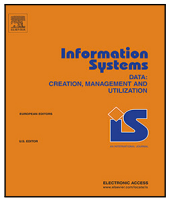
Information Systems 124 (2024) 102404

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



Recognizing task-level events from user interaction data

Adrian Rebmann [a,*], Han van der Aa [b]

[a] Data and Web Science Group, University of Mannheim, B6 26, 68159 Mannheim, Germany
[b] Faculty of Computer Science, University of Vienna, Währinger Str. 29, 1090 Vienna, Austria

ARTICLE INFO

Keywords:
User interaction data
Event abstraction
Process mining
Streaming process mining

ABSTRACT

User interaction data comprises events that capture individual actions that a user performs on their computer. Such events provide detailed records about how users carry out their tasks in a process, even when this involves different applications. Although the comprehensiveness of such data provides a promising basis for process mining, user interaction events cannot be used directly for this purpose, because they do not meet two essential requirements. In particular, they neither indicate their relation to a process-level activity nor their relation to a specific process execution. Therefore, user interaction data needs to be transformed so that it meets these requirements before process mining techniques can be applied. This transformation problem comprises identifying tasks and their types and determining the relation between tasks and process executions. While some existing approaches tackle parts of this problem, none address it comprehensively. Therefore, we propose an unsupervised approach for recognizing task-level events from user interaction data that addresses it in full. It segments user interaction data to identify tasks, categorizes these according to their type, and relates tasks to each other via object instances it extracts from the user interaction events. In this manner, our approach creates task-level events that meet the requirements of process mining settings. Our evaluation demonstrates the approach's efficacy and shows that its combined consideration of control-flow, data, and semantic information allows it to outperform baseline approaches in both online and offline settings.

1. Introduction

User interaction data consists of events that capture a user's actions when performing tasks in a process at a fine-granular level. Each user interaction event corresponds to a single interaction between the user and the user interface of a software application, such as clicking a button, entering a text into a field, or ticking a checkbox [1,2]. In the context of process mining, a field concerned with the data-driven analysis of organizational processes [3], user interaction data provides a lot of potential, since it records events across applications, at a detailed level, and without the need to extract or integrate data from heterogeneous systems. Compared to traditional event data used in process mining, which typically comprises events recorded by a single information system, user interaction data can provide a more complete picture of the tasks performed by a user. Specifically, because user interaction data captures task executions that involve multiple applications, e.g., email, spreadsheets, and web applications, it plays a crucial role in addressing gaps within the process coverage provided by traditional event data. This extended process coverage, in turn, enables organizations to obtain insights that more accurately show how their process is really executed, e.g., through more comprehensive models after applying process discovery techniques [3].

However, user interaction events are unsuitable to be directly used for process mining, because they do not meet two essential requirements of event data in such settings. First, user interaction events do not indicate their relation to a process-level activity. Consequently, when analyzing a process using such events, obtained insights will show how a user interacted with their applications, rather than show how the process was executed. For instance, in the context of an order-handling process, applying process mining to user interaction events would result in insights such as *input text* is commonly followed by *click button*, instead of insights such as *create order* is commonly followed by *update order*. Second, user interaction events do not relate to specific process executions, which means that the relation between different process steps is not captured. For example, having identified a number of process steps involving the handling of orders, it is crucial to understand which of these steps relate to the same customer order and which to different ones. Therefore, to enable process mining on the basis of user interaction data, the data must be transformed so that it meets the requirements of events in process mining settings (which we refer to as *task-level events*). This transformation problem involves identifying (1) which events jointly form tasks of certain types (e.g., creating an order or updating an order quantity) and (2) which of

* Corresponding author.
E-mail addresses: rebmann@uni-mannheim.de (A. Rebmann), han.van.der.aa@univie.ac.at (H. van der Aa).

https://doi.org/10.1016/j.is.2024.102404
Received 23 October 2023; Received in revised form 26 April 2024; Accepted 12 May 2024
Available online 15 May 2024
0306-4379/© 2024 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

these identified tasks relate to the same process execution (e.g., that a particular order is created, after which its order quantity was updated).

This problem is partially addressed in *robotic process mining* (RPM) [2], where the main purpose is to identify automatable tasks in user interaction data. To this end, several supervised and unsupervised approaches have been recently proposed. Supervised approaches require task models as input [4,5] or labeled user interaction data for training [6] in order to identify tasks, whereas unsupervised approaches [7,8] avoid such requirements. In particular, the unsupervised approach by Leno et al. [7] segments user interaction data to identify tasks and subsequently groups frequent tasks into so-called routines. These routines can be seen as different types of tasks. Similarly, the approach by Urabe et al. [8] first segments user interaction data into tasks, which it then clusters into types. However, these existing works on RPM neither recognize a task's process-level type nor its relation to a process execution. Therefore, they only partially solve part (1) of the transformation problem and do not address part (2) at all.

Against this background, we use this paper to propose an unsupervised approach for recognizing task-level events that addresses both parts of the problem. It segments user interaction data to identify tasks, categorizes these according to their type, and relates tasks to each other via object instances it extracts from the low-level events. In this manner, our approach creates task-level events that meet the requirements of process mining settings. In addition to being the first approach that provides an end-to-end solution to this transformation problem, it has two further benefits over existing approaches:

- *Works in online settings:* Unlike the existing approaches, which target offline analysis based on stored user interaction logs, our approach can also deal with online streams of user interaction events. This has several benefits. First, our approach emits task-level events, which can directly be used as input for streaming process mining techniques [9]. These, in turn, can provide a timely understanding of current process behavior and enable process monitoring as well as predictions on-the-fly. Second, online processing allows our approach to handle unseen data without having to retrain its components from scratch. For instance, if a new task is introduced into a process, our approach can recognize this without the need to retrain it on historic data. Third, online processing avoids the need to store large amounts of low-level events that are not relevant from a process analysis perspective, such as a user logging into an SAP system, or that contain personal information, such as visits to news websites or private communication.
- *Improved accuracy:* Our approach can more accurately identify and categorize tasks in user interaction data in comparison to the existing approaches, because it considers a combination of control-flow, data, and semantic information, whereas the existing approaches only consider control-flow information. Our experiments demonstrate that these accuracy benefits apply to both online and offline settings. Therefore, our approach tackles the data transformation problem in a more comprehensive and more accurate manner than existing works.

Note that this paper is an extended and revised version of our earlier work on recognizing task-level events from user interaction streams [10]. The work presented here extends its previous version in three main ways: First, we broadened the *scope* of our approach by adding a component that extracts object-instance information that allows it to relate identified tasks and, thus, task-level events to process executions. Second, we improved the *accuracy* of our approach by expanding our task-identification component with an additional segmentation check, which allows it to better identify groups of related interaction events. Finally, we conducted additional evaluation experiments to assess the aforementioned extensions of our approach and to also demonstrate its applicability and accuracy in offline settings, i.e., when user interaction logs rather than streams are available.

The remainder of this paper is structured as follows. Section 2 illustrates the challenges of recognizing task-level events from user interaction data and Section 3 defines necessary concepts. Section 4 presents our approach, which we evaluate in Section 5. Finally, Section 6 summarizes related work and Section 7 discusses limitations and concludes.

2. Problem illustration

In this section, we first describe the two main parts involved in recognizing task-level events from user interaction data, before highlighting the additional challenges of doing this in a streaming setting.

