Push your objects into streams!

Streaming OCPM / Take 1

Jeppe M. Mikkelsen[1], Andrey Rivkin[1], and Andrea Burattin[1]

DTU Compute, Technical University of Denmark
Kgs. Lyngby, Denmark
{s204708, ariv, andbur}@dtu.dk

Abstract. Object-Centric Process Mining (OCPM) addresses the limitations of traditional process mining by allowing events to relate to multiple object types, thus better reflecting the complexity of real-world systems. However, current OCPM techniques are limited to static, log-based inputs and do not support real-time analysis, which is increasingly becoming critical in dynamic business settings. This work introduces a novel framework for Streaming Object-Centric Process Mining (SOCPM), enabling the online discovery of object-centric process models. The framework supports the construction – in real-time – of Object-Centric Directly-Follows Graphs (OC-DFGs) and a relational structure that captures inter-object cardinalities from streaming data, for which we introduce a dedicated lossy counting miner. The framework is implemented using the open-source pyBeamline library, and it is validated on public datasets, demonstrating strong alignment with offline methods and capabilities to adapt to evolving behaviors.

Keywords: Streaming Process Mining, OCPM, Beamline

1 Introduction

Modern enterprise systems generate vast, continuous streams of event data that reflect complex, interconnected business operations. Traditional process mining approaches, which rely on static logs and a single-case perspective, struggle to capture the multi-object interactions inherent in systems such as ERP and CRM platforms. Object-Centric Process Mining (OCPM) addresses this limitation by modeling events as shared among multiple object types, offering a richer and more realistic view of business processes.

OCPM research has largely remained confined to offline settings, creating a gap between the theoretical potential of object-centric analysis and the real-time demands of contemporary business environments. Streaming Object-Centric Process Mining (SOCPM) bridges this gap by enabling continuous, low-latency discovery of object-centric process models from live data streams. This paradigm not only ensures timely insights and adaptability to evolving behaviors (such as concept-drift) but also aligns naturally with the structure of object-centric data, where events co-occur across multiple entities. Therefore, adapting OCPM techniques to the streaming settings is not only timely but essential for operational scalability, responsiveness, and decision-making in data-intensive systems.

One of the most prominent methodological workflows for conducting OCPM discovery specifically aimed at generating E2E and E2O process models [1]. E2E refers to models that capture relationships, such as the "directly-follows" relationship between event types (i.e., activities), while E2O refers to models that represent relationships between event types and object types, such as being able to derive object type from an arc (example Fig. 1). The workflow reflects a standard step-by-step pipeline used to transform a static object-centric event log to a system-wide process object-centric process model capable of capturing complex multi-object interactions [2,3,1,4]. The steps are: *(i)* (Input) Given a static object-centric event log that adheres to the correct specifications, where each event is associated with one or more object identifiers of potentially diverse object types (e.g., `Order`, `Package`, `Item`) as illustrated in Tab. 1. *(ii)* (Flatten Events) Convert each multi-object event into a set of flattened events, each tied to a single object identifier. This effectively "clones" the original event per object identifier, ensuring that only one identifier is retained per instance. *(iii)* (Projection) Partition the flattened events into separate logs, one for each object type. This isolates the behaviour of each object class, such as activities specifically involving `Orders` or `Items` *(iv)* (Mining) Apply traditional process mining discovery techniques independently to each object-type-specific log. Since object identifiers are unique, they can serve as the traditional version of case identifiers, enabling the construction of process models that capture the lifecycle of each object identifier of a specific object type. *(v)* (Merge) Combine the individually discovered models by aligning activities shared across object types and incorporating E2E and E2O interactions. This results in a cohesive object-centric process model that captures the integrated behaviour of the entire system across multiple interacting entities.

Ultimately, this workflow yields an OCPM process model, potentially resembling the structure of an OC-DFG illustrated in Figure 1. This method represents the prevailing approach in object-centric process analysis [1]. This paper adapts the given framework to work in a streaming setting, i.e., instead of consuming a finite log, it connects to a never-ending event stream and produces a never-ending sequence of updates for the models, allowing real-time analysis of what is currently happening.

The rest of the paper is structured as follows: Sect. 2 describes the preliminaries for object-centric streams; Sect. 3 presents the proposed framework; Sect. 4 shows some experimental results; and Sect. 5 concludes the paper.

