accuracy in Step 2, to establish an upper bound on achievable performance.

Part A: Accuracy by grouping option. Table 7 presents the accuracy of Step 3, differentiated by the applied grouping option and simulated scenarios. The results are obtained using the best configuration identified in Step 2.

Table 7: Accuracy of Step 3.

Configuration	**Simulated scenario**						**Avg ± Std**
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	
Mean							
AP	**0.97**	**0.87**	0.59	**0.55**	**0.64**	0.40	**0.67 ± 0.22**
KM-E	0.64	0.63	0.32	0.24	0.40	0.40	0.44 ± 0.17
KM-S	0.64	**0.87**	**0.60**	0.24	0.49	**0.57**	0.57 ± 0.22
Mean + standard deviation							
AP	**0.97**	**0.87**	0.59	**0.55**	0.62	0.40	0.66 ± 0.23
KM-E	0.64	0.63	0.32	0.24	0.40	0.40	0.44 ± 0.17
KM-S	0.64	0.63	0.40	0.24	0.52	0.46	0.48 ± 0.19

Note: The highlighted MCC values indicate the best result in each column.

We see that including both the mean and standard deviation as grouping features does not improve accuracy; both yield nearly identical results across all clustering methods. This indicates that mean process behavior alone provides sufficient separation. When only a few steady states are available, adding extra dimensions can even reduce accuracy due to an unfavorable feature-to-sample ratio. Among the clustering methods, Affinity Propaga-

tion (AP) performs best, achieving an average MCC of 0.67 when grouping by mean values. K-Means with the Silhouette score (KM-S) performs moderately (0.57), while K-Means with the Elbow method (KM-E) shows consistently lower accuracy (0.44).

Findings. The results indicate two main findings: grouping based on mean values is sufficient, and AP is the most accurate and robust clustering option for Step 3.

Part B: Accuracy isolated from Step 2. Table 8 shows the accuracy of Step 3 assuming perfect results in Step 2. Under this condition, Affinity Propagation (AP) and K-Means with the Silhouette score (KM-S) achieve the highest average MCC of 0.89 and perform consistently across both feature configurations. K-Means with the Elbow method (KM-E) performs considerably worse, with an average MCC of 0.70.

Table 8: Accuracy of Step 3 when assuming perfect accuracy of Step 2.

Configuration	**Simulated scenario**						**Avg ± Std**
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	
Mean							
AP	**1.00**	**1.00**	0.67	**1.00**	**1.00**	0.67	**0.89 ± 0.16**
KM-E	0.67	0.75	0.67	0.67	0.75	0.67	0.70 ± 0.04
KM-S	0.67	**1.00**	**1.00**	0.67	**1.00**	**1.00**	**0.89 ± 0.16**
Mean + Std.							
AP	**1.00**	**1.00**	0.67	**1.00**	**1.00**	0.67	**0.89 ± 0.16**
KM-E	0.67	0.75	0.67	0.67	0.75	0.67	0.70 ± 0.04
KM-S	0.67	**1.00**	**1.00**	0.67	**1.00**	**1.00**	**0.89 ± 0.16**

Note: The highlighted MCC values indicate the best result in each column within each configuration group.

Findings. This result shows that most of the accuracy loss in Step 3 is due to errors carried over from Step 2. The relative ranking of clustering techniques remains unchanged: AP and KM-S consistently outperform KM-E. Overall, when steady states are detected accurately in Step 2, Step 3 reliably distinguishes steady-state groups, with AP providing the most stable and robust performance.

4.2.4. Misclassification analysis

Finally, we analyze the misclassifications to understand their type and extent. First, we conduct a visual analysis by examining a representative event log to identify the type of misclassifications. Then, we use the confusion matrix to assess the overall extent of these misclassification types.

Misclassification types. Figure 11 illustrates the actual and detected steady states for the best-performing framework configurations using a representative event log for each simulated scenario. In Scenarios 1–3, we visualize the number of arrived cases, while in Scenarios 4–6 we focus on average lead time as the process characteristic reflecting the key process dynamics.

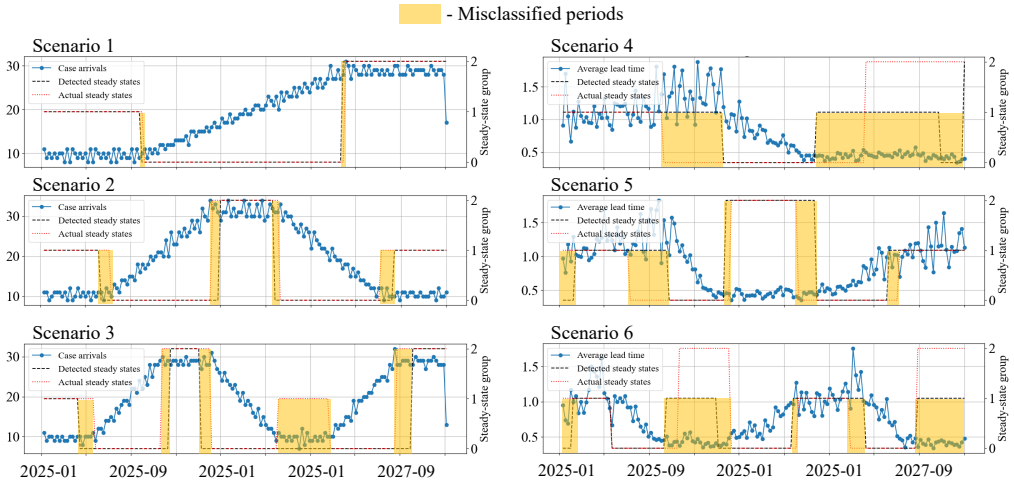


Figure 11: Visual analysis of detected vs. actual steady and non-steady states.

We identify three main reasons behind the misclassifications.

Detection latency. The most common misclassifications arise from detection latency, meaning that the steady state is identified too early or too late. This occurs in Scenarios 1, 2, and 5, with a shorter delay in Scenario 1 and a more pronounced delay in Scenario 5. Such latency results from gradual transitions between states, which make the exact change point difficult to detect. A similar delay appears in Scenario 4, likely because the process needs time to return to normal conditions after stress factors disappear.

Confusion between steady and non-steady periods. Another type of misclassification arises when steady and non-steady periods are confused, as in Scenario 3. Here, the steady state is missed, and a non-steady period is incor-

rectly classified as steady. This can occur when the steady state is very short and the transition between states is too smooth to detect clearly.

Confusion between different steady states. A third type of misclassification involves confusing different steady states, as seen in Scenarios 4 and 6. Here, the steady state of Group 2 is missed, and a steady state from Group 1 is detected instead. A deeper analysis of the data revealed that this occurred because other process characteristics, such as the number of arrived and completed cases, did not show notable fluctuations in their time series. As a result, these stable indicators outweighed the changes in lead time during the consensus-based aggregation, leading to the incorrect final decision.

Findings. This finding highlights another important and straightforward insight: steady-state detection should focus on process characteristics that are expected to be relevant for the objective of SSD. Otherwise, when too many process characteristics are included that are not relevant, the analysis may overlook the actual steady states.

Actual	Predicted groups			Total
	SS_0	SS_1	SS_2	
SS_0	9439	1398	960	11 797
SS_1	1444	5129	68	6641
SS_2	600	328	4248	5176
Total	11 483	6855	5276	23 614

a) Confusion matrix

Measure	SS_0	SS_1	SS_2	Overall
Precision	0.82	0.75	0.81	0.79
Recall	0.80	0.77	0.82	0.80
Specificity	0.83	0.90	0.94	0.89
F1-score	0.81	0.76	0.81	0.80
MCC (ϕ)	0.63	0.66	0.76	0.67

b) Performance measures per state

Figure 12: Analysis of the extent of misclassifications.

Misclassification extent. After detecting different misclassification types, we now assess the extent of the observed misclassifications by considering the confusion matrix and the resulting detailed performance measure differentiated by states, depicted in Figure 12.

Confusion matrix. The majority of cases lie on the diagonal, indicating strong agreement between predicted and actual steady-state groups. Most misclassifications occur between steady and non-steady states. Specifically, 1,398 periods of the non-steady-state group (SS_0) were predicted as SS_1 and 960 as SS_2, while the reverse occurred 1,444 and 600 times. Confusion between SS_1 and SS_2 is far less common, with 68 periods of SS_1 predicted as SS_2 and 328 of the reverse. These patterns are mainly caused by detection delays around transitions between steady and non-steady states. In contrast,

confusion between the two steady states is rare because they do not occur consecutively in the simulated data.

Performance per group. The performance measures show some differences across the three groups. SS_2 is detected most reliably, with the highest recall (0.82), strong precision (0.81), and the best MCC (0.76). SS_0 is also detected well, with balanced precision and recall around 0.80, although transitions at its boundaries introduce some uncertainty. Detection of SS_1 is the weakest, with lower precision and F1-score (0.75 and 0.76), which aligns with the confusion between SS_0 and SS_1 during transition periods. Specificity remains high for all states (0.83–0.94), and the overall MCC of 0.67 indicates solid detection performance across groups.