Recognizing task-level events. To illustrate the problem of recognizing task-level events in an unsupervised manner, consider the excerpt of event data in Table 1, where the events record how a user handles requests related to orders. Although the user interaction events show what a user does at a detailed level (e.g., which buttons are clicked), it fails to give clear information about the actual process that is executed. In particular, the user interaction events neither make their relation to process-relevant tasks nor to specific process executions explicit. For instance, they do not indicate that *u1–u8* correspond to the execution of a particular task, i.e., creating order *O007501*, and *u9–u16* to a different one, i.e., updating order *O008102* after a change request was made. Identifying these relations involves the following two parts:

1. *Identify tasks and their types.*The first part involves identifying groups of user interaction events that jointly form tasks and their types. This is typically done in a two-step manner:
   (1) We need to find sequences of user interaction events that together form individual tasks, referred to as *segmentation* in the context of robotic process mining [2,8]. Working under the assumption that a user performs one task before moving to the next, this involves the identification of points in the data where one task completes and the next one starts. In the given example, this is the case after events *u8* and *u16*, which denote the completion of two higher-level tasks. The difficulty here is that such end points are not explicitly indicated in the data. For instance, although *u8* ends the first task by the press of a *Save order* button, event *u13* involves such a button as well, even though it occurs only halfway through the execution of the second task. As such, this requires identifying when execution has moved to the next task, based on clues from the context and attributes of events.
   (2) Having identified individual tasks, we aim to recognize which tasks correspond to the same type (e.g., creating an order), and which to different ones. However, variability makes this difficult, since the same process-level task may be executed by performing different sequences of user interaction events. For instance, the *create order* task (*u1–u8*) could also be executed without first logging in (*u2–u4*) or by having to search multiple times (*u6*) until the right customer is found.
2. *Identify task relations.* The second part is to identify the relations of tasks to process executions. To identify such relations, process-relevant object instances, such as specific *orders* and *customers*, provide valuable information. For example, by identifying object instances in the events of the running example, we can recognize that the events comprising the first task (*u1–u8*) do not relate to those comprising the subsequent one (*u19–u16*), since they, respectively, relate to orders *O007501* and *O008102*. However, extracting such information is challenging, because user interaction events generally record object instances only implicitly. In particular, there are typically no dedicated attributes associated with user interactions that capture information about process-related object instances. Instead, these are spread across attributes that describe user interface elements and their values, such as button labels or input values. For example, event *u5* does not make its relation to *order O007501* explicit. Instead, the object type, *order*, is contained in its `Label` and the identifier (*O007501*) in its `Value` attribute.

Table 1
An excerpt of a user interaction data recording the execution of two tasks.

ID	Action	Application	Timestamp	Element	Label	Value
…	…	…	…	…	…	…
u1	click	Mail	15:41:32	list	Order	–
u2	input	Chrome	15:42:10	field	Login	–
u3	input	Chrome	15:42:26	field	Password	–
u4	click	Chrome	15:42:31	button	ok	–
u5	click	Chrome	15:43:01	button	Create order	O007501
u6	input	Chrome	15:43:29	field	Search	Pete Miller
u7	input	Chrome	15:43:43	field	Customer	C0075
u8	click	Chrome	15:43:58	button	Save order	O007501
u9	click	Mail	15:44:32	list	Change request	–
u10	input	Chrome	15:44:41	field	Customer	C0081
u11	click	Chrome	15:45:39	button	Edit order	O008102
u12	input	Chrome	15:45:48	field	Quantity	4
u13	click	Chrome	15:46:05	button	Save order	O008102
u14	click	Chrome	15:46:39	button	Edit order	O008102
u15	input	Chrome	15:46:48	field	Quantity	5
u16	click	Chrome	15:46:55	button	Save order	O008102
…	…	…	…	…	…	…

Challenges of the streaming setting. As indicated in Section 1, our work (also) targets online settings, in which user interaction events arrive in a stream. Recognizing task-level events from a stream is more complex than doing it in an offline manner using an event log, due to the general constraints of streaming settings [11]. Specifically, we have to identify tasks, recognize their type, and extract objects to infer their relations as they are observed, using just a limited buffer to temporarily store a relatively small number of events. This leads to two main difficulties:

1. *Single-pass processing.* In an offline setting, approaches can do multiple passes over an entire collection of events, allowing them to use global information, such as overall co-occurrence counts [8], when making decisions. However, in a streaming setting, events can only be accessed for a limited duration and relevant context information for an observed event may not yet be available. [9]. Therefore, decisions have to be made on the basis of potentially incomplete information, e.g., the co-occurrence counts observed up to the latest event in the stream.
2. *Adapting to changes over time.* An associated issue is that when dealing with streams, decisions have to be made without knowing what kind of events and objects will arrive in the future. For example, while offline approaches can be certain that all types of tasks they need to identify are already available, this is not the case in a streaming setting. At any point in time, events corresponding to new kinds of applications, actions, objects, or task types may be observed. For instance, for the running example, events *u9–u16* must be properly analyzed, even if no such *update order* task has been seen before, which requires on-the-fly updating of the recognition mechanisms.

3. Preliminaries

User interactions and user interaction events. A *user interaction* is a manual action performed on a user interface, such as clicking a button or entering a value into a text field [12]. In line with the definitions of Leno et al. [7], we denote a *user interaction event* (simply *event* in the remainder) $u = (uid, ts, P, V)$ as a tuple that records a user interaction, with $\mathcal{U}$ the universe of all user interaction events. Each event has a unique identifier $u.uid$, a timestamp $u.ts$, a set of context attribute values $u.P$, capturing the interaction type and information about the affected user interface element, and a set of data attribute values $u.V$, capturing data associated with an interaction, e.g., what the user typed into a field. For instance, *u6*=(*u6*, *15:43:29*, {*input*, *Chrome*, *field*, *Search*}, {*Pete Miller*}).

Event classes. Given an event, we let its context attributes values, i.e., $u.P$, define its event class. For instance, the event class of *u6* is given as {*input*, *Chrome*, *field*, *Search*}. We use the shorthand $X.P$ to refer to the set of event classes of all events in a collection X, with $X \subset \mathcal{U}$.

User interaction streams. A user interaction stream S_U is a potentially infinite sequence of events recorded during task execution, i.e., $S_U \in \mathcal{U}^* \forall_{1 \leq i < j \leq |S_U|} S_U(i) \neq S_U(j)$.

Object instances. An object instance o is a pair $(oi, otype)$ with $o.oi$ its identifier and $o.otype$ its object type. For example, the specific order that *u5* creates is given by $o = (O007501, order)$.

Tasks and task-level events. A task is a single unit of work that is part of an organizational process. A *task-level event* $te = (tid, type, ts, D, objects)$ is a tuple that corresponds to the execution of a task, with $\mathcal{T}$ the universe of all task-level events. Each task-level event has an identifier $te.tid$ that uniquely identifies the task that te refers to, relates to a task type $te.type$, has a timestamp $te.ts$, and has optional information captured in its set of attribute–value pairs$te.D$, such as life cycle information that, e.g., indicates whether the event corresponds to the start or completion of a task. Furthermore, a task-level event has a set $te.objects$ of process-related object instances. For example, the start of the task that corresponds to *u9–u16*, is given by te_1 =(*1*, *Edit order*, *15:44:32*, {(lifecycle, *start*)}, {(C0075, *customer*), (O007501, *order*)}).

Task-level event stream. A task-level event stream S_T is a potentially infinite sequence of task-level events, i.e., $S_T \in \mathcal{T}^* \forall_{1 \leq i < j \leq |S_T|} S_T(i) \neq S_T(j)$.

4. Approach

Fig. 1 provides a high-level overview of our approach, which we complement with a formalization in Algorithm 1. As depicted, it recognizes task-level events from a user interaction stream S_U based on three components: the *object-instance-identification component* determines to which object instances events relate, the *task-identification component* identifies sequences of events that correspond to individual tasks, and the *task-categorization component* assigns a type to an identified task. For each recognized task, our approach emits a start and a completion event to a task-level event stream S_T, consisting of a task's identifier, its type, its object instances, a timestamp, and lifecycle information that indicates whether it corresponds to the start or completion of the task.