2 Object-Centric Streams

Let us fix the following pairwise disjoint sets: *(i)* $\mathcal{E}$ the countably infinite set of event identifiers; *(ii)* $\mathcal{ET}$ the countably infinite set of event types (i.e., activities); *(iii)* $\mathcal{T}$ the infinite set of timestamps; *(iv)* $\mathcal{OT}$ the countably infinite set of object types; *(v)* $\mathcal{I}$ the infinite set of object identifiers.

In this work, we consider simplified versions of object-centric events, in which we focus only on event identifiers, timestamps, activities, and involved objects.

Using such events, one can then define two types of relationships: E2O (between events and objects) and O2O (between objects and objects). This definition aligns with the format adopted in OCEL 1.0 [5], thereby allowing for the application of already existing object-centric discovery techniques [1].

Definition 1 (OC-event). *An* object-centric event *(OC-event) for short) is a tuple* $(e, act, t, omap)$*, where: (1)* $e \in \mathcal{E}$ *is the event's id; (2)* $act \in \mathcal{ET}$ *is the activity name; (3)* $t \in \mathcal{T}$ *is the event's timestamp; (4)* $omap : \mathcal{OT} \nrightarrow (\mathcal{P}(\mathcal{I}) \setminus \{\emptyset\})$ *is a partial function indicating which objects participate to event* e*.*

From each OC-event, it is possible to extract the associated object types and identifiers, which are required to enable any form of OCPM (in which an object-centric event log is just a set of object-centric events), including approaches analogous to offline process discovery [4,1]. Hereinafter, we use *OCevents* to denote the set of all OC-events, and for $o \in OCevents$ use projection functions π_e, π_{act}, π_t and π_{omap} to access the event's components.

In the context of streaming OCPM, we focus on a setting in which no post-mortem data are provided. Instead, as customary to large-scale enterprise information systems, events (along with the objects involved) are being continuously logged in a stream. To support this setting, it is essential to define the nature of the data being processed. We thus define object-centric streams (OC-streams for short) as an infinite, unbounded sequence of observable units $\mathcal{O} = \langle o_1, o_2, \ldots \rangle$ that evolves through their continuous addition [6,7]. Each observable unit o_i must be an OC-event. To align with the traditional temporal semantics of event streams, where events are emitted upon occurrence in real time [7,8], we assume that a timestamp associated with each OC-event corresponds exactly to the moment in which the event takes place in the real world.

Definition 2 (OC-stream). *Let* $\mathcal{O} \subseteq OCevents$ *be the set of observable units. An* object-centric event stream *is an infinite sequence* $S : \mathbb{N}_0 \to \mathcal{O}$*, which respects the temporal ordering property:* $i < j$ $(i, j \in \mathbb{N}_0)$ *iff* $\pi_t(S(i)) \leq \pi_t(S(j))$*.*

The following is an OC-event with the identifier `e1`, which corresponds to the activity `"Place Order"` which appears in an OC-stream:

$$o = (\texttt{e1}, \texttt{Place Order}, \{\texttt{Order} : \{o1\},\ \texttt{Item} : \{i1, i2\}\})$$

This event involves multiple object types and the respective identifiers of involved objects. The timestamp is omitted assuming that it is implicitly assigned at the event emission into the stream.

2.1 Models for Object-Centric Data Representation

A fundamental process model relevant to this work is the Object-Centric Directly-Follow Graph (OC-DFG) [9,10]. In the nutshell, as opposed to standard DFGs, OC-DFGs allow to link activities across multiple interacting object types. Like that, in an OC-DFG, an event activity may be associated with several objects by their co-occurrence (e.g., in Table 1, activity "Place Order" contains objects of types `Order` and `Item`), and the graph reflects relations between activities through the lens of these object relations (via typed arcs).

Table 1: Snippet of an object-centric stream

Event ID	Activity	Timestamp	Order	Item	Package
e1	Place Order	2025-05-10T09:05:00	o1	i1, i2	∅
e2	Prepare Package	2025-05-10T09:30:00	o1	i1, i2	p1
e3	Reserve Item	2025-05-10T09:32:00	∅	i1, i2	∅
e4	Pack Item	2025-05-10T11:40:00	o1	i1, i2	p1
e5	Ship Package	2025-06-10T09:25:00	∅	i1, i2	p1
e6	Send Invoice	2025-06-10T11:30:00	o1	∅	∅
e7	Receive Review	2025-10-10T15:35:00	o1	∅	∅
...	...	...	...	...	...