The performance measures show some differences across the three groups. SS_2 is detected most reliably, with the highest recall (0.82), strong precision (0.81), and the best MCC (0.76). The detection of SS_0 is also well, with balanced precision and recall around 0.80. The measures for SS_1 show the weakest performance, with lower precision and F1-score (0.75 and 0.76), which aligns with the confusion between SS_0 and SS_1 during transition periods. Specificity remains high for all states (0.83–0.94), and the overall MCC of 0.67 indicates solid performance across all groups.

Findings. The results of the misclassification analysis show that the main challenge in steady-state detection for business processes is the detection delay. This is driven by the fact that business processes are complex and dynamic systems with nonlinear and time-lagged relations due to their socio-technical nature, which makes transitions from one state to another difficult to detect precisely.

Overall findings. Overall, the evaluation confirms that the proposed framework can effectively detect steady and non-steady states in business processes using event logs. While the misclassification analysis highlights detection latency as the main challenge, this limitation primarily affects the precise timing of transitions rather than the identification of the states themselves. Moreover, the simulated scenarios were intentionally designed with increasing levels of complexity to rigorously test the framework, which makes the setting more challenging than many real-life event logs. Therefore, despite the observed detection delays, the results still provide strong evidence that the framework can reliably identify steady-state behavior and serve as a solid foundation for its application to real-life event logs, which is examined in the next section.

5. Usefulness of SSD in Process Mining

In this section, we investigate the usefulness of our SSD framework by examining its impact on three well-known downstream tasks in process mining: process discovery, process performance analysis, and predictive process monitoring (remaining time prediction). For each task, we use real-life event logs and compare the insights obtained when steady and non-steady states are distinguished with those obtained when this distinction is not applied.

In Section 5.1, we first describe the general experimental setup and the obtained groups of steady and non-steady states. Then, in Sections 5.2–5.4, we discuss each process mining task separately, outlining the task-specific setup and the corresponding results.

5.1. SSD on Real-Life Event Logs

In the following, we describe the general setup of this experiment and present the results of our SSD framework, that is, the detected groups of steady and non-steady states. For each group, we extract the corresponding event logs, which serve as input for the three considered downstream process mining tasks.

5.1.1. General Setup

Data collection. Table 9 summarizes our data collection. It consists of nine publicly available real-life event logs that are widely used in research for the three process mining tasks considered in this experiment[4]. These logs represent processes from different domains, such as a loan application process in financial services (BPIC12), building permit application processes in five municipalities (BPIC15), a health care process (Sepsis), and a billing process in a hospital (Hostipal). These logs represent real-life process executions and show differences in several aspects, such as the number of cases, variants (traces with a unique sequence of activities), recorded events, classes (the number of unique event activities), average case length, and lead time (the duration between the first and last event of a case), as shown in Table 9.

[4]We excluded the BPI Challenge 2013 and 2020 event logs due to long periods of without any activities in the process, the Traffic Fine log due to strong batching behavior, and the Environment Permit log due to the low number of events, which makes further segmentation unsuitable for the process mining tasks of interest.

Table 9: Characteristics of the used real-life event logs.

Event log	Number of				Case length		Lead time	
	Cases	Variants	Events	Classes	Avg	Max	Avg	Max
BPIC12	13 087	4366	262 200	36	20.0	175	8.6	137
BPIC15-1	1199	1170	52 217	398	43.6	101	95.9	1486
BPIC15-2	832	828	44 354	410	53.3	131	160.3	1326
BPIC15-3	1409	1349	59 681	383	42.4	123	62.2	1512
BPIC15-4	1053	1049	47 293	356	44.9	115	116.9	927
BPIC15-5	1156	1153	59 083	389	51.1	153	98.0	1344
Helpdesk	4580	226	21 348	14	4.7	15	40.9	60
Hospital	100 000	1020	451 359	18	4.5	217	127.2	1035
Sepsis	1050	846	15 214	16	14.5	185	28.5	422

Note: Average and maximum case length and lead time are measured in number of activities and days, respectively.

Framework configurations. In Step 1, we consider four process characteristics: the number of arrived cases (*arc*), the number of active cases (*arc*), the number of completed cases (*cc*), and the average lead time of completed cases (*alt*), that is, $f \in F = \{\text{arc}, \text{ac}, \text{cc}, \text{alt}\}$. We apply weekly windowing for all event logs, except for BPIC12, which covers a short time period and therefore we use daily window size. In other steps of our framework, we use the configuration from Experiment 1 that yielded the best overall performance. Specifically, in Step 2, we use the slope-based SSD technique with probability (PSL) and consensus-based aggregation. In Step 3, we apply clustering with affinity propagation based on mean values.

Derivation of state-specific event logs. To support the three downstream process mining tasks, we derive state-specific event logs by assigning each case to one steady-state group. Specifically, we link each event to a time window based on its timestamp. Since each window is classified as steady or non-steady according to Step 2 of our framework, this allows us to label events as non-steady (ID 0) or as steady with group IDs 1, 2, and so on. For each case, we count how many events fall into each state and assign the case to the state with the highest count. In this way, each case receives exactly one group label. Finally, all cases with the same label are combined into a single state-specific event log, ensuring that each case is assigned to exactly one such log. Following the notation introduced at the end of Section 3, we

denote all cases that are assigned to the steady-state group SS_g with L_{SS_g}.

Table 10: Characteristics of the state-specific event logs.

Event log	State-specific event log	Number of			
		Cases	Variants	Events	Classes
BPIC12	L_{SS_0}	2516	1056	48 261	24
	L_{SS_1}	10 571	3627	213 939	24
BPIC15-1	L_{SS_0}	122	109	3860	186
	L_{SS_1}	1077	1064	48 357	398
BPIC15-2	L_{SS_0}	30	30	1388	104
	L_{SS_1}	802	798	42 966	408
BPIC15-3	L_{SS_0}	104	89	3213	149
	L_{SS_1}	1305	1263	56 468	382
BPIC15-4	L_{SS_0}	68	67	2331	158
	L_{SS_1}	985	985	44 962	353
BPIC15-5	L_{SS_0}	85	85	3693	111
	L_{SS_1}	1071	1069	55 390	388
Helpdesk	L_{SS_0}	1457	115	6613	11
	L_{SS_1}	2936	139	13 746	13
	L_{SS_2}	187	51	989	10
Hospital	L_{SS_0}	48 154	584	207 298	18
	L_{SS_1}	51 846	677	244 061	17
Sepsis	L_{SS_0}	556	461	7946	16
	L_{SS_1}	494	425	7268	16

5.1.2. SSD Results

Table 10 presents information on the steady states detected for the nine real-life event logs. We see two key results.

First, for most event logs, our framework detects only one steady-state group (SS_1), while the remaining behavior is assigned to the non-steady-state group (SS_0). A closer view shows that SS_0 reflects the warm-up and cool-down periods at the beginning and at the end of the recorded event log period, as shown for a representative event log on the left side of Figure 13. In

this case, our framework enables us to differentiate process behavior during regular operations (SS_1) while excluding warm-up and cool-down periods during SS_0, which often show irregular behavior.

Second, for some event logs, the situation is more complex, such as for the Helpdesk log. Our framework detects two steady-state groups, as shown on the right side of Figure 13. The first group (SS_1) consists of two steady states that describe the process behavior after the warm-up period and continues until about two-thirds of the recorded event log period, with one short non-steady phase in between. After that, a longer transition takes place toward the end of the log, leading to a new steady state (SS_2).

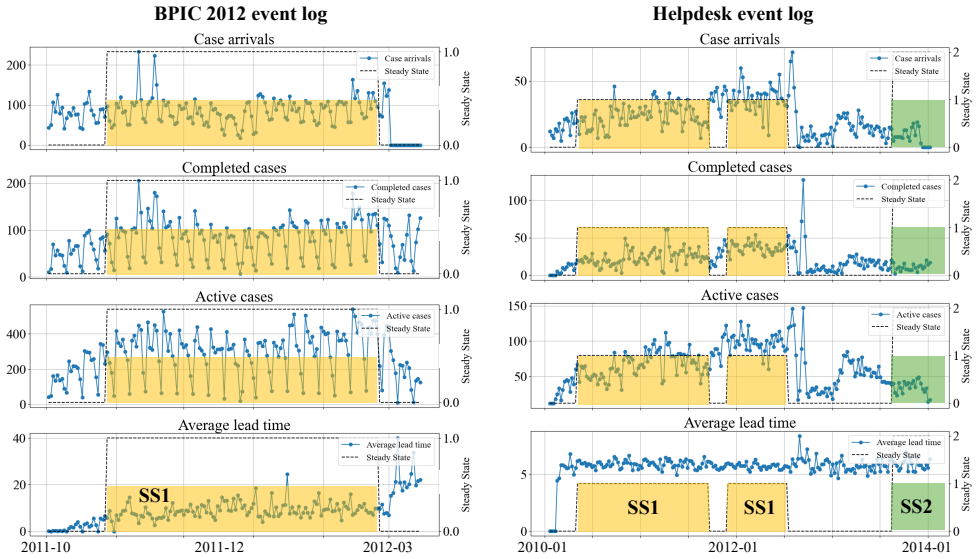


Figure 13: Outcome of our SSD framework on two exemplary real-life event logs.

Finding. Overall, this shows that our SSD framework can detect different steady states in real-life event logs. Even when only one steady state is present, the detected non-steady state reflects the typical warm-up and cool-down periods that can often be excluded to enable a more accurate process mining analysis.

In the following sections, we discuss how these steady and non-steady states influence the insights gained for different process mining tasks.