While object-instance identification is applied to each event as soon as it arrives, the task-identification and categorization components operate on the basis of an event buffer B. In this work, we assume that B is large enough to store the events comprising a single task instance. In online settings, our approach applies task categorization as soon as it identified a task based on the events in the buffer. In offline settings, it first applies task identification to an entire event log and only then continues with task categorization. Also note that, as mentioned in Section 2, we assume that a user performs one task before moving to the next.

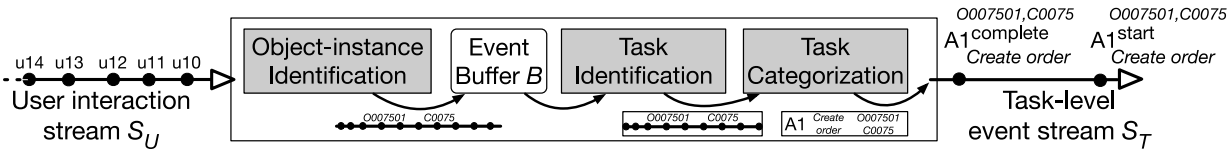


Fig. 1. Overview of our approach.

Algorithm 1 Recognizing task-level events from user interaction data

Input S_U: User interaction stream, b: Maximum buffer size
Output S_T: Task-level event stream

▷ Initialize buffer B, clustering model M, chunk list C, and object instance set $objects$
1: $B \leftarrow$ new FIFOQueue(b), $M \leftarrow$ new ClusteringModel(), $C \leftarrow []$, $objects \leftarrow \emptyset$
2: **loop forever**
3: $u \leftarrow S_U$.observeEvent() ▷ A new event is consumed from the stream
▷ Object-instance identification
4: $objects \leftarrow objects \cup \{(u.uid, \texttt{identifyObject}(u))\}$ ▷ Identify object instance
▷ Add the event to the buffer
5: B.insert(u)
▷ Task identification
6: **if** completesChunk(u) **then**
7: C.add(B.getEventsSinceLastChunk(C)) ▷ Create and store new chunk
8: **if** $|C| \geq 2$ **then** ▷ Check if enough chunks available
9: $c_i, c_{i+1} \leftarrow C[-2], C[-1]$ ▷ Get chunks to be checked
10: **if** endsTask(B, C, c_i, c_{i+1}, $objects$) **then** ▷ Check if c_i completes a task
11: $taskEvents \leftarrow (B$.dequeueUpThrough(c_i)) ▷ De-queue events that comprise the task
12: $C \leftarrow C$.removeRange($C[0], c_i$) ▷ Remove chunks that are part of new task
▷ Task categorization
13: $v \leftarrow$ vectorize($taskEvents, objects$) ▷ Create a feature vector of the task
14: M.update(v) ▷ Update the clustering model
15: $type \leftarrow M$.categorizeTask(v) ▷ Assign a type to the task
16: $label \leftarrow$ label, getDefiningTerms($taskEvents$) ▷ Assign a label to the task
▷ Emit task-level events
17: $tid \leftarrow$ newID() ▷ Assign an ID to the task
18: $taskObjects \leftarrow \{obj \mid (event, obj) \in objects\}$
19: $objects \leftarrow \emptyset$ ▷ Empty the objects set for the next task
20: emit(S_T, (tid, $type$, $taskEvents[0].ts$, {(lifecycle, *start*),(label, $label$)}, $taskObjects$))
21: emit(S_T, (tid, $type$, $taskEvents[-1].ts$, {(lifecycle, *complete*),(label, $label$)}, $taskObjects$))

4.1. Object-instance identification

First, our approach identifies object instances in the events, which (1) it uses in the subsequent components and (2) which indicate if and how recognized task-level events relate to each other through shared object instances (line 4). As shown in Fig. 2, the object-instance-identification component consists of two parts that are applied for each event u: First, *type extraction* aims to extract an object type ot from u. Then, *instance recognition* detects if u indeed refers to an instance of ot and—if so—adds it to the current task's object instances. In the following, we explain these parts in detail.

4.1.1. Type extraction

The first part of this component extracts an object type ot from a given event u. For instance, it aims to detect that *u1*, *u5*, and *u8* each refer to an *order*, while *u7* refers to a *customer*. This involves *noun identification* and *UI-object removal*, which we describe next.

Noun identification. We recognize that object type information is often contained in context attribute values, e.g., in button labels such as *Save order*. Therefore, noun identification establishes a set of nouns N_u from the context attribute values P of event u.

To establish N_u, we employ a part-of-speech (POS) tagger provided by standard NLP tools (e.g., spaCy [13]). Given a context-attribute value p, e.g., *Create order* (see *u5*), a POS-tagger assigns linguistic roles to individual words in p, e.g., it assigns VERB to *Create* and NOUN to *order*. Using such a tagger, we instantiate a function nouns that, given a value $p \in u.P$ returns the set of nouns in p. For instance, nouns(*Create order*) $= \{order\}$. Noun identification applies nouns to all $p \in u.P$, which results in a set $N_u = \bigcup_{p \in u.P} \texttt{nouns}(p)$ that may contain process-related nouns. However, this set likely also contains nouns that rather relate to the user interface itself, which are removed next.

UI-object removal. Having identified a set of nouns N_u, UI-object removal discards any nouns that pertain to user interface elements (e.g., *textfield* or *excel*) rather than process-related objects, as these cannot be used to establish meaningful relations between tasks.

To this end, we use a set K_U of user-interface-specific terms, which consists of names of different UI elements, such as *button*, *field*, and *link*, common application names, such as names of browsers, spreadsheet applications, text-processing software, productivity tools, and application-specific objects, such as *workbook*, *sheet*, and *cell* in case of MS Excel.[1] If a noun $n \in N_u$ corresponds to a term in K_U, it is not process-related and thus removed from N_u. In this manner, our approach establishes a set of process-related nouns $N_u^{ot} = N_u \setminus K_U$.

Output. If N_u^{ot} still contains nouns after this removal step, i.e., $N_u^{ot} \neq \emptyset$, our approach concatenates these to represent the object type ot of u. For instance, $N_u^{ot} = \{order\}$ becomes $ot = order$, while $N_u^{ot} = \{order, line\}$ becomes $ot = order\ line$ before it continues with instance recognition

[1] For the full set K_U of terms we refer to our repository linked in Section 5.

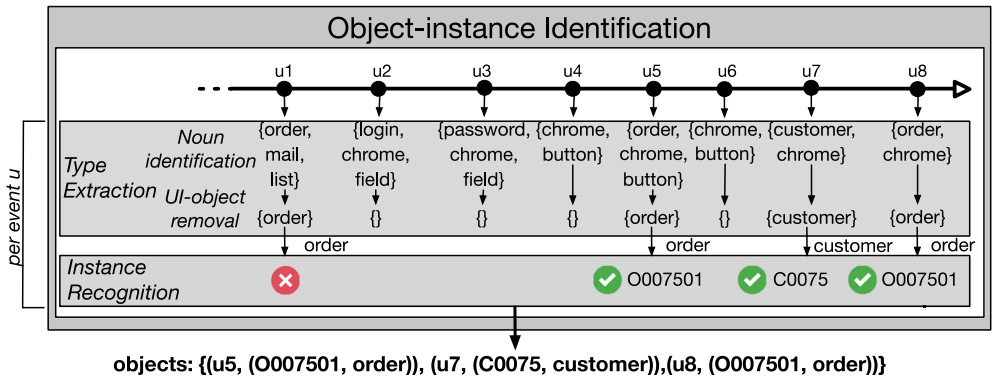


Fig. 2. Object-instance-identification component.

for u. If no nouns remain, it continues with type extraction for the next event.