Definition 3 (OC-DFG [9]). *An* object-centric directly-follows graph *(OC-DFG for short) is a tuple* $(A, OT, N, F, \pi_{freqn}, \pi_{freqe})$ *where:*

- A *and* OT *are respectively sets of activities and object types;*
- $N = A \cup \{n_{S,ot} \mid ot \in OT\} \cup \{n_{E,ot} \mid ot \in OT\}$ *is the set of nodes, where* $n_{S,ot}$ *and* $n_{E,ot}$ *are start and end nodes for each object type* ot*, respectively;*
- $F \subseteq N \times OT \times N$ *is the set of typed arcs.*
- $\pi_{freqn} : A \nrightarrow \mathbb{N}$ *assigns a frequency to the activities.*
- $\pi_{freqe} : F \nrightarrow \mathbb{R}_{\geq 0}$ *assigns a frequency to the arcs.*

OC-DFGs can provide valuable insights by highlighting cross-object behavioural patterns, thus enabling analysts to uncover interdependencies and coordination points between different object lifecycles [9]. However, OC-DFGs do not capture such control-flow structures as AND/XOR splits and joins, which in turn limits their expressiveness in complex process scenarios. Despite this, they remain a valuable exploratory tool in the object-centric process mining toolkit.

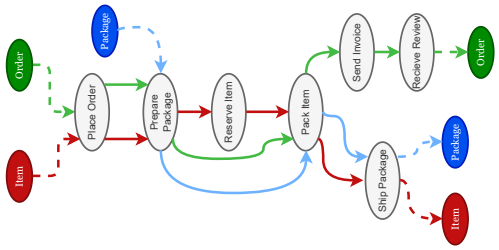


Fig. 1: Simplified OC-DFG based on the stream from Table. 1

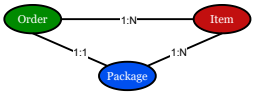


Fig. 2: Object Occurence Model (OOM) projected on activity "Pack Item"

Figure 1 shows a simple OC-DFG, with the frequency values of activities and arcs left out for simplicity. The graph features "dummy" start/end nodes for all the types (represented by dashed lines with ingoing/outgoing arrows). By following the colour-coded (typed) arcs, one can trace the behavioural flow specific to each object type. For example, the activity `"Send Invoice"` is observed only after `"Pack Item"`, and requires objects of type `Order` to be executed.

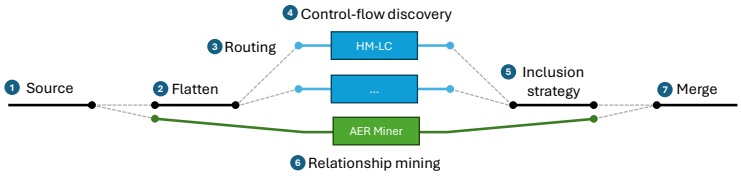


Fig. 3: Architectural overview of S-OCPM, with individual steps are numbered.

To complement the OC-DFG model and account for missing information about (O2O) relations between concrete objects, we introduce an auxiliary structure that approximates relational semantics between activities and object types based on co-occurrence observations happening at the level of OC-events. Such structure can encode both unary participations (i.e., activities linked to a single object type) and binary relations (i.e., co-participation of multiple object types), where the latter are qualified with observed cardinalities.

Definition 4 (OOM). *The* Object Occurrence Model *(OOM for short) is a tuple* (A, E, R)*, where:* $A \subset \mathcal{ET}$ *is a finite set of activities;* $E : A \to \mathcal{P}(\mathcal{OT})$ *is a mapping that assigns to each activity from* A *a set of associated object types;* $R \subseteq A \times \mathcal{OT} \times \mathcal{OT} \times \mathcal{C}$ *is the ternary relation linking activities with pairs of involved object types and a cardinality qualifier from* $\mathcal{C}$*, where* $\mathcal{C} = \{1{:}1, 1{:}N, N{:}M\}$.