5.2. Process Discovery

In this section, we examine the impact of SSD on process discovery. The goal in process discovery is to derive a model that represents the business process and describes its control flow based on an event log that records past process executions. However, if the process operated in several steady and non-steady states, using the entire event log may lead to models that are inaccurate, since they mixes behavior from different states that reflect different operating conditions. For this reason, the experiment aims to determine whether the control flow differs when comparing the full event log with the logs that represent the detected steady and non-steady states.

Next, we first describe the task-specific experimental setup and then present the obtained results.

5.2.1. Setup

To examine whether there are differences that may affect process discovery in the control flow between the full event log and the logs representing the detected steady and non-steady states, we consider two measures:

1. **DF-relations**: The *number of directly-follow relationships* [30], that is, the number of distinct activity pairs where one activity occurs directly after another in at least one trace. In addition, this number is reported in relation to the total number of directly-follow relations in L.
2. **Fitness**: The *footprint-based fitness* between the log that contains all traces and the logs representing the steady and non-steady states [31]. Values close to 1 indicate high behavioral similarity, while values close to 0 reflect low similarity.

For the Helpdesk event log, which has two different steady-state groups (SS_1 and SS_2), we also examine the directly-follows graph (DFG) to compare the control flow of the full log with the behavior in the detected steady-state groups. A DFG is a graph in which nodes represent activities and edges represent directly-follows relations. To derive the DFGs, we use the tool Disco[5] with the activity filter set to 100 and the path filter set to 0. The tool visualizes the directly-follows relations in the event log and highlights paths to indicate the extent of a given property, such as case coverage or the average median duration between two activities.

.

[5]Available online: https://fluxicon.com/disco/.

Table 11: Behavioral differences between the state-specific event logs compared to the full event log.

Event log	State-specific event log	DF-relations		Fitness
		Count	%	
BPIC12		125	100	
	L_{SS_0}	118	94	0.99
	L_{SS_1}	125	100	1.00
BPIC15-1		4699	100	
	L_{SS_0}	821	17	0.63
	L_{SS_1}	4544	97	1.00
BPIC15-2		4802	100	
	L_{SS_0}	321	7	0.22
	L_{SS_1}	4718	98	1.00
BPIC15-3		4909	100	
	L_{SS_0}	539	11	0.57
	L_{SS_1}	4814	98	1.00
BPIC15-4		3565	100	
	L_{SS_0}	554	16	0.53
	L_{SS_1}	3462	97	0.99
BPIC15-5		4892	100	
	L_{SS_0}	385	8	0.26
	L_{SS_1}	4790	98	1.00
Helpdesk		55	100	
	L_{SS_0}	43	78	0.94
	L_{SS_1}	38	69	0.92
	L_{SS_2}	34	62	0.94
Hospital		143	100	
	L_{SS_0}	125	87	0.99
	L_{SS_1}	127	89	0.99
Sepsis		115	100	
	L_{SS_0}	107	93	0.99
	L_{SS_1}	102	89	1.00

% indicates the share of DF-relations relative to all DF-relations in the event log L.

5.2.2. Results

Table 11 presents the results that show the impact of SSD on process discovery. We see three key points.

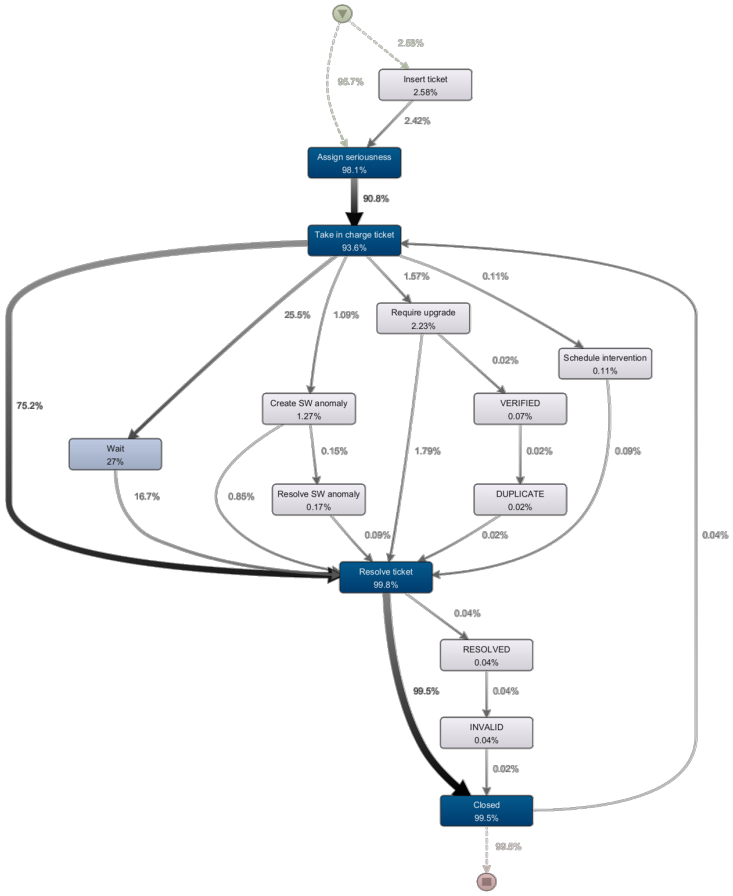


Figure 14: The DFG for the entire Helpdesk event log. The numbers indicate the case coverage, i.e., the number of cases in which the activity or the path appears.

For the BPIC15 event logs, the detected non-steady-state group SS_0 mainly captures the warm-up and cool-down periods at the beginning and end of the event log. Removing these cases does not affect the discovered process, because L_{SS_0} covers only a small share of the behavioral relations and yields fitness values between 0.22 and 0.63. In contrast, the cases assigned to the steady-state group SS_1 closely match the behavior of the full event log: the footprint-based fitness between L_{SS_1} and L reaches 1.00 or

0.99, and the directly-follows relations are almost identical.

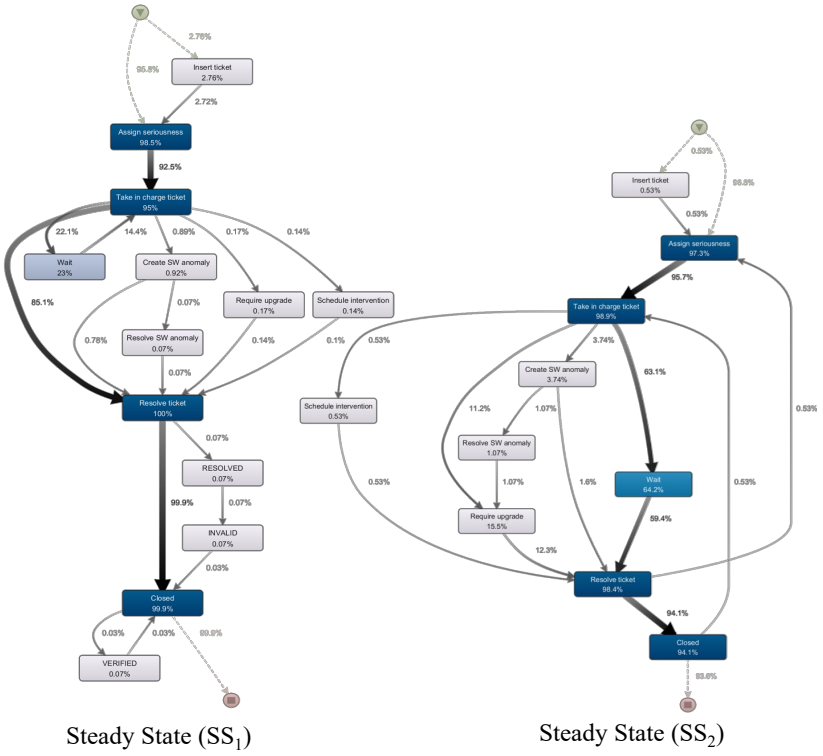


Figure 15: The DFGs for the state-specific event logs in the Helpdesk event log. The numbers indicate the case coverage, i.e., the number of cases in which the activity or the path appears.

Second, for the Helpdesk event log, we observe clear differences between the full log and the two state-specific event logs. The number of directly-follows relations decreases from 55 in the full log to 38 in L_{SS_1} and 34 in L_{SS_2}, while the fitness remains high at 0.92 and 0.94, respectively. L_{SS_2} contains 51 variants, which is 4.4 times fewer than in the full log (226) and almost three times fewer than in L_{SS_1}, as shown in Table 10.

These differences become more evident when comparing the DFG of the full log (Figure 14) with the DFGs of the two state-specific logs (Figure 15). For example, the directly-follows relation *Take in charge ticket* $\rightarrow$ *Resolve ticket* appears in 75.2% of cases in the full event log. However, when examining the DFGs of the two steady states, this path appears only in L_{SS_1}, where it occurs in 85.1% of cases, and it is absent in L_{SS_2}. Similarly, the

path *Take in charge ticket* → *Wait* → *Resolve ticket* appears frequently in the full log. In the steady-state analysis, however, this behavior is mainly associated with the second steady state, whereas in L_{SS_1} a looping pattern between *Take in charge ticket* and *Wait* is observed. Finally, the activities *Resolved* and *Invalid* appear in the full event log and in L_{SS_1}, but they are not present in L_{SS_2}. Overall, this shows that the DFG of the full event log reflects a combination of the behaviors observed in the DFGs of the two detected steady states, which differ notably in their control-flow patterns.