4.1.2. Instance recognition

Having extracted an object type ot from u, the object-instance identification component next aims to recognize if u indeed refers to a specific instance of ot. For example, it recognizes that the *O007501* `Value` of event *u5* represents the identifier of the specific order that was created by *u5*.

To this end, instance recognition establishes a set of identifying values I_u^{oi} that jointly represent an object identifier oi based on u's value attributes V. We recognize that, in the context of user interaction events, such identifying values are typically alphanumeric IDs, URLs, email addresses, and names of people, organizations, and places. Therefore, our approach adds any value $v \in u.V$ to I_u^{oi} that (partially) consists of digits, corresponds to a URL or email address, or refers to a named entity such as *Pete Miller* (see *u6*). Note that digits, URLs, and email addresses can be straightforwardly detected using regular expressions, whereas named entities, i.e., persons, organizations, countries, cities, etc., can be detected using named-entity-recognition capabilities of standard NLP-tools [13].

Instance recognition then concatenates any identifying values in the same manner as done for nouns in type extraction to represent the object identifier oi.

Provided that the object-instance-identification component extracts an object type and recognizes a corresponding identifier in u, it establishes an object instance $o = (oi, ot)$ and adds it to the set of current object instances (line 4). Regardless of whether an object instance was added or not, the component then continues with the next event.

4.2. Task identification

The task-identification component identifies sequences of events from the stream that correspond to individual tasks. It consists of two main operations, as visualized in Fig. 3. Here, *chunking* identifies sequences of observed events that represent sub-tasks, such as filling in a form or sending an e-mail, whereas *segmenting* determines if consecutive sub-tasks corresponds to the same process-level task or rather to different ones. Once such a transition from one task to the next has been detected, we forward the segment that corresponds to the completed task to our task-categorization component.

4.2.1. Chunking

We recognize sub-tasks by looking for common keywords in user interaction data that indicate the conclusion of an interaction sequence, achieved through the `completesChunk` function in Algorithm 1 (line 6). To operationalize this function, we established a set of completion actions K_A, which consists of 20 keywords stemming from design guidelines for user interfaces by IBM [14], covering typical terms that indicate the conclusion of a smaller part in a process, such as *ok* (to go to the next step in a user interface), *submit* (for a form), *send* (e-mail), or *save* (changes).[2]

For an event u, `completesChunk`(u) returns true if u's event class contains a mention of an action in K_A. Based on the events stored in B, a sub-task is formed by the events that occurred since the last completed chunk in C (line 7). For instance, for the running example, *u4*, *u8*, *u13*, and *u16* complete chunks (due to their *ok* and *save* labels), which results in *u1–u4*, *u5–u8*, *u9–u13*, and *u14–u16* as chunks.

4.2.2. Segmenting

The segmenting operation aims to decide whether a chunk c_i corresponds to the end of a task or if it continues with the next chunk, c_{i+1} (function `endsTask` in line 10). Specifically, as shown in Fig. 3, `endsTask` identifies c_i as finalizing a task if: (1) the chunks are contextually unrelated to each other, (2) the chunks have no overlap in data values, (3) c_i does not represent an overhead activity, and (4) the control-flow after c_i is non-deterministic. Otherwise, c_i and c_{i+1} are considered to belong to the same task.

(1) Assessing contextual relatedness. Our approach first checks if c_i and c_{i+1} are contextually related or not. We do this by lifting the notion of contextual relatedness proposed by Urabe et al. [8], which targets offline segmentation, to our setting. The idea is to check if the event classes contained in c_i and c_{i+1} commonly co-occurred so far (indicating a shared context) or not (suggesting that the chunks belong to different tasks).

As illustrated in Fig. 4, contextual relatedness is quantified on the basis of a global co-occurrence matrix, which tracks how often pairs of event classes have been observed to be part of the same chunk. Based on the global counts, we obtain the co-occurrence vectors of the event classes per chunk (i.e., rows in the co-occurrence matrix) and compute their centroid. Then, we compute the similarity score $\texttt{sim}(c_i, c_{i+1})$ as the cosine similarity between the centroids of c_i and c_{i+1}. Given a similarity threshold $t \in [0, 1]$,[3] we consider the two chunks contextually unrelated if $\texttt{sim}(c_i, c_{i+1}) < t$.

In this manner, given the four chunks identified in the previous operation, we would determine that the transitions from *u1–u4* (logging in) to *u5–u8* (creating an order), and from *u5–u8* (creating an order) to *u9–u13* (updating a quantity) are both clear changes in context. By contrast, the transition from *u9–u13* to *u14–u16* occurs within the same context (updating and fixing an order quantity), due to the chunks' strongly related event classes.

(2) Checking for data value overlap. Next, we recognize that sub-tasks may be part of the same task, even when they relate to different

[2] We refer to our repository for the full list of keywords, though K_A can naturally be extended with, e.g., self-defined keywords or other languages.

[3] t is configurable and we set it to 0.3 by default.

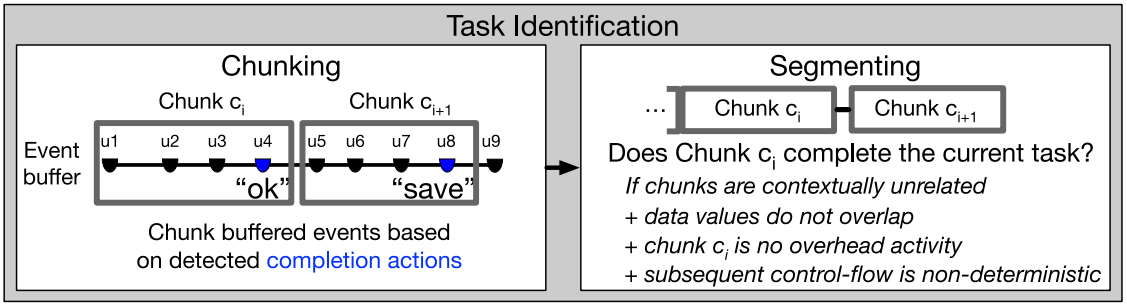


Fig. 3. Task-identification component.

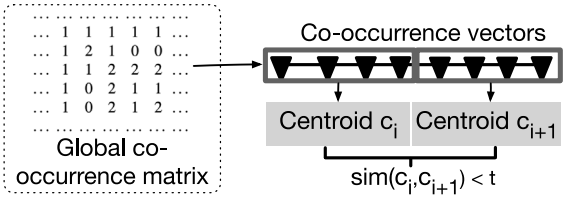


Fig. 4. Contextual-relatedness approach inspired by Urabe et al. [8].

contexts, such as opening a request sent by a customer per e-mail and subsequently updating one of their orders in a system. Therefore, we check if events belonging to chunks c_i and c_{i+1} share particular attribute values, including IDs of identified object instances, such as customer names or order numbers. Specifically, we check the last two events of c_i and the first two of c_{i+1} for exact matches in their attribute sets V, and, if such matches are present, determine that there should be no segmentation between c_i and c_{i+1}.

In this manner, we would, for instance, recognize that chunks *u9–u13* and *u14–u16* also relate to each other in terms of their data values, because events *u13* and *u14* both refer to order *O008102*, thus avoiding segmentation here.

(3) Checking for overhead sub-tasks. Then, we check if c_i actually corresponds to a sub-task performed for a particular process instance or that it, rather, corresponds to overhead being performed. Common examples of this include logging into a system, launching an application, or visiting non-work related websites. If c_i represents such an overhead sub-task, we do not want to treat this chunk as a distinct task on a process level, which is why we would not segment after c_i (even though contextual relatedness or shared data values between c_i and c_{i+1} are unlikely).