It is easy to see that elements of $\mathcal{C}$ represent standard cardinality constraints: one-to-one, one-to-many and many-to-many.

In OOMs, observed co-occurrence relations are considered per activity. This enables the model to reflect object type relations based solely on their joint appearances within individual activities. For illustration, consider the activity "Pack Item" from Table 1. The projection of this activity within the OOM, based on the observed object types and their identifiers, is depicted in Figure 2. The associations are derived from actual co-occurrence observations. Consequently, if a variant of the trace involves different object types that do not appear together within the same instance of an activity, no association is formed. This provides a complementary perspective to the OC-DFG, which does not distinguish between object types that share activities but may not be jointly involved in the same event instance.

3 Streaming Object-Centric Process Mining Framework

This section introduces the general streaming object-centric process mining (SOCPM) framework that enables real-time discovery of object-centric process models from object-centric event streams. The proposed pipeline architecture is presented in Fig. 3. Each step is numbered and described briefly below:

1. **Source**: this step continuously emits OC-events in an OC-stream (cf., Def. 2);
2. **Flatten**: this step transforms multi-object events into a series of single-object events (i.e., flattening), preserving the object-centric semantics while facilitating projection;

3. **Routing**: in here, events are filtered and flattened into object-type specific substreams, enabling the per-type object-centric model discovery;
4. **Control-flow discovery**: this step applies existing streaming control-flow discovery techniques (e.g., Heuristics Miner with Lossy Counting) to each object-type stream to build and update a model for each object type;
5. **Inclusion strategy**: this step decides which object types are deemed ACTIVE (based on recency and frequency) and thus to be included in the final model, to enable the framework to gracefully handle concept drift;
6. **Relationship mining**: this step goes beyond the control-flow discovery and focuses on capturing inter-object cardinalities using a new mining algorithm;
7. **Merging**: in this final step, all the type-specific control-flows are merged into an OC-DFG representation together with the OOM.

We describe each of the steps in more detail below. **Step 1** of the SOCPM framework relies on the concept of object-centric streams formally introduced in Def. 2. It is also important to mention that the emitted events are formatted according to the OCEL 1.0 standard [5], thus ensuring compatibility with standard object-type activity mappings and interoperability with existing off-line OCPM techniques. Since a single OC-event, as per Def. 1, may refer to several object types and multiple object identifiers per type, an event cannot be directly attributed to a single object type without ambiguity in the context of type-specific mining. **Step 2** aims at addressing this challenge by re-introducing a new operational notion of *flattening*, inspired by [11,1,10]. Here, flattening deterministically transforms each OC-event into a set of simplified, type-specific event representations. Specifically, given $o = (e_{id}, act, t, omap) \in OCevents$, its flattening is defined as $\mathsf{flatten}(o) = \{(e_{id}, act, t, ot \mapsto \{o_{id}\}) \mid ot \in \mathcal{OT}, o_{id} \in omap(ot)\}$. It is easy to see that flattened set of events consists of OC-events in which object maps have singleton co-domains. Notice that the flattening operation is information-preserving, i.e., the original event structure can be reconstructed through a composition operation over the flattened events. Such a composition operator can be easily defined for flattened OC-events that share the same event id, activity, and timestamp. From the framework's perspective, the flattening-composition pair thus forms a lossless and reversible transformation, ensuring that the object-centric structure of the events remains intact.

Step 3 defines an operation that is similar to the traditional type-based log projection in OCPM (used for filtering events based on the object types, enabling per-type process analysis [4]). In the streaming setting, we replicate this selective isolation of object-type-specific behaviour through a routing mechanism that supports dynamic and selective discovery. The proposed routing component supports two modes of operation, allowing for the control of which object types are to be mined. In *static routing*, a routing configuration is provided as a set $R \subseteq \mathcal{OT}$ of object types. The framework then pre-registers all sub-OC-streams for all types in R, meaning only events associated with the object types in R are projected into type-specific OC-streams; all events of other types are discarded/suppressed. This gives analysts explicit control to focus on selected process views. If no such configuration is supplied, the framework uses *dynamic rout-*