Finally, for some logs, such as Sepsis, Hospital, and BPIC12, we do not observe clear control-flow differences between the detected states and the full log. These logs come from domains where each case reflects a unique execution, such as medical care or financial services, so SSD does not affect process discovery.

Finding. Based on the obtained results, we can conclude that for many real-life event logs, the use of our SSD framework can improve the accuracy of process discovery. It helps avoid misleading insights by preventing the mixing of behavior that appears only in non-steady periods with behavior that reflects the steady execution of the process during different steady states.

5.3. Process Performance Analysis

In this section, we examine the impact of SSD on performance analysis in process mining. The goal of performance analysis is to compute measures that describe how the process operates with respect to characteristics of interest, such as average lead time or the ratio of successfully completed cases. These measures can become biased if we do not consider whether the process was in a steady or non-steady state, since performance in these states may differ. For this reason, the experiment aims to compare the performance measures across the detected states and to determine whether these differences are statistically significant.

Next, we first describe the experimental setup and then present the obtained results.

5.3.1. Setup

To assess the impact of SSD on process performance, we consider the *average lead time*, that is, the average time between the first and last event of each case. We report this value for the full event log and for the state-specific event logs. To determine whether the observed differences are statistically significant, we apply the Welch t-test [32], which compares two means while

allowing unequal variances and sample sizes. This test indicates whether differences in average lead time likely reflect true variation rather than random noise.

Table 12: Process performance differences between the state-specific event logs and to the full event log.

Event log	State-specific event log	Cases		Lead time (in days)		Welch t-test	
		%	Nr.	Avg.	Std.	p-value	Sig[1]
BPIC12		100	13 087	8.62	12.13		
	L_{SS_0}	19	2516	7.57	10.23	0.00	***
	L_{SS_1}	81	10 571	8.87	12.52	0.12	
BPIC15-1		100	1199	95.72	121.41		
	L_{SS_0}	10	122	53.62	56.02	0.00	***
	L_{SS_1}	90	1077	100.49	125.83	0.36	
BPIC15-2		100	832	160.10	168.52		
	L_{SS_0}	4	30	128.08	82.62	0.06	*
	L_{SS_1}	96	802	161.30	170.80	0.89	
BPIC15-3		100	1409	62.23	97.65		
	L_{SS_0}	7	104	33.99	25.82	0.00	***
	L_{SS_1}	93	1305	64.48	100.87	0.56	
BPIC15-4		100	1053	116.81	108.22		
	L_{SS_0}	6	68	99.30	173.05	0.41	
	L_{SS_1}	94	985	118.01	102.27	0.80	
BPIC15-5		100	1156	98.34	108.24		
	L_{SS_0}	7	85	111.21	95.18	0.24	
	L_{SS_1}	93	1071	97.32	109.18	0.83	
Helpdesk		100	4580	40.86	8.39		
	L_{SS_0}	32	1457	40.88	8.30	0.94	
	L_{SS_1}	64	2936	40.88	8.42	0.92	
	L_{SS_2}	4	187	40.29	8.47	0.37	
Hospital		100	100 000	127.36	130.88		
	L_{SS_0}	48	48 154	120.31	123.38	0.00	***
	L_{SS_1}	52	51 846	133.91	137.16	0.00	***
Sepsis		100	1050	28.47	60.54		
	L_{SS_0}	53	556	29.78	65.74	0.70	
	L_{SS_1}	47	494	27.00	54.12	0.63	

[1] Significance levels: * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$.

5.3.2. Results

Table 12 presents the differences in average lead time between the entire event log and the state-specific event logs.

In 6 out of 9 event logs, we observe a significant difference in process performance between the detected states and the full event log. This difference is most evident for the non-steady state SS_0, which shows a lower average lead time compared to the entire event log. The detected non-steady states in these logs correspond to warm-up and cool-down phases, as discussed in Section 5.1.2. These phases can exhibit unusual behavior and should therefore be excluded from performance analysis to obtain a reliable view of process performance in a steady state.

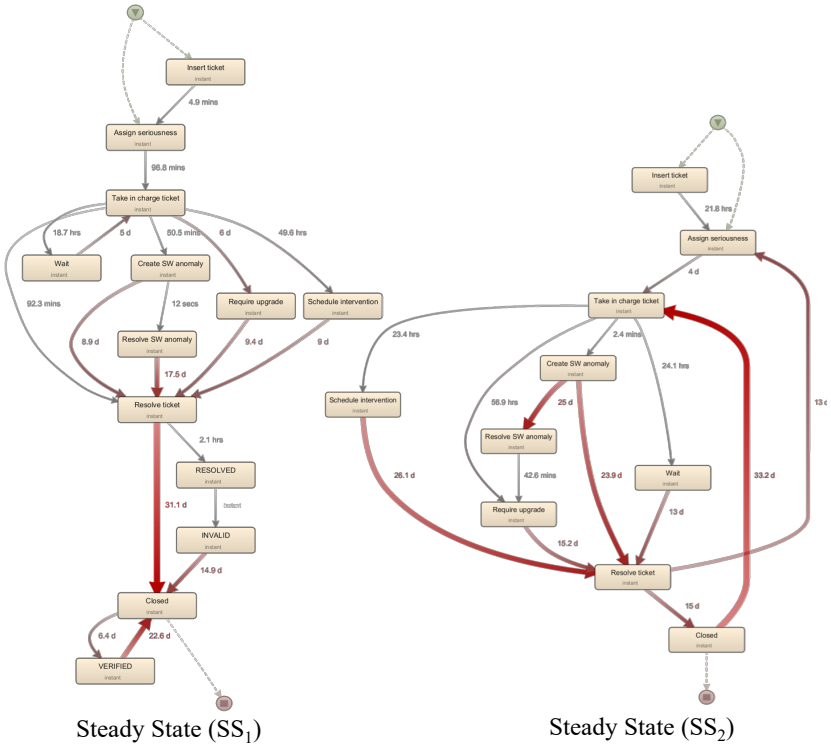


Figure 16: The DFGs for the detected steady states in the Helpdesk event log. The numbers indicate the median duration for each activity and path.

For the Hospital event log, we also observe a notable difference in the performance of the detected steady state SS_1. Compared to the performance in the full event log, the steady state shows a worse average lead

time. This suggests that the warm-up and cool-down phases captured in the detected non-steady state may falsely improve the overall performance when contrasted with the performance of the steady state alone.

Finally, for the Helpdesk event log, the two detected steady states also provide meaningful insights from a performance perspective, as shown in Figure 16. The overall average lead time of the two steady states remains similar, around 40 days. At the same time, when examining the median duration between activities, we see clear differences in the execution patterns. This indicates that the transformation affected the control flow and the timing within the process, but not the overall performance.

Findings. Based on the discussed results, we see that distinguishing process performance between steady and non-steady states can impact the permanence analysis in real-life event logs and lead to differences that are statistically significant.

5.4. Predictive Process Monitoring

In this section, we examine the impact of SSD on predictive process monitoring (PPM). We focus on the task of predicting the remaining time of an ongoing case by using a predictive model trained on past process executions recorded in an event log. When training such models, it is common to apply bucketing strategies that group training samples by their properties and train a separate model for each bucket . Typical strategies include grouping training samples by trace length or clustering them based on control flow patterns (and possibly other data attributes). During inference, an incomplete case is assigned to one of the buckets, and the corresponding trained model is used for prediction. Our SSD framework identifies traces that belong to steady and non-steady states. These states can serve as buckets for model training. Therefore, this experiment aims to compare the prediction accuracy of the bucketing strategy based on SSD with other bucketing strategies. We conduct this comparison on nine real-life event logs and with three predictive models.

In the following, we describe the experiment-specific setup and the obtained results.

5.4.1. Setup

Predictive models. We conducted experiments with three remaining time prediction approaches:

- DUMMY: A simple baseline that predicts the remaining time of an ongoing case by averaging the remaining time of training cases that share the same number of executed activities.
- DALSTM: A deep learning model based on the long short-term memory (LSTM) architecture that incorporates event data payloads into its sequential encoding [33], thereby, outperforms other LSTM-based approaches in remaining time prediction [34].
- PGTNET: A deep learning approach that transform the event log into a graph dataset, and utilizes graph transformers to balance learning from the local contexts with capturing long-range dependencies [35], demonstrating state-of-the-art results.

For DALSTM and PGTNET, we use the configurations recommended in the original papers [33, 35].

Prefix generation. We generate multiple prefixes from each trace $\sigma = \langle e_1, e_2, \ldots, e_n \rangle$ in the event log to establish training samples. A prefix of length $k \in [1, n-1]$, denoted as $\sigma^k = \langle e_1, \ldots, e_k \rangle$ represents the partial execution of σ up to its k-th event. We excluded the prefixes of length n, since they do not form meaningful prediction tasks. Moreover, PGTNET can only predict remaining time of the prefixes of length at least two. Therefore, we excluded prefixes of length one for other models to ensure a fair comparison.