To operationalize this check, we established a set K_O of overhead keywords based on the guidelines [14] we also use for chunking, including *log in*, *sign up*, *reload*, and *open*. Using this set, we check if a member of the last two event classes of c_i is contained in K_O and, if so, avoid segmentation. In this manner, we, e.g., recognize that the first sub-task in our running example (*u1–u4*), which corresponds to a user logging into a web app, belongs to the same task as the next chunk *u5–u8*, where the same app is used to create an order.

Note that if one wants to specifically investigate the occurrence of overhead actions in a process, one can simply disable this check in our approach, which will result in such chunks being treated as separate tasks.

(4) Checking for control-flow determinism. Finally, we check if chunks that consist of c_i's event classes have always been followed by the same behavior so far. Such control-flow determinism suggests that c_i does not complete a task because it is always necessary to perform the exact same steps after it.

To be able to check for control-flow determinism, we count how often sets of event classes directly follow each other throughout the stream and compare it to the number of times the set of c_i's event classes, $c_i.P$, formed a chunk so far. If these counts are the same, i.e., $\texttt{count}(c_i.P) = \texttt{countDF}((c_i.P, c_{i+1}.P))$[4], the chunk's subsequent control-flow is considered to be deterministic.[5] This suggests that their corresponding sub-tasks belong to the same task and we avoid segmentation after c_i. These counts can be stored efficiently by identifying event classes through their index in the co-occurrence matrix used in the first check, thus, only storing sets of indices and their counts instead of storing sets of entire event classes.

Note that the checks based on contextual relatedness and control-flow determinism may benefit from a *warm-up phase*, during which we populate the co-occurrence matrix and control-flow counts for a certain number of events before making the first segmentation decision based on them.

4.2.3. Post processing

When our approach has detected that c_i represents the final chunk of a task (line 10), this means that all events currently in the buffer, up to and including the final event of c_i, together form a task. The events that comprise the task are then forwarded to the task-categorization component and removed from buffer B as well as chunk list C (lines 11–12), so that the first event in B is the first event of the next task.

4.3. Task categorization

The task-categorization component assigns a type to identified tasks. Given that we cannot store identified tasks in a streaming setting, we categorize them directly after task identification. This is complex, though, because it means we may not yet have observed all possible task types.

To deal with this challenge, we perform task categorization on the basis of an online clustering model M, which is incrementally updated as new task instances arrive. As shown in Fig. 5, this involves the transformation of a task into a feature vector, updating the model M, then using it to assign a cluster to the task, and—finally—providing a textual label for the task.

4.3.1. Establishing feature vectors of tasks

Given an identified task, we first transform its contents into a feature vector that can be used for clustering (line 13). We use a vector encoding that accounts for variability in the executions of tasks of the same type, such as tasks that consist of slightly different sets of event classes or that are performed in a different order. Therefore, we capture the number of unique event classes (as an indicator of a task's complexity), and the frequency of each event class (to capture its contents) as features, with a fixed position for each event

[4] DF stands for directly-follows.

[5] Note that we only apply this check if $\texttt{count}(c_i.P) \geq 3$, to avoid using it for new sets of event classes.

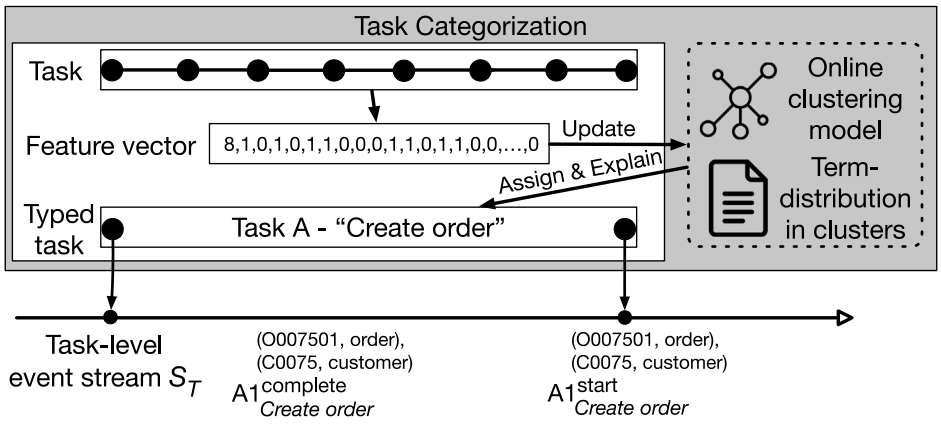


Fig. 5. Task-categorization component.

class.[6] For instance, the task *u1–u8* in our running example consists of eight event classes that are all performed once, resulting in a vector $\langle 8,1,0,1,0,1,1,0,0,0,1,1,0,1,1,0,0,0,\ldots,0\rangle$. Here, the zeros at the end are used to ensure that vectors remain of the same size s_v throughout the stream,[7] accounting for a number of event classes not seen so far. In offline settings, the vector size can be set to the number of unique event classes in the entire input log plus one (to account for the number of unique event classes in the task).

4.3.2. Clustering tasks

We use an online clustering model M to recognize tasks that are of the same type, based on their vector representation. Specifically, we use *DenStream* [15], a density-based online clustering technique building on the DBSCAN algorithm [16]. It dynamically creates, updates, and deletes so-called micro-clusters in the online feature space it maintains. This technique has two key benefits. First, the technique is highly memory efficient, since it only stores summaries of vector sets (the micro-clusters), rather than the vectors themselves. Second, unlike many other clustering techniques, it does not depend on a user-defined number of desired clusters.

As shown in Fig. 5 and Algorithm 1, we update the clustering model M with the vector v that corresponds to an identified task (line 14), before using it to assign the task to a cluster (line 15). For instance, due to the distinct features of the two tasks in our running example, these are assigned to different clusters (e.g., types *A* and *B)*.

To improve the performance of task categorization, it can also be beneficial to introduce a *warm-up phase*, which postpones assignment of task types to a point in time when the clustering model has already been updated with a number of identified tasks, and thus is more mature than it is with a cold start. Note that, in offline settings, such a warm-up phase effectively spans the entire event log, since we can then establish a clustering model based on all identified tasks, before assigning categories to them.

4.3.3. Task labeling

After using clustering to recognize the type of a task, we provide the task with a textual label that indicates what its type actually means.

We automatically generate a suitable label for a given task, by considering the terms that are distinctive to the event classes for tasks in its assigned cluster. For this, we use the well-known *term frequency–inverse document frequency* (tf-idf) score. Specifically, we use a term dictionary to keep track of the frequency of terms used in the event classes within tasks of a specific type, i.e., cluster. Using $M.types$ to refer to the types currently recognized by the clustering model M, we write $\texttt{tf}-\texttt{idf}(x, type)$ as the score for a term x and a type $type \in M.types$.

Then, we set the label of *type* as the term x with the highest $\texttt{tf}-\texttt{idf}(x, type)$, e.g., *Create order* for type *A*. If multiple types are assigned the same label (e.g., if *Quantity* is the most distinctive term for types *B* and *C*), we add the term with the next highest `tf-idf` score to each label, until they are all unique. For example, after basic post-processing (term re-ordering and removing capitalization), *B* may get the label *Edit order*, while *C* gets *Confirm quantity*.