ing, where all object types encountered in the OC-stream are considered. Hence, if a new object type is encountered for the first time, a new sub-OC-stream will be instantiated. This mode supports broad discovery without prior assumptions, enabling comprehensive modelling of the system's behaviour across all object types when no domain knowledge is available. The routing mechanism situated at the (fan-out) position in the architectural overview in Fig. 3 constitutes individual sub-OC-streams based per object type. To account for such streams, we define *ot-type projected streams* as $S_{ot} : \mathbb{N}_0 \to \mathcal{O}|_{ot}$ (where $\mathcal{O}|_{ot} = \{o \in \mathcal{O} \mid \mathcal{O} \subseteq OCevents, \text{DOM}(omap_o) = \{ot\}, |omap_o| = 1\}$)[1], which can be seen as filtered, time-preserving views of the fan-out sub-OC-stream, containing only events relevant to a specific object type $ot \in \mathcal{OT}$. These streams form the foundation for per-type online process discovery and align with the modularity principles of multistep offline approaches [1,11].

In **Step 4**, we continue adapting OCPM into the streaming setting. In particular, the step focuses on discovering control-flow models for each *ot*-type projected stream obtained in the previous step, and does that by satisfying key criteria for streaming environments such as time and memory efficiency, single-pass processing, real-time responsiveness, and adaptability to evolving behaviour [7,6,8]. The aforementioned criteria, for example, are satisfied by *Heuristics Miner with Lossy Counting* [6] and sliding window techniques. These techniques can be effectively adapted to object-centric settings by replacing the traditional case identifier with object identifiers from flattened OC-events.

Step 5 analyses the models emitted by individual stream miners and provides a mechanism for regulating which of the related object types should contribute to the global model over time. This becomes critical in the presence of object-type drift or when selective filtering is required to see specific views on the process. To relevance analyses is performed using an inclusion strategy that defines observer logic governing whether an object type $ot \in \mathcal{OT}$ is considered ACTIVE (its model is to be incorporated into the merged global process model), or INACTIVE (its model is to be excluded). The observer is situated on the (fan-in) position, yielding an aggregated stream of the individual projected sub-OC-streams. This design, in turn, facilitates cross-type comparison, enabling the identification of inconsistencies, redundancies, or drift across interconnected object behaviours.

The inclusion strategy observer is inherently governed by the update frequencies of the individual stream miners. This introduces a structural limitation, as the system's responsiveness and granularity are tightly coupled with each miner's configuration. Nevertheless, this design also promotes the incorporation of domain knowledge, empowering analysts to tailor the system's adaptivity. Three well-established inclusion strategies from the data stream processing are considered in our framework: *(i) Relative Frequency*: Retain object types in the global model based on how frequently their corresponding miners emit updates; frequently observed object types reflect dominant, ongoing behaviours within the system. *(i) Sliding Window*: Retain object types that have emitted models within a temporal window of a fixed size; fixed-size windows are suitable for processes

[1] Here, given a function $f : X \to Y$, $\text{DOM}(f) = X$ represents the domain of f.

with changing stages. *(i) Lossy Counting*: Using the lossy counting approximation technique, track object type update frequencies using bounded memory. Object types with low support (i.e., infrequent model updates) are gradually discarded, which allows to strike a balance between accuracy and scalability, offering robustness in high-volume streaming environments by ensuring that only statistically significant patterns persist. These inclusion strategies offer complementary trade-offs between adaptability, interpretability, and computational overhead. While the two previous steps contribute to an eventual construction of a single OC-DFG model representing all relevant lifecycles (and their interactions), a known limitation of OC-DFGs is their inability to distinguish whether the associated object types, linked to an activity, actually co-occurred in the same event. This creates an ambiguity in interpretation: for example, if two object types have separate relations to the same activity label across different traces, the OC-DFG cannot determine whether the object types appeared together within the same event, or if they merely share the same activity independently in separate contexts. To address this limitation, at **Step 6** we use a parallel stream miner and a corresponding model, designed to infer object-type association relationships based on observed co-occurrences within the same activity of an event. The miner operates on the raw, unflattened object-centric event stream (see Def. 2), which essentially removes co-participation information. The proposed miner observes how object types are jointly involved in activities, and through object identifier manipulation and a simple cardinality heuristic, it approximates structural associations such as one-to-one, one-to-many, and many-to-many. In a nutshell, the miner maintains, for each activity, a record of the object types that have been observed in conjunction with it. The co-occurrence relationships are tracked using a separate *Lossy Counting* structure per activity, which monitors the observed cardinalities based on which object types appear together. Once an activity-specific bucket threshold is reached, the algorithm prunes infrequent cardinality patterns within that activity, thereby retaining only those associations that reflect frequent and stable behavioural tendencies. This process ensures a memory-efficient representation of the most relevant object-type relations. It then outputs an OOM which, for each activity, reflects all observed object types and their most frequently observed cardinality associations relationships. These associations are interpreted as undirected and non-hierarchical, reflecting the data's lack of explicit O2O semantics. Nonetheless, the inferred structure aims at complementing the behavioural lens of the OC-DFG with an observed co-occurrence relational perspective, which would allow analysts to detect separate traces when no object types are linked, and enable concept drift detection within observed relations constructed by the frequent co-occurrence semantics of object types and their identifiers.