Bucketing strategies. Extracted prefixes are grouped into buckets. We consider four bucketing options:

- SINGLE-BUCKETING (S-B) [36]: It serves as the baseline with no bucketing applied, i.e., all extracted prefixes are placed in a single bucket. A single prediction model is trained on all cases in the event log, enabling assessment of the benefits (if any) of other bucketing strategies.
- LENGTH-BASED BUCKETING (L-B) [37]: It partitions prefixes by their length, training a separate model for each group. Instead of using each possible prefix length as a separate bucket, prefixes of similar lengths are combined to ensure each bucket contains at least 10% of training prefixes, allowing the deep learning model to have data for training.
- CLUSTERING-BASED BUCKETING (C-B) [38]: Clustering-based bucketing groups similar prefixes and trains a separate predictive model for each cluster. Prefixes can be simply clustered based on their control-flow similarity, but this ignores how cluster membership relates to the remaining time. Instead we adapt a more effective strategy by using an auxiliary decision tree that directly predicts the remaining time and

to treat its leaf nodes as clusters. Similar to length-based bucketing, we enforce the minimum number of prefixes in each cluster (i.e., 10% of training prefixes). For DUMMY and DALSTM models, we use all available categorical attributes of the prefixes to train the decision tree. For PGTNET model, we extract a compact feature vector from each graph by combining a node-label histogram with summary statistics of the edge attributes. The node histogram captures the distribution of activity types, while the edge features encode the mean and variance of all features from temporal, resource, and data perspectives.

- STEADY STATE-BASED BUCKETING (SSD-B): This strategy groups cases based on whether they belong to steady or non-steady states detected by our SSD framework. The extraction of cases assigned to each state is described in the setup in Section 5.1. In contrast to length-based and clustering-based bucketing, this strategy paritions prefixes at the case level. In other words, two prefixes of the same case cannot belong to different buckets.

Data split. We use a 64%-16%-20% holdout split that divides data into training, validation, and testing sets while preserving the natural chronological order. This method mitigates data leakage and simulates real-world scenarios where predictions are made based on historical data [39].

Evaluation measures. To evaluate the prediction accuracy, we use *Mean Absolute Error* (MAE) that quantifies the average magnitude of absolute errors between predicted and actual remaining time. It is formally defined as: $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$, where n is the number of predictions, y_i represents the actual observed values, and $\hat{y}_i$ denotes the predicted values. Lower MAE values indicate higher predictive accuracy.

We applied the Friedman test [40] to assess whether the choice of bucketing strategy leads to statistically meaningful differences in predictive performance across the event logs for each prediction model. This test is non-parametric and suitable when only a small number of datasets is available, which aligns with our setting, as we work with nine real-life event logs and examine the strategies separately for each model. In addition, we applied the post-hoc Nemenyi test [41] to examine which pairs of bucketing strategies differ once the Friedman test indicates a significant effect. This allows us to identify, for each model, the strategies that achieve clearly better predictive accuracy across the logs.

5.4.2. Results

Table 13 reports the mean and standard deviation of the MAE for each remaining time prediction approach and bucketing strategy, across the nine event logs, computed over five runs with different random seeds.

Overall results. Overall, the results show that there is no universal strategy that performs best for all event logs and models. Length-based bucketing (L-B) performs well in many settings. It yields the lowest MAE for most DALSTM models and for several PGTNet models. Clustering-based bucketing (C-B) also gives strong results, especially for the Helpdesk and BPIC12 logs. In contrast, Single-bucketing (S-B) is less accurate in most cases. The SSD-based bucketing (SSD-B) improves performance only in a few specific settings. It gives the best results for PGTNet on BPIC15-2 and for DALSTM on BPIC15-4. For the remaining logs, SSD-B produces higher errors than the other strategies. This indicates that SSD-B is not the best choice for general predictive accuracy, but it can be helpful for some event logs.

Friedman tests. To support our observations with statistical evidence, we applied the Friedman test to assess whether the bucketing strategies yield significant differences in predictive performance across all event logs at a significance level of 95%. For the DUMMY model, the test did not show significant differences between bucketing strategies ($p = 0.137$). In contrast, for the DALSTM and PGTNet models, the Friedman test indicated significant differences between bucketing strategies, i.e., $p = 0.031$ and $p = 0.017$, respectively.

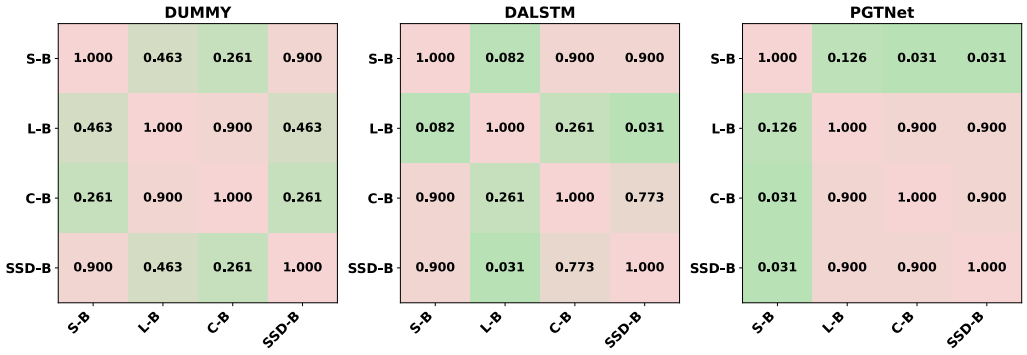


Figure 17: P-value of the pairwise Nemenyi post-hoc test.

Nemenyi tests. For the DALSTM and PGTNet models, where the Friedman test shows a significant effect of the bucketing strategy, we apply the

Table 13: Predictive accuracy measured by mean absolute error (MAE) in days, using different bucketing strategies: single-bucketing (S-B), length-based Bucketing (L-B), clustering-based Bucketing (C-B), bucketing based on steady-state detection (SSD-B).

Event log	Model	S-B	L-B	C-B	SSD-B
		Avg ± Std	**Avg ± Std**	**Avg ± Std**	**Avg ± Std**
Hospital	DUMMY	49.89 ± 0.00	**45.85 ± 0.00**	49.55 ± 0.00	57.82 ± 0.00
	DALSTM	32.33 ± 0.70	30.62 ± 0.63	**30.05 ± 0.58**	59.90 ± 5.16
	PGTNet	38.21 ± 2.42	43.39 ± 0.69	**35.62 ± 0.69**	51.05 ± 1.76
Sepsis	DUMMY	32.86 ± 0.00	**32.65 ± 0.00**	32.76 ± 0.00	38.81 ± 0.00
	DALSTM	15.67 ± 0.28	**15.51 ± 0.16**	15.66 ± 0.04	17.57 ± 0.70
	PGTNet	**16.57 ± 0.18**	18.34 ± 0.21	19.40 ± 1.06	20.95 ± 2.19
Helpdesk	DUMMY	13.09 ± 0.00	12.26 ± 0.00	**6.06 ± 0.00**	13.34 ± 0.00
	DALSTM	13.11 ± 0.36	**12.48 ± 0.62**	12.66 ± 0.03	17.91 ± 0.44
	PGTNet	**5.55 ± 0.19**	5.86 ± 0.12	6.11 ± 0.06	6.02 ± 0.22
BPIC12	DUMMY	7.85 ± 0.00	7.57 ± 0.00	**7.00 ± 0.00**	7.73 ± 0.00
	DALSTM	**5.98 ± 0.39**	5.99 ± 0.18	6.17 ± 0.26	6.35 ± 0.69
	PGTNet	**5.59 ± 0.15**	5.63 ± 0.01	5.79 ± 0.05	6.26 ± 0.46
BPIC15-1	DUMMY	40.45 ± 0.00	38.33 ± 0.00	**28.78 ± 0.00**	38.60 ± 0.00
	DALSTM	35.16 ± 6.79	**25.96 ± 0.73**	31.88 ± 6.08	32.77 ± 5.21
	PGTNet	**21.79 ± 0.14**	23.25 ± 1.80	28.13 ± 1.30	26.01 ± 3.96
BPIC15-2	DUMMY	**71.67 ± 0.00**	72.94 ± 0.00	111.12 ± 0.00	72.27 ± 0.00
	DALSTM	54.79 ± 17.04	**48.29 ± 4.33**	59.60 ± 19.13	58.90 ± 12.88
	PGTNet	65.06 ± 13.34	65.26 ± 0.60	64.11 ± 6.32	**49.11 ± 9.60**
BPIC15-3	DUMMY	26.50 ± 0.00	24.13 ± 0.00	**19.06 ± 0.00**	26.34 ± 0.00
	DALSTM	15.97 ± 1.21	**14.15 ± 0.17**	15.46 ± 1.19	20.55 ± 2.75
	PGTNet	**16.03 ± 1.01**	19.09 ± 0.29	17.89 ± 1.50	16.16 ± 0.73
BPIC15-4	DUMMY	**75.07 ± 0.00**	77.23 ± 0.00	80.20 ± 0.00	77.00 ± 0.00
	DALSTM	73.13 ± 5.01	67.08 ± 1.58	69.62 ± 0.83	**65.55 ± 12.36**
	PGTNet	**33.56 ± 1.57**	45.75 ± 1.59	57.95 ± 2.75	64.81 ± 8.98
BPIC15-5	DUMMY	48.31 ± 0.00	48.10 ± 0.00	**35.23 ± 0.00**	47.60 ± 0.00
	DALSTM	37.23 ± 6.75	36.18 ± 0.92	39.22 ± 7.51	**32.93 ± 2.59**
	PGTNet	**35.97 ± 0.98**	42.47 ± 4.06	54.41 ± 10.50	36.92 ± 3.88

Lower MAE values indicate better prediction accuracy.
The highlighted values indicate the best result in each row.

post-hoc Nemenyi test to identify which pairs of strategies differ. Figure 17 presents the results. Each subplot corresponds to one model (DUMMY, DALSTM, PGTNET) and shows the Nemenyi p-values between all strategies as a heatmap. Larger p-values indicate no meaningful difference, while smaller p-values ($p < 0.05$) mark statistically significant differences.