4.4. Output

The output of our approach is a stream of task-level events based on the identified and categorized tasks as well as the object instances identified in the events that comprise the tasks. For each task, we emit a start event with the timestamp of its first user interaction event and a completion event with the last timestamp (lines 20–21). Both task-level events are assigned a task identifier (created in line 17), the task's type, its label, and its object instances. For example, for *u1–u8*, we emit:

$$te_1 = \{1, A, 15{:}41{:}32,\ \{(\texttt{lifecycle}, start), (\texttt{label}, Create\ order)\},\ \{(O007501, order), (C0075, customer)\}\}$$

$$te_2 = \{1, A, 15{:}43{:}58, \{(\texttt{lifecycle}, complete), (\texttt{label}, Create\ order)\},\ \{(O007501, order), (C0075, customer)\}\}$$

and for *u9–u16* we emit:

$$te_3 = \{2, B, 15{:}44{:}32,\ \{(\texttt{lifecycle}, start), (\texttt{label}, \{Edit\ order\})\},\ \{(O008102, order), (C0081, customer)\}\}$$

$$te_4 = \{2, B, 15{:}46{:}55, \{(\texttt{lifecycle}, complete), (\texttt{label}, \{Edit\ order\})\},\ \{(O008102, order), (C0081, customer)\}\}$$

Compared to the user interaction events in our running example, our approach thus emits events that fulfill the requirements of process mining settings because they relate to a process-level activity and to a process execution.

[6] Note that the encoding could also capture the types of object instances identified in the events that comprise a task, but in our experiments this did not lead to performance improvements.

[7] Given that, in online settings, the final number of event classes is unknown, s_v should be set sufficiently large. We set 1000 as the default for s_v, which already covers more than 6 times the total number of event classes in our evaluation data.

Table 2
Characteristics of the task logs used in our experiments.

Task Log	Source	Description	#Tasks	Avg.len.	#Var.	#Events	#Classes
1	[7]	Copy address data	100	14.5	7	1462	15
2	[7]	Enter reimbursement details	50	62.3	1	3113	32
3	[7]	Enter student record	50	30.8	2	1539	23
4	[17]	Fill in travel request	40	73.5	2	2940	48
5	[1]	Create structural unit	30	10.9	30	331	9
6	[1]	Create chapter	30	10.1	30	425	12
7	[1]	Create organizational unit	30	14.2	30	304	8
8	[1]	Create specification	30	11.0	30	326	9

5. Evaluation

In this section, we describe the evaluation experiments we conducted. We describe the data collection used in our experiments in Section 5.1 and the setup in Section 5.2. In Section 5.3 we present the evaluation results showing our approach's capability to automatically recognize task-level events from user interaction data in comparison to baseline approaches. The implementation, data collection, evaluation pipeline, and raw results can be found in our repository.[8]

5.1. Data collection

We aim to show that our approach is capable of recognizing task-level events of varying types. However, there are no publicly available event logs (let alone streams) that contain interaction data related to different task types that are associated with a necessary gold standard. Therefore, we follow the idea of Urabe et al. [8] and take available task logs, each recording various instances of the same type, and combine these into three evaluation logs, which thus cover multiple task types, in various orders. We use these event logs as input in offline settings and as a basis to simulate event streams in online settings.

5.1.1. Task logs

As shown in Table 2, we have eight task logs from three sources available, which include gold-standard task instances (though contained task IDs) and associated types (through their description in the source).[9]

The types of tasks that these logs cover can be divided into two groups:

1. Task logs 1–4 involve *copying address data* from spreadsheets to web forms, entering *reimbursement details*, entering *student records* into a web-based app, and filling in *travel requests*. Logs 1–3 were all published by Leno et al. [7], whereas log 4 stems from a tutorial given by Agostinelli et al. [17]. Note that task log 1 originally contained substantially more instances (1000 versus 50) and events (14,582 versus 1,539–3,113) than the others. Therefore, we use a sample of its tasks, so that we maintain a more balanced distribution among types as well events per log. To this end, we randomly select 100 instances using stratified sampling with respect to the different execution variants.
2. Task logs 5–8 all relate to the creation of informational objects, such as organizational units, in an SAP system and are provided by Abb and Rehse [1].

Four out of these eight task logs (1–4) contain overhead tasks such as logging into a system, starting an application, or opening a file. As shown in the table, the logs also differ considerably in their variation and task lengths. Note that we also unified the data structure across the task logs as much as possible before combining them, such that attribute names are the same for all task types.

5.1.2. User interaction logs

We established three evaluation logs (L_{U1}–L_{U3}) by combining the tasks from individual task logs:

$\boldsymbol{L_{U1}}$: L_{U1} consists of instances from task logs 1–3 (which are also used by Urabe et al. [8] in their evaluation). We start from an empty log L_{U1} and (1) randomly select a task log and an instance from it, (2) add this instance to L_{U1}, and (3) remove it from the task log, until all instances have been added to L_{U1}. The resulting user interaction log L_{U1} consists of 200 tasks and has a total of 6114 events.

$\boldsymbol{L_{U2}}$: We established L_{U2} from task logs 1–4, in the same manner as L_{U1}. The resulting user interaction log L_{U2} consists of 240 tasks and contains 9054 events.

$\boldsymbol{L_{U3}}$: L_{U3} is based on task logs 5–8. We use these task logs separately from the other ones, since their event attributes, and thus event classes, differ considerably. In particular, none of the event classes that logs 5–8 contain, occur in logs 1–4. Combining them into one evaluation log would, thus, considerably simplify the identification of transition points between tasks. We combined logs 5–8 in the same way as done for L_{U1} and L_{U2}, so that the resulting user interaction log L_{U3} consists of 120 tasks and contains 1,386 events.

We provide all logs (task logs 1–8 and L_{U1}–L_{U3}) in our repository[10] alongside the implementation. When performing experiments for online settings, we use L_{U1}–L_{U3} to simulate event streams (S_{U1}–S_{U3}).

5.2. Setup

This section describes the environment, configurations, baselines, and measures used in our experiments.

5.2.1. Environment

We implemented our approach in Python and ran our experiments single-threaded on a laptop with a 2 GHz Intel Core i5 processor and 16 GB of memory.

5.2.2. Configurations

Our approach requires a buffer size b that can store all events belonging to a single task. We report on the results using a buffer size of 250 events (also for the streaming baselines), which covers three times the number of events of the longest task in our data collection. Furthermore, we consider that parts of our approach are initialized at the beginning of a stream and populated over time: the global co-occurrence matrix and control-flow-counts for task identification and the online clustering model for task categorization. Given that the accuracy of these components may improve as more events are

[8] https://gitlab.uni-mannheim.de/processanalytics/task-recognition-from-user-interaction-data

[9] Note that these logs do not include a gold standard for object instances. To establish that gold standard, both authors of this manuscript annotated user interaction events with correct object instances independently. Afterwards, these were compared and any discrepancies were settled in a discussion. The gold standards are available in our repository.

[10] https://gitlab.uni-mannheim.de/processanalytics/task-recognition-from-user-interaction-data

observed, we test the value of a *warm-up phase*, where our approach populates the matrix, control-flow counts, and clustering model using the first 0, 100, 250, 500, or 1000 events, before starting identification and categorization on them. Note that, in offline settings, the number of warm-up events corresponds to the number of events in the input log.

5.2.3. Baselines

We compare against three streaming baselines and two offline baselines with respect to task identification and categorization. Note that we cannot compare against any baseline with respect to object-instance identification, because none of the existing works considers that as a part of recognizing tasks.

Streaming baselines. We compare our approach to three baselines in a streaming setting. Aside from our previous work [10], we established two baselines by adapting existing works, since there are currently no other techniques capable of recognizing tasks based on a stream of user interactions. The task-identification component of the two additional baselines consists of an existing, offline identification technique, lifted to an online setting (as described below). Their task-categorization components, by contrast, are operationalized with the same technique used in our approach. This is necessary because existing offline categorization techniques cannot be lifted to an online setting.