At **Step 7**, the merging operation is performed. The operation is placed at the end of the architecture pipeline (see Fig. 3) so that it integrates the outputs of individual object-type-specific substream miners yielding DFGs (**Step 4**), information collected by dynamic inclusion strategies (**Step 5**), and the OOM (**Step 6**) into a unified representation of the object-centric process. The end

result is a comprehensive, system-wide OC-DFG supported by an aligned OOM, ensuring consistency between behavioural and structural perspectives.

The merge step maintains: *(i)* the most recent DFGs produced by sub-OC-stream miners (per object type); *(ii)* the currently ACTIVE object types, filtered by the chosen inclusion strategy; *(iii)* the most recent OOM reflecting maximum co-occurrence cardinality relationships between object types per activity.

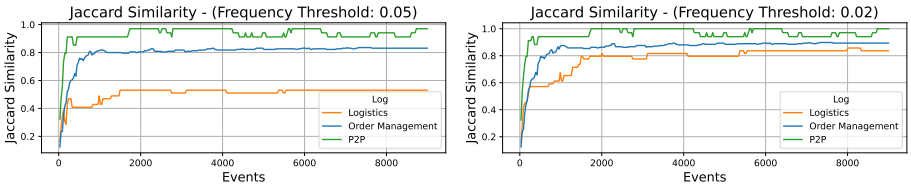
The merging approach used for constructing the OC-DFG models is very similar to the one from [9,10]. The step also inherits the simplifying assumption that activity labels are globally unique across object types [12,5], which allows to merge activity nodes solely based on their labels [10,3,9]. The key difference in our merge approach lies in the data source for model composition. In this framework, the individual DFGs are continuously emitted from type-specific stream miners, and the set of ACTIVE object types is dynamically maintained based on the selected inclusion strategy. On top of that, streaming constraints such as bounded memory, approximate frequency tracking (e.g., Lossy Counting), and dynamic inclusion strategies inherently limit the process model's completeness. Furthermore, due to the inherent limitations arising from the fundamental issues of *convergence* and *divergence* in object-centric event data [3,1], together with the constraint of the availability of results of the individual miners (i.e., each miner has its own frequency updates, and thus we cannot guarantee synchronization between edge frequency updates and the underlying activity frequencies across object types), it is not possible to associate frequencies with individual activities in the global model reliably. We thus generate restricted OC-DFGs in which frequencies of activities π_{freqn} (see Def. 3) are omitted.

The generated (live) OC-DFG reflects the most recent behavioural state of the system. Our framework also ensures that all the behavioural relations discovered by each stream miner are retained in the merged output, regardless of the specific inclusion strategy employed. This means that the stream-derived model remains a proper subset both structurally and behaviourally of the log-derived model, and it adheres to the constraints of sound approximation.

Lastly, the final step modifies the previously obtained OOM in order to align it with the recent version of the behavioural model (i.e., the OC-DFG). Since the produced OC-DFG includes only a subset of object types and activities, limited by inclusion strategies and streaming constraints, this is needed to account for the fact that OOM may reference entities that are no longer behaviourally relevant. The aforementioned adjustment can be easily realised by traversing the OC-DFG structure and pruning relations involving activities and object types that do not appear therein.