The post-hoc Nemenyi tests show that for DALSTM, length-based bucketing (L-B) performs significantly better than single-bucketing (S-B) and steady state-based bucketing (SSD-B). For PGTNET, single-bucketing (S-B) performs significantly better than clustering-based bucketing (C-B) and steady state-based bucketing (SSD-B). For all other pairs of strategies, we do not observe significant performance differences.

Findings. Based on the results, we observe that for some real-life event logs and certain prediction models, bucketing based on detected steady states leads to the best performance. However, we cannot conclude that SSD yields statistically significant improvements for remaining time prediction in general. Although it is possible to identify a good bucketing strategy for a specific model and dataset tested in this experiment, it is unlikely that a single strategy will perform best on all real-life event logs and for all models in predictive process monitoring in general. The optimal choice of bucketing method will likely depend on the characteristics of the underlying business process.

Overall findings. Overall, in this experiment, we examined the impact of SSD on three common process mining tasks: process discovery, performance analysis, and predictive process monitoring (remaining time prediction). For several real-life event logs, we found that SSD can have a notable effect on the accuracy of process discovery and performance analysis, as ignoring steady states may lead to misleading results. In contrast, using SSD as a bucketing strategy in predictive process monitoring did not yield statistically significant improvements. Nevertheless, the experiment highlights the relevance of SSD as an important pre-processing step in process mining, as accounting for different steady states of a business process can improve the accuracy of insights derived from event logs.

6. Discussion

In this section, we reflect on the strengths and weaknesses of the proposed framework based on the conducted evaluations.

Strengths. Our framework demonstrates several strengths. Most importantly, the findings indicate that the framework is capable of reliably identifying steady states in a business process in a data-driven manner using an event log. It achieves solid detection performance across the simulated scenarios, with an average MCC of 0.67, ranging from 0.97 in the simplest scenario (S1) to 0.40 in the most complex scenario (S6). This is also reflected in the state-specific evaluation measures presented in Figure 12.

Another point is that the framework is modular and flexible. It allows the instantiation of different SSD techniques in Step 2 and various grouping strategies in Step 3. This design ensures that the framework can incorporate future methodological advances and remain applicable as new detection or clustering techniques emerge.

Finally, the framework demonstrates practical usefulness in two classical process mining tasks: process discovery and performance analysis. The results show that ignoring steady states can lead to notable inaccuracies, whereas explicitly distinguishing between process states improves the reliability of the derived process mining insights.

Limitations. Our framework is also subject to several limitations. Although the approach is data-driven, it still requires human involvement. In Step 1, the selection of the window size is not automated and, as shown in the evaluation, can influence the SSD results. It is therefore advisable to test multiple window sizes, which also support the analysis of steady states at different levels of temporal granularity.

In Step 2, steady states are detected based on selected process characteristics; however, it is not known in advance which characteristics are most informative for a given business process. The optimal choice depends on the domain context and the available data and must therefore be determined by the user. The use of domain-specific or advanced characteristics can potentially improve the detection accuracy of the framework. In this study, we focused on common characteristics that can be derived from standard event logs. However, even with this limited set of features, we were able to demonstrate the usefulness of the framework for process discovery and performance analysis.

Finally, the accuracy of the framework depends on the performance and timeliness of the steady state detection techniques adopted from other domains in Step 2. These techniques are not specific to business processes, which are complex and dynamic systems that often exhibit nonlinear and

time-lagged relationships between variables. Therefore, we plan to develop a multidimensional steady state detection approach tailored to the business process domain in future work.

7. Related Work

This section reviews related work on steady-state detection (SSD). We first examine SSD methods that have been developed in other scientific and technical domains. We then discuss how SSD relates to similar problems in process mining.

7.1. SSD in Scientific and Technical Disciplines

In the following, we describe how the notion of a steady state is used across disciplines and review representative methods that have been developed to detect such states.

The concept of a steady state. The concept of a steady state appears in many scientific and technical disciplines to describe conditions in which the key properties of a system remain constant over time. In thermodynamics [42, 43], steady states refer to macroscopic variables such as temperature or pressure that stabilize after the transient phase. In chemical [44, 45] and mechanical [46] engineering, steady states describe continuous processes in which reaction rates, concentrations, flow rates, or throughput become time-invariant. In biology [47, 48, 49], steady states arise when production and degradation rates balance so that molecular concentrations remain stable. In ecology [50], steady states describe stable environmental conditions in natural water systems with complex and uncertain hydrological behavior. In economics [51, 52] they characterize long-term convergence patterns of macroeconomic variables. The concept also plays an important role in other fields, including signal processing [53] and queuing theory [54], where stable long term behavior is central for modeling and analysis.

Across these domains, steady states share common characteristics. A system is in a steady state when its main variables stay constant and no longer show clear trends or changes. The system reaches a stable mode in which the effects of earlier fluctuations fade, and small short-term variations do not alter the overall behavior. Under stable external conditions, this pattern remains until those conditions change. This form of stability is what connects steady state concepts across many scientific and engineering fields.

In process mining, we see an analogous notion: a process is in a steady state when its statistical properties remain stable over time. In this situation, e.g., the number of active cases does not show systematic growth or decline, and cases enter and leave the process at approximately equal average rates. As a result, key performance indicators such as waiting times, throughput times, and resource loads fluctuate only within a narrow range and do not display long-term drift. Just as in other scientific domains, this stability reflects an equilibrium between inflows and outflows under fixed operating conditions. This analogy highlights that steady states in business processes capture the same fundamental idea of persistent, time-invariant behavior that characterizes steady states in physical, biological, and economic systems.

Solutions for SSD. Since the SSD problem was first introduced in the literature [55], many methods have been developed. These approaches rely on statistical tests, time series analysis, model-based reasoning, machine learning techniques, or combinations of these ideas. Some methods are designed for specific application domains, while others address different operational settings, such as online versus offline SSD.

Earliest solutions. One of the earliest approaches relied on statistical tests [55]. The covariance similarity test (CST) divides process data into equal intervals and evaluates whether the covariance matrices of two consecutive intervals are identical. Later, the mathematical theory of evidence (MTE) was introduced [56], using the Hotelling T^2 test to compare the means and variances of successive intervals to determine whether the process is in a steady or non-steady state.

Model-based solutions. Model-based SSD approaches capture process behavior through explicit system models. An ARX-based method was proposed in [45], where an autoregressive model with exogenous inputs identifies deviations from steady conditions. Other model-based techniques include methods aimed at detecting slow drifts [57] and deterministic trend models that apply sliding windows to identify steady segments in data in a robust and computationally efficient manner [58].

Time series-based solutions. Time series techniques form another major category. Wavelet-based approaches [59, 60, 61, 50] detect steady states by exploiting multi-resolution signal properties. Other methods rely on statistical time series tests, such as fitting a first-order autoregressive model and applying the Dickey–Fuller test to assess mean reversion [62]. Bayesian approaches have also been developed [63], enabling steady state testing across

multiple data streams and estimating steady state conditions under a suitable prior.

Machine learning-based solutions. Machine learning-based solutions address SSD by learning patterns associated with steady and non-steady behavior. Gaussian discriminant analysis has been applied to gas turbine systems [64]. Clustering-based methods, such as those using k-means [65], detect steady states after removing outliers and computing trend-based indices. In power systems, steady state intervals are identified using k-means clustering and piecewise linear modeling [66]. Other methods rely on compact representations of multivariate time series to support efficient classification [67].

Domain-specific solutions. Several SSD techniques have been tailored to or tested on specific application domains. Examples include steady state detection in residential air conditioning systems for fault diagnosis [68], in computational fluid dynamics simulations to identify stabilized flow fields [69], in chemical processes [45], in gas turbine operation [64], and in the evaporation of sodium aluminate solutions [65, 70]. SSD has also been studied in power systems [66] and in industrial heat exchangers [71]. These works show that application domains often impose unique requirements that motivate specialized detection strategies.

Multi-variable solutions. Some of the aforementioned solutions address the multi-variable setting, where steady state detection must account for several interdependent variables that evolve together. Efficient methods have been proposed in [72] to detect steady conditions in such environments. Domain-specific multivariable techniques include adaptive data fusion for the alumina evaporation process [70], classification-based representations of multivariate time series [67], and clustering-based methods for industrial process data [65]. Sequential Bayesian methods have also been introduced to support online SSD by modeling joint mean and covariance in a piecewise constant form [73]. These approaches show that multivariable steady state detection often requires strategies that capture shared trends, variability, and interactions among several process variables.

Online solution. Several solutions address SSD in online settings, where decisions must be made as data arrive. A Rao–Blackwellized particle filter was proposed in [74], enabling real-time updates of steady state beliefs. Online SSD for batch processes has been studied through correlation- and distance-based tracking [75]. Bayesian inference has been applied to update autoregressive models and test steady behavior dynamically [63]. Fast

variance-based tests without data windows have also been proposed for continuous processes [76]. These methods aim to provide timely and reliable decisions in real-time environments.