- BL_{caise}: *Our previous approach.* In our previous work [10], we proposed the first approach for recognizing task-level events from user interaction streams, which we use to investigate the value of our extended task-identification component.
- BL_{dfg}: *Back-edge-based identification.* Leno et al. [7] proposed a log segmentation technique based on back-edges identified from a directly-follows graph (DFG). We adapted the technique to build the DFG incrementally using the same event classes as available to our approach and apply the authors' back-edge detection method periodically, after every b events (i.e., each time the buffer is full).
- BL_{co-oc}: *Co-occurrence-based identification.* The segmentation approach by Urabe et al. [8], which also inspired parts of our work, leverages co-occurring event classes in fixed windows to segment a log. We adapted it to count co-occurrence incrementally and compute similarities on a buffer of events. We use the same parameter configurations as reported in the original paper.

Offline baselines. When evaluating our approach in offline settings, we compare it against two baselines:

- BL_{leno}: *Back-edge-based identification.* The original approach by Leno et al. [7] serves as the first offline baseline. As described for BL_{dfg}, it constructs a DFG based on event classes that immediately follow each other in a given log. Then, it detects back-edges in the graph, which it uses to segment the log into tasks. As such, this baseline only covers task identification.
- BL_{urabe}: *Co-occurrence-based identification and subsequent clustering.* The original approach by Urabe et al. [8] consists of two phases: task identification and task clustering. First, it segments the complete input log using co-occurring event classes (as explained above), before applying agglomerative hierarchical clustering to categorize tasks. As such, it covers both task identification and task categorization.

5.2.4. Measures

We use the following measures to assess quality in our experiments.

Object-instance-identification quality. We assess object-instance-identification quality through the well-known precision, recall, and F_1-measures by comparing the object instances our approach identified from a stream/log to the object instances in the manually created gold standard. Using A to denote the set of object instances identified by our approach and G for the set of object instances in the gold standard these measures are defined as follows:

- *Precision* (*Pre.*). Precision is the fraction of object instances that are actually correct ($|A \cap G| / |A|$).
- *Recall* (*Rec.*). Recall is the fraction of object instances in the gold standard that were also correctly identified by our approach ($|A \cap G| / |G|$).
- F_1*-score* (F_1). The F_1-score is the harmonic mean of precision and recall.

Task-identification quality. We assess task-identification quality by comparing the identified task segments to those of the corresponding gold standard, for which we use two measures:

- *#tasks.* To assess if an approach makes the right amount of segmentation decisions, we compare the numbers of identified and gold-standard tasks.
- *Normalized edit distance* (*n.ED*). To quantify how similar the identified tasks are in comparison to the gold standard, we calculate the average normalized edit distance between identified tasks and their closest task in the gold standard.

Task-categorization quality. We assess categorization quality through measures for cluster quality, by comparing the tasks that are assigned to the same category in the gold standard (i.e., task types).

Cluster quality.

- *Rand index* (R). We compute the Rand index, which considers the fraction of pairs (tasks at macro level, events at micro level) that are correctly assigned to the same or to different categories, i.e., $(TP+TN)/(TP+TN+FP+FN)$, where a true positive (TP) indicates that two tasks/events are correctly assigned to the same category.
- *Jaccard index* (J). We also compute the weighted average Jaccard index to quantify the similarity between identified clusters and the gold-standard clusters, which is given as $A \cap G / A \cup G$ per cluster, with A a cluster's contents identified by our approach and G its gold-standard contents (i.e., tasks at the macro level, events at the micro level).

5.3. Results

In this section, we first present the overall results of our approach applied in an online setting in comparison to the streaming baselines (Section 5.3.1), before examining the impact of a warm-up phase (Section 5.3.2). Following this, we discuss the importance of considering various types of information in task identification through an ablation study (Section 5.3.3). Shifting our focus to offline settings, we report on our approach's performance in comparison to the offline baselines (Section 5.3.4), followed by a discussion on performance online versus offline settings (Section 5.3.5). Subsequently, we provide insights into the results from our approach and baselines applied on the individual task logs (Section 5.3.6) and conclude with a discussion on computational efficiency (Section 5.3.7).

5.3.1. Online results

Table 3 shows the results obtained using our approach applied in an online setting, the three streaming baselines, and a perfect identification strategy (to show the quality of task categorization independently of task-identification quality), all with a warm-up phase of 250 events.

Object-instance-identification results. Our approach achieves accurate results in terms of object-instance identification for S_{U1} and S_{U2}, both with an F_1-score of 0.9 or higher. The similar precision and recall scores (0.91 resp. 0.88 for S_{U1} and 0.95 and 0.93 for S_{U2}) further suggest that the object-instance-identification component is both accurate in identifying actual object instances and effective in identifying most of these object instances. Note that for S_{U3}, we could not obtain any results, because there is simply no object instance information available to be identified. Furthermore, recall that none of the baselines considers

Table 3
Results of our approach and the baselines (warm-up of 250 events). *Perf. ident.* shows task-categorization results in case of perfect task identification. ↑ and ↓ indicate the desired direction per measure.

Stream	Approach	Obj. ident.			Task ident.		Task categorization			
		Pre.↑	Rec.↑	F_1 ↑	#tasks	n.ED↓	R(mi)↑	R(ma)↑	J(mi)↑	J(ma)↑
S_{U1}	*Ours*	0.91	0.88	0.90	**202**	**0.04**	**0.96**	**0.97**	**0.92**	**0.95**
	BL_{caise}	–	–	–	**198**	**0.04**	0.94	0.95	**0.92**	0.92
	BL_{dfg}	–	–	–	**202**	0.83	0.58	0.82	0.38	0.89
	BL_{co-oc}	–	–	–	159	0.33	0.74	0.80	0.64	0.71
	Perf. ident.	–	–	–	200	0.00	1.00	1.00	1.00	1.00
S_{U2}	*Ours*	0.95	0.93	0.94	**241**	**0.05**	**0.96**	**0.97**	**0.92**	**0.95**
	BL_{caise}	–	–	–	231	0.06	0.89	0.95	0.82	0.89
	BL_{dfg}	–	–	–	99	0.32	0.42	0.80	0.24	0.56
	BL_{co-oc}	–	–	–	198	0.33	0.69	0.71	0.50	0.51
	Perf. ident.	–	–	–	240	0.00	0.97	0.97	0.95	0.95
S_{U3}	*Ours*	–	–	–	**132**	**0.17**	**0.97**	**0.99**	**0.93**	**0.99**
	BL_{caise}	–	–	–	138	0.23	0.87	0.88	0.76	0.76
	BL_{dfg}	–	–	–	29	0.58	0.34	0.49	0.16	0.35
	BL_{co-oc}	–	–	–	58	0.37	0.45	0.57	0.31	0.46
	Perf. ident.	–	–	–	120	0.00	1.00	1.00	1.00	1.00

object-instance identification, thus, we could not obtain respective results for those either.

Looking at the results in detail reveals that our approach is capable of correctly identifying objects of various types. For instance, it correctly extracts objects of the type *bank account*, identified by an account number, and of type *tax code*, identified by a respective code. However, there are also cases where our approach fails, in particular because object types were not recognized correctly. Type recognition relies on the accurate identification of nouns; hence, failing to correctly identify nouns hinders the recognition of object instances. For instance, noun identification failed to recognize that *address* is a noun, as it can also function as a verb, which leads to relevant instances being missed. Similarly, it does not recognize types that are not represented by actual words, e.g., *countycode* (which misses white space between *county* and *code*). Conversely, although discarding UI-specific objects (such as *button* or *text field*) generally avoids false positives well, our approach identified some instances that do not have gold standard counterparts. For instance, it identified objects of type *personal data*, associating these with an identifier. However, such objects are not informative and their identification should, thus, be avoided.

Overall, the results still show that our approach is capable of accurately identifying object instances from user interaction streams, allowing it to relate tasks to each other.