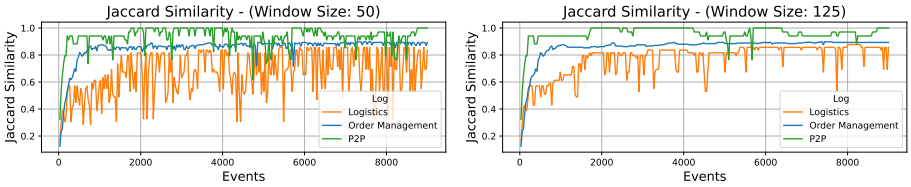
4 Experimental Evaluation

The goal of this evaluation is to assess the framework's capability in reconstructing behavioural models and to benchmark its performance against offline approaches, namely PM4Py OC-DFG discovery [13]. The approach has been implemented in pyBeamline [14] and is available online at `https://github.`



(a) Frequency threshold of 0.05 (b) Frequency threshold of 0.02

Fig. 4: Jaccard similarity wrt offline OC-DFG discovery as a function of events processed, with inclusion strategy *Relative Frequency* with different thresholds



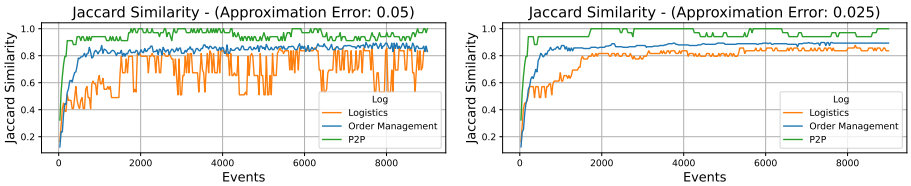
(a) Window size of 50 (b) Window size of 125

Fig. 5: Jaccard similarity wrt offline OC-DFG discovery as a function of events processed, with inclusion strategy *Sliding Window* with different thresholds

`com/beamline/pybeamline`. The comparison measures the similarity between the edge sets F defined in Def. 3. For this, the *Jaccard Similarity* metric is employed, computed over time to observe how the discovered OC-DFG evolves.

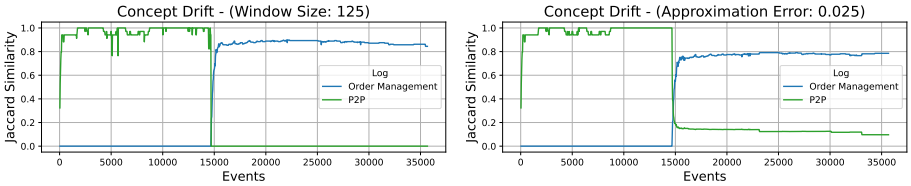
Datasets. To evaluate the proposed streaming framework, three multi-object process logs are used, each representing distinct real-world scenarios. The *Logistics* log is a synthetic dataset derived from a simulation of a logistics process. It includes 7 object types and 14 event classes, resulting in a total of 35 761 events [15]. The *Order Management* log captures the lifecycle of customer order processing. It contains 6 object types, 11 event types, and a total of 21 008 events [16]. Finally, the *Procure-to-Pay (P2P)* log reflects a complete procure-to-pay process within an organizational setting. This dataset comprises 7 object types, 10 event types, and 14 671 events [17].

Results. The framework has been configured to use *dynamic routing*, with default miner of *Heuristics Miner with Lossy Counting* with $\epsilon = 0.001$. Therefore, all object types are treated uniformly. For readability, plots have been truncated at 9 000 processed events, beyond which the model tends to stabilise. Fig. 4 demonstrates the framework's capacity to capture behaviour early in the stream for strategy *Relative Frequency*. For *P2P*, near-perfect similarity is reached within 2000 events. In contrast, *Logistics* initially stagnates around 0.5 similarity, as shown in subfigure (a). Reducing the threshold to 0.02 improves performance across all datasets. These results suggests that tuning inclusion parameters directly impacts structural completeness and discovery quality. Fig. 5



(a) Approximation error of 0.05 (b) Approximation error of 0.025

Fig. 6: Jaccard similarity wrt offline OC-DFG discovery as a function of events processed, with inclusion strategy *Lossy Counting* with different thresholds



(a) Sliding Window ($W = 125$). (b) Lossy Counting ($\varepsilon = 0.025$).