Overall, the literature shows a wide range of techniques for SSD across many scientific and engineering domains. However, none of these methods addresses SSD in business processes, which have distinct forms of variability, temporal dynamics, and operational complexity. With our framework, we introduce the first solution tailored to the process mining domain. We assess the capabilities of established SSD techniques[6] in this context and extend the analysis beyond distinguishing steady and non-steady states to also differentiate between several types of steady behavior.

7.2. SSD in Process Mining

The SSD problem relates to various other problems in process mining.

Concept drift detection. The problem of SSD is related to concept drift detection in process mining, but they address different aspects. Concept drift detection identifies changes in the process that lead to a new process version [77], which operates for a certain period. In contrast, SSD focuses on the system-level behavior of the process, identifying periods where key process characteristics remain stable over time. These aspects are not necessarily correlated. For example, if a new activity (drift in the control flow) creates a bottleneck due to limited resource capacity, it is likely to impact system-level characteristics such as the average lead time. This would disrupt the steady state, potentially leading to a non-steady state or another steady state. However, if the new activity does not create a bottleneck, the system may remain in the same steady state despite transitioning to a new process version. Conversely, a business process can transition from a steady state to a non-steady state without changing its process version, for example, due to fluctuations in the arrival rate.

Business process simulation. In business process simulation, SSD can be used to address the initialization bias (or startup issue) of simulation models [78]. Many simulations begin from an empty state, causing early

[6]Only a limited number of existing SSD methods provide publicly available implementations that allow for direct testing and benchmarking. In our framework, we use several techniques for which implementations are available and which have been employed in previous studies.

fluctuations that distort results and limit analysis. The primary goal of SSD in process simulation is to identify when a process reaches a steady state, which is essential for predicting reliable long-term insights. Despite the similar terminology, the SSD problem discussed in this paper is distinct, focusing on detecting the steady state of a business process based on past recorded behavior in an event log, and serving as a crucial preprocessing step for various offline process mining techniques. We believe that the SSD problem discussed in this paper could also impact future applications in business process simulation, particularly in the automated extraction of business process simulation models from event logs.

Anomaly detection. Anomaly detection seeks to identify outliers or unusual patterns at the case level that deviate from expected process behavior [79]. In contrast, SSD focuses on identifying periods of stable, consistent process behavior across all active cases for a given period. However, SSD can provide a baseline for anomaly detection, making it easier to identify and explain unexpected behaviors. Once a steady state is reached, significant deviations can signal potential irregularities, while anomalies during non-steady states can often be explained by the process's inherent instability during that period.

Statistical quality control. The problem of SSD is closely related to statistical quality control (SQC) [80], with both aiming to monitor process stability over time. However, SSD focuses on identifying when a process has reached a steady state, where its characteristics remain relatively stable. In contrast, SQC emphasizes detecting deviations from a desired range, typically defined by specific process characteristics that reflect the process's quality or efficiency. Moreover, it is important to note that reaching a steady state does not necessarily mean the process is operating within the optimal performance range that SQC seeks to maintain. A process can be stable but still fall outside the desired limits.

8. Conclusion

In this paper, we propose a data-driven framework that detects steady and non-steady states of a business process based on information recorded in

event logs. The framework first derives time series that represent system-level process characteristics. It then applies existing SSD techniques to these time series to identify steady states. Finally, these periods are grouped based on the behavior in the process characteristics. In this way, the framework detects non-steady periods and groups of periods that belong to different steady states. Our evaluation shows that the SSD framework can identify steady states in an event log with good accuracy. The usefulness assessment further demonstrates its impact on process discovery and performance analysis. For predictive process monitoring, however, we were not able to obtain conclusive results, as there is no single bucketing technique that performs best across all settings. Nevertheless, we identified specific event logs and approaches for which our framework, when used as a bucketing technique, led to the most favorable results.

In future work, we plan to pursue two directions. First, we aim to improve the approach by addressing the discussed limitations. In particular, in Step 1 we see an opportunity to develop automated window-size selection methods based on factors such as the length of the covered period and the number of recorded events and traces. The goal is to determine a suitable window size so that the resulting process characteristics, represented as time series, provide more expressive and reliable information. In addition, correlation analysis of the extracted time series could help identify process characteristics that are less correlated and therefore more complementary, making them more likely to capture different perspectives of the process and improve steady-state detection accuracy. In Step 2, our evaluation showed that generic SSD techniques from other domains are not fully effective in the business process context. The complexity of business processes and the non-linear relations between their characteristics limit the performance of these techniques. This indicates the need for an SSD approach specifically designed for the process mining domain. Second, to gain a broader understanding of the role of SSD in process mining, we plan to study its impact on additional tasks, such as concept drift detection and business process resilience assessment. This work will further demonstrate the importance of SSD and its value for process mining research and practice.

References

[1] W. van der Aalst, Process Mining: Data Science in Action, Springer, 2016.

[2] F. E. Kast, J. E. Rosenzweig, General systems theory: Applications for organization and management, Academy of Management Journal 15 (4) (1972) 447–465.

[3] T. A. Lipo, E. P. Cornell, State-variable steady-state analysis of a controlled current induction motor drive, IEEE Transactions on Industry Applications IA-11 (6) (1975) 704–712.

[4] Z. Derzsi, Optimal approach for signal detection in steady-state visual evoked potentials in humans using single-channel eeg and stereoscopic stimuli, Frontiers in Neuroscience 15 (2021).

[5] J. v. Brocke, W. van der Aalst, et al., Process science: the interdisciplinary study of socio-technical change, Process Science 1 (2024).

[6] A. Kraus, K. A. Elyasi, H. van der Aa, On the use of steady-state detection for process mining: Achieving more accurate insights, in: International Conference on Advanced Information Systems Engineering (CAiSE), Springer, 2025, pp. 204–220.

[7] W. van der Aalst, J. Carmona, Process mining handbook, Springer, 2022.

[8] M. Weske, Business Process Management: Concepts, Languages, Architectures, 4th Edition, Springer, 2024.

[9] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, et al., Fundamentals of business process management, Vol. 2, Springer, 2018.

[10] M. Von Rosing, S. White, F. Cummins, H. De Man, Business Process Model and Notation - BPMN., OMG, 2015.

[11] D. Capecchi, F. Vestroni, Steady-state dynamic analysis of hysteretic systems, Journal of engineering mechanics 111 (12) (1985) 1515–1531.

[12] H. Kitano, Systems biology: A brief overview, Science 295 (5560) (2002) 1662–1664.

[13] E. Ranta, P. Lundberg, V. Kaitala, Ecology of Populations, Cambridge University Press, 2005.

[14] B. B. van Dongen, BPI challenge 2015. 4TU ResearchData collection (2015).

[15] M. Pourbafrani, W. van der Aalst, Discovering system dynamics simulation models using process mining, IEEE Access 10 (2022) 78527–78547.

[16] A. Kraus, J.-R. Rehse, H. van der Aa, Data-driven assessment of business process resilience, Process Science 1 (2024).

[17] M. Vidgof, B. Wurm, J. Mendling, The impact of process complexity on process performance: A study using event log data, in: International Conference on Business Process Management (BPM), Springer, 2023, pp. 413–429.

[18] E. Zivot, J. Wang, Modeling financial time series with S-PLUS, Vol. 2, Springer, 2006.

[19] F. Gustafsson, Adaptive filtering and change detection, Vol. 1, John Wiley & Sons, 2000.

[20] R. R. Rhinehart, Automated steady and transient state identification in noisy processes, in: American Control Conference, IEEE, 2013, pp. 4477–4493.

[21] K. Haynes, P. Fearnhead, I. A. Eckley, A computationally efficient nonparametric approach for changepoint detection, Statistics and Computing 27 (2017) 1293–1305.

[22] S. Yu, X. Li, Identification of steady state and transient state, Journal of Shanghai Jiaotong University (Science) 29 (2) (2024) 261–270.

[23] A. V. Oppenheim, Discrete-time signal processing, Pearson Education India, 1999.

[24] B. J. Frey, D. Dueck, Clustering by passing messages between data points, Science 315 (5814) (2007) 972–976.

[25] J. MacQueen, Multivariate observations, in: Proceedings ofthe 5th Berkeley Symposium on Mathematical Statisticsand Probability, Vol. 1, 1967, pp. 281–297.

[26] D. Pelleg, A. W. Moore, et al., X-means: Extending k-means with efficient estimation of the number of clusters., in: ICML, Vol. 1, Stanford, CA, 2000, pp. 727–734.

[27] P. J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, Journal of computational and applied mathematics 20 (1987) 53–65.

[28] L. Kirchdorfer, R. Blümel, T. Kampik, H. Van der Aa, H. Stuckenschmidt, Agentsimulator: An agent-based approach for data-driven business process simulation, in: 2024 6th International Conference on Process Mining (ICPM), IEEE, 2024, pp. 97–104.

[29] B. W. Matthews, Comparison of the predicted and observed secondary structure of t4 phage lysozyme, Biochimica et Biophysica Acta (BBA)-Protein Structure 405 (2) (1975) 442–451.

[30] W. Van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE transactions on knowledge and data engineering 16 (9) (2004) 1128–1142.

[31] J. Carmona, B. Van Dongen, A. Solti, M. Weidlich, Conformance checking, Switzerland: Springer.[Google Scholar] 56 (2018) 12.

[32] B. L. WELCH, The generalization of 'student's' problem when several different population varlances are involved, Biometrika 34 (1-2) (1947) 28–35. arXiv:https://academic.oup.com/biomet/article-pdf/34/1-2/28/553093/34-1-2-28.pdf, doi:10.1093/biomet/34.1-2.28. URL https://doi.org/10.1093/biomet/34.1-2.28

[33] N. Navarin, B. Vincenzi, M. Polato, A. Sperduti, LSTM networks for data-aware remaining time prediction of business process instances, in: 2017 IEEE Symposium series on Computational Intelligence (SSCI), 2017, pp. 1–7.

[34] E. Rama-Maneiro, J. C. Vidal, M. Lama, Deep Learning for Predictive Business Process Monitoring: Review and Benchmark, IEEE Transactions on Services Computing 16 (1) (2023) 739–756.

[35] K. Amiri Elyasi, H. van der Aa, H. Stuckenschmidt, PGTNet: A process graph transformer network for remaining time prediction of business

process instances, in: International Conference on Advanced Information Systems Engineering (CAiSE), Springer, 2024, pp. 124–140.

[36] M. De Leoni, W. M. Van Der Aalst, M. Dees, A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs, Information Systems 56 (2016) 235–257.

[37] A. Leontjeva, R. Conforti, C. Di Francescomarino, M. Dumas, F. M. Maggi, Complex symbolic sequence encodings for predictive monitoring of business processes, in: International conference on business process management, Springer, 2015, pp. 297–313.

[38] C. Di Francescomarino, M. Dumas, F. M. Maggi, I. Teinemaa, Clustering-based predictive process monitoring, IEEE transactions on services computing 12 (6) (2016) 896–909.

[39] N. Tax, I. Verenich, M. La Rosa, M. Dumas, Predictive Business Process Monitoring with LSTM Neural Networks, in: International Conference on Advanced Information Systems Engineering (CAiSE), Springer, 2017, pp. 477–492.

[40] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, Journal of the american statistical association 32 (200) (1937) 675–701.

[41] P. B. Nemenyi, Distribution-free multiple comparisons., Princeton University, 1963.

[42] N. W. Tschoegl, Fundamentals of equilibrium and steady-state thermodynamics, Elsevier, 2000.

[43] D. J. Evans, G. Morriss, Thermodynamics of steady states, Cambridge University Press, 2008, p. 283–300.

[44] H. S. Fogler, Elements of chemical reaction engineering, Pearson Educacion, 1999.

[45] F. D. Rincón, F. V. Lima, G. A. Le Roux, An arx-based technique for steady-state identification of chemical processes, in: 2015 American Control Conference (ACC), IEEE, 2015, pp. 1113–1118.

[46] J. A. McLennan Jr, Statistical mechanics of the steady state, Physical review 115 (6) (1959) 1405.

[47] N. Georgescu-Roegen, The steady state and ecological salvation: a thermodynamic analysis, BioScience 27 (4) (1977) 266–270.

[48] B. Ø. Palsson, Systems biology: simulation of dynamic network states, Cambridge University Press, 2011.

[49] M. H. Khammash, Perfect adaptation in biology, Cell Systems 12 (6) (2021) 509–521.

[50] Y. Yin, R. Xia, Y. Chen, R. Jia, N. Zhong, C. Yan, Q. Hu, X. Li, H. Zhang, Non-steady state fluctuations in water levels exacerbate long-term and seasonal degradation of water quality in river-connected lakes, Water Research 242 (2023) 120247.

[51] C. Kerschner, Economic de-growth vs. steady-state economy, Journal of cleaner production 18 (6) (2010) 544–551.

[52] H. E. Daly, Steady-state economics, in: Thinking about the environment, Routledge, 2015, pp. 250–255.

[53] L. Liu, D. Grombacher, M. Griffiths, M. Vang, J. J. Larsen, Signal processing steady-state surface nmr data, IEEE Transactions on Instrumentation and Measurement 72 (2023) 1–13.

[54] K. Pawlikowski, Steady-state simulation of queueing processes: survey of problems and solutions, ACM Computing Surveys (CSUR) 22 (2) (1990) 123–170.

[55] S. Narasimhan, R. Mah, A. Tamhane, J. Woodward, J. Hale, A composite statistical test for detecting changes of steady states, AIChE Journal 32 (9) (1986) 1409–1418.

[56] S. Narasimhan, C. S. Kao, R. Mah, Detecting changes of steady states using the mathematical theory of evidence, AIChE journal 33 (11) (1987) 1930–1932.

[57] J. D. Kelly, J. D. Hedengren, A steady-state detection (ssd) algorithm to detect non-stationary drifts in processes, Journal of Process Control 23 (3) (2013) 326–331.

[58] Ø. Ø. Dalheim, S. Steen, A computationally efficient method for identification of steady state in time series data from ship monitoring, Journal of Ocean Engineering and Science 5 (4) (2020) 333–345.

[59] T. Jiang, B. Chen, X. He, P. Stuart, Application of steady-state detection method based on wavelet transform, Computers & chemical engineering 27 (4) (2003) 569–578.

[60] F. Flehmig, R. Watzdorf, W. Marquardt, Identification of trends in process measurements using the wavelet transform, Computers & chemical engineering 22 (1998) S491–S496.

[61] M. Korbel, S. Bellec, T. Jiang, P. Stuart, Steady state identification for on-line data reconciliation based on wavelet transform and filtering, Computers & Chemical Engineering 63 (2014) 206–218.

[62] E. M. Turan, J. Jäschke, A simple two-parameter steady-state detection algorithm: Concept and experimental validation, in: Computer Aided Chemical Engineering, Vol. 52, Elsevier, 2023, pp. 1765–1770.

[63] S. Koermer, A. Noble, Estimation of process steady state with autoregressive models and bayesian inference, Minerals Engineering 191 (2023).

[64] Z. Wang, Y. Gu, A steady-state detection method based on gaussian discriminant analysis for the on-line gas turbine process, Applied Thermal Engineering 133 (2018) 1–7.

[65] S. Xie, C. Yang, X. Wang, X. Yuan, Y. Xie, A data-driven adaptive multivariate steady state detection strategy for the evaporation process of the sodium aluminate solution, Journal of Process Control 68 (2018) 145–159.

[66] P. Cao, J. Wang, C. Zhang, Steady-state interval detection and nonlinear modeling for automatic generation control systems, IEEE Access 7 (2019) 139592–139600.

[67] B. Esmael, A. Arnaout, R. K. Fruhwirth, G. Thonhauser, Multivariate time series classification by combining trend-based and value-based approximations, in: International Conference on Computational Science and Its Applications, Springer, 2012, pp. 392–403.

[68] M. Kim, S. H. Yoon, P. A. Domanski, W. V. Payne, Design of a steady-state detector for fault detection and diagnosis of a residential air conditioner, International journal of refrigeration 31 (5) (2008) 790–799.

[69] M. Boesler, N. Weber, Steady state detection for computational fluid dynamics, arXiv preprint arXiv:1709.03158 (2017).

[70] X. Qian, L. Xu, X. Cui, Steady-state detection of evaporation process based on multivariate data fusion, PloS one 19 (9) (2024) e0309652.

[71] Y. Wang, J.-P. Cassar, V. Cocquempot, A.-S. Guilbert, Selection of steady state time-periods for monitoring an industrial heat exchanger, in: International Conference on Diagnostics of Processes and Systems, Springer, 2017, pp. 368–379.

[72] H. Xu, Efficient approaches to steady state detection in multivariate systems, Ph.D. thesis, The University of Texas at El Paso (2022).

[73] J. Wu, H. Xu, C. Zhang, Y. Yuan, A sequential bayesian partitioning approach for online steady-state detection of multivariate systems, IEEE Transactions on Automation Science and Engineering 16 (4) (2019) 1882–1895.

[74] Y. Hou, J. Wu, Y. Chen, Online steady state detection based on rao-blackwellized sequential monte carlo, Quality and Reliability Engineering International 32 (8) (2016) 2667–2683.

[75] Y. Yao, C. Zhao, F. Gao, Batch-to-batch steady state identification based on variable correlation and mahalanobis distance, Industrial & engineering chemistry research 48 (24) (2009) 11060–11070.

[76] S. Cao, R. R. Rhinehart, An efficient method for on-line identification of steady state, Journal of Process Control 5 (6) (1995) 363–374.

[77] R. J. C. Bose, W. van der Aalst, I. Žliobaitė, M. Pechenizkiy, Handling concept drift in process mining, in: International Conference on Advanced Information Systems Engineering (CAiSE), Springer, 2011, pp. 391–405.

[78] M. Pourbafrani, N. Lücking, M. Lucke, W. van der Aalst, Steady state estimation for business process simulations, in: International Conference on Business Process Management (BPM), Springer, 2023, pp. 178–195.

[79] J. Ko, M. Comuzzi, A Systematic Review of Anomaly Detection for Business Process Event Logs, Business & Information Systems Engineering 65 (4) (2023) 441–462.

[80] D. C. Montgomery, Introduction to statistical quality control, 8th Edition, John Wiley & Sons, 2019.