Fig. 7: Jaccard similarity under concept drift conditions over events.

shows that peak Jaccard similarities using *Sliding Window* are comparable to *Relative Frequency*. Smaller windows introduce fluctuations due to object types intermittently entering/exiting the window. Fig. 6 shows stable model quality with moderate Jaccard similarity. *P2P* and *Order Management* maintain high accuracy, while *Logistics* shows more variability. The approximation error allows frequent object types to remain ACTIVE, filtering out less significant ones.

Concept Drift. To evaluate the framework's responsiveness to object-centric concept drift, an experiment was conducted by merging two event logs: *P2P* and *Order Management*. *Sliding Window* and *Lossy Counting*, were selected for testing. Fig. 7(a) shows that *Sliding Window* discards outdated *P2P* patterns and quickly adjusts to *Order Management* behaviour. This demonstrates its strength in reacting to sharp behavioural transitions. The OC-DFG structure dynamically shifts to reflect new dominant object-type relations. Fig. 7(b) illustrates *Lossy Counting*'s performance. Although it eventually adapts, remnants of the initial *P2P* log persist for longer periods, due to the reliance on approximate frequency thresholds, delaying model transition. Both strategies demonstrate adaptability, but *Sliding Window* is more responsive to abrupt shifts, and *Lossy Counting* is more conservative and may offer robustness in scenarios with gradual or noisy drift, confirming the framework's capacity to detect and react to concept drift.

5 Conclusions and Future Work

This work presents a novel framework for Streaming OCPM (SOCPM). By adapting object-centric discovery techniques to a streaming context, the framework enables real-time analysis of complex, multi-object business processes. Empirical results demonstrate that the framework retains high alignment with traditional offline discovery while offering responsiveness to concept drift. These contributions lay the foundation for a new class of process mining solutions, closing the gap towards real-time and system-wide operational intelligence.

Future work will extend the framework to support more expressive models and incorporate richer semantics from OCEL 2.0. Additionally, integrating real-time anomaly detection, predictive analytics, and conformance checking would further enhance the framework's applicability.

References

1. Berti, A., Montali, M., van der Aalst, W.M.: Advancements and challenges in object-centric process mining: A systematic literature review. arXiv (2023)
2. van der Aalst, W.M., Berti, A.: Discovering object-centric petri nets. Fundamenta informaticae **175**(1-4) (2020) 1–40
3. van der Aalst, W.M.P.: Object-centric process mining: Unraveling the fabric of real processes. Mathematics **11**(12) (2023)
4. van der Aalst, W.M.P. In: Object-Centric Process Mining: An Introduction. Springer International Publishing, Cham (2023) 73–105
5. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.: Ocel standard. Process and Data Science Group, RWTH Aachen University, techreport **1** (2020)
6. Burattin, A.: Streaming process mining. Process Mining Handbook (2022) 3–10
7. Aggarwal, C.C.: Data streams: models and algorithms. Springer (2007)
8. Burattin, A. In: Streaming Process Discovery and Conformance Checking. Springer International Publishing, Cham (2018) 1–8
9. Berti, A., van der Aalst, W.M.: Oc-pm: analyzing object-centric event logs and process models. International Journal on STTT **25**(1) (2023) 1–17
10. van der Aalst, W.M.: Object-centric process mining: dealing with divergence and convergence in event data. In: Proceedings of SEFM 2019, Springer (2019) 3–25
11. van der Aalst, W.M.P., Berti, A.: Discovering object-centric petri nets (2020)
12. Berti, A., Koren, I., Adams, J.N., Park, G., Knopp, B., Graves, N., Rafiei, M., Liß, L., et al.: OCEL (Object-Centric Event Log) 2.0 Specification (2024)
13. Berti, A., van Zelst, S., Schuster, D.: PM4Py: A process mining library for Python. Software Impacts **17** (2023) 100556
14. Burattin, A.: Beamline: A comprehensive toolkit for research and development of streaming process mining. Software Impacts **17** (2023)
15. Knopp, B., Graves, N.: Container logistics object-centric event log (August 2023)
16. Knopp, B., van der Aalst, W.M.: Order management object-centric event log in ocel 2.0 standard (September 2023)
17. Park, G., Leah Tacke, g.U.: Procure-to-payment (p2p) object-centric event log in ocel 2.0 standard (October 2023)