Task-identification results. Our approach achieves highly accurate results for S_{U1} and S_{U2}, identifying approximately the same numbers of tasks as in the gold standard (202 vs 200 and 241 vs 240), to which they are very close in terms of contents, yielding edit distances of just 0.04 and 0.05. However, stream S_{U3} is more challenging. Our approach overestimates the total number of tasks (132 versus 120), achieving an edit distance of 0.17. Taking an in-depth look into the results, we find that it occasionally fails to recognize that certain sub-tasks belong to the same gold-standard task, since they lack contextual relatedness and overlapping data values.

Compared to the baselines, our approach consistently obtains better results in terms of edit distances. This indicates that the tasks that the baselines identify differ more from their gold-standard counterparts than the ones identified by our approach. Our previous approach, BL_{caise}, performs similarly well as our approach for S_{U1} and S_{U2} (with 0.007 improvement for S_{U1} and 0.01 improvement for S_{U2}), yet considerably better for S_{U3} (improved by 0.06). This improvement can be attributed to the additional check for deterministic control-flow that our approach employs, which BL_{caise} does not do. BL_{dfg} and BL_{co-oc} often miss segmentation points, resulting in much lower numbers of identified tasks than contained in the gold standard. BL_{dfg}, specifically, only finds 99 tasks for S_{U2} (out of 240) and 29 for S_{U3} (out of 120). Although BL_{co-oc} generally performs better than BL_{dfg}, we find that its results are heavily dependent on the selection of two parameter values, with the edit distances differing by up to 0.5 across configurations.[11]

Overall, these results indicate that our approach, which considers the semantic and data perspectives in addition to the control-flow perspective considered by BL_{dfg} and BL_{co-oc}, leads to more accurate task identification. Furthermore, our approach does not depend on user-defined parameters (unlike BL_{co-oc}).

Task-categorization results. As for task categorization, our approach achieves high macro Rand scores of 0.97 for S_{U1} and S_{U2} and 0.99 for S_{U3}, which shows that it accurately assigns pairs of tasks to the same category as their gold-standard counterparts. The comparable micro-level scores show that this categorization quality generally also holds for pairs of events, which thus accounts for tasks of different lengths. The Jaccard index, which provides insights into the quality per cluster, rather than per task (or event) pair, confirms the accurate categorization quality, achieving macro scores of 0.95 for S_{U1} and S_{U2} and 1.00 for S_{U3} and the comparable scores on the micro level (0.92, 0.92, and 0.97).

As shown by the results obtained when using our task-categorization component on perfectly identified tasks (gray row in Table 3), the categorization itself is highly accurate, achieving perfect scores for S_{U1} and S_{U3}, and near-perfect ones (≥0.95) for S_{U2}. The improved task-categorization performance on S_{U3} compared to BL_{caise} can, therefore, be attributed to our improved task-identification component. The subpar results of BL_{dfg} and BL_{co-oc}, which use the same categorization technique as our approach, also clearly indicate that lower identification quality leads to lesser categorization results.

5.3.2. Impact of the warm-up phase

Table 4 shows the results of our approach for warm-up phases of 0, 100, 250, 500, and 1000 events. The results indicate that the warm-up phase does not have any impact on task-identification quality. For task-categorization quality, the benefit of a warm-up phase becomes clear, though.[12] A closer look at the streams suggests that this benefit relates to whether the warm-up phase covers each task type at least once. For S_{U1}, this coverage occurs within the first 250 events but not within the first 100 events. Between these two warm-up phases we see a clear performance boost. While for up to 100 warm-up events, we achieve a macro Rand score of 0.84 and Jaccard score of 0.78 for S_{U1}, setting the warm-up phase to 250 events increases the scores by 0.13

[11] See our repository for detailed experiments regarding BL_{co-oc}'s parameter configurations.

[12] Note that warm-up phases do not apply for object-instance identification as this is done per event.

Table 4
Results of our approach with warm-up phases of 0, 100, 250, 500, and 1000 events. ↑ and ↓ indicate the desired direction per measure.

Stream	#Events	Obj. Ident.			Task ident.		Task categorization			
		Pre.↑	Rec.↑	F_1 ↑	#tasks	n.ED↓	R(mi)↑	R(ma)↑	J(mi)↑	J(ma)↑
S_{U1}	*0*	0.91	0.88	0.90	202	0.04	0.89	0.84	0.84	0.78
	100	‘’	‘’	‘’	‘’	‘’	0.89	0.84	0.84	0.78
	250	‘’	‘’	‘’	‘’	‘’	0.96	0.97	0.92	0.95
	500	‘’	‘’	‘’	‘’	‘’	**0.98**	**0.99**	**0.97**	**0.98**
	1000	‘’	‘’	‘’	‘’	‘’	0.98	0.99	0.97	0.98
S_{U2}	*0*	0.95	0.93	0.94	241	0.05	0.96	0.97	0.92	0.95
	100	‘’	‘’	‘’	‘’	‘’	0.96	0.97	0.92	0.95
	250	‘’	‘’	‘’	‘’	‘’	0.96	0.97	0.92	0.95
	500	‘’	‘’	‘’	‘’	‘’	0.96	0.97	0.92	0.95
	1000	‘’	‘’	‘’	‘’	‘’	**0.98**	**0.99**	**0.97**	**0.99**
S_{U3}	*0*	–	–	–	132	0.17	0.90	0.93	0.81	0.87
	100	–	–	–	‘’	‘’	0.91	0.94	0.84	0.88
	250	–	–	–	‘’	‘’	**0.97**	**0.99**	**0.93**	**0.99**
	500	–	–	–	‘’	‘’	0.97	0.99	0.93	0.99
	1000	–	–	–	‘’	‘’	0.97	0.99	0.93	0.99

Table 5
Task-identification results of the ablation study. ↑ and ↓ indicate the desired direction per measure.

Stream	S_{U1}		S_{U2}		S_{U3}	
Considered perspectives	#tasks	n.ED↓	#tasks	n.ED↓	#tasks	n.ED↓
Full approach (control-flow, semantic, & data)	**202**	**0.04**	**241**	**0.05**	**132**	**0.17**
Control-flow & semantic	**202**	**0.04**	**241**	**0.05**	150	0.27
Control-flow & data	302	0.35	338	0.33	**132**	**0.17**
Control-flow only	336	0.42	372	0.38	150	0.27

resp. 0.17. A further extension of the warm-up to 500 still improves the results, yet, less substantially (by 0.02–0.03). For S_{U3}, where all task types are covered by the first 100 events, performance improves as well. However, this improvement is not as considerable as observed for S_{U1} (macro scores increase by only 0.01). Notably, increasing the warm-up phase to 250 events has a more significant impact on categorization quality for S_{U3} (macro scores increase by 0.05–0.11). This may be attributed to the coverage of additional execution variants per task type, of which S_{U3} has significantly more compared to other streams (30 versus at most 7). Interestingly, for S_{U2}, performance remains consistently high regardless of whether all task types are seen during the warm-up phase or not. This suggests that, in some cases, it is not necessary to have observed all task types before starting categorization to achieve good quality.

Overall, these results show that a warm-up phase is not necessary for task-identification, but that it can be beneficial for task categorization, if an application context allows for it. However, even without any warm-up phase our approach achieves good categorization results across streams.

5.3.3. Ablation study

Our task-identification component uses various types of information associated with events to decide whether a task completes or continues (cf. Section 4.2). In order to understand the value of taking information beyond the control-flow into account, we conducted an ablation study. This involves, in turn, removing those task-identification strategies that consider data values (including object instances), semantic information (specifically overhead actions), and both of these.

We present the results of the ablation study in Table 5. It shows that we achieve the same identification performance on S_{U1} and S_{U2} when removing the data values from consideration, whereas the performance on L_{U3} decreases (the edit distance worsens from 0.17 to 0.27). Conversely, when not considering the semantic perspective, we obtain considerably worse results S_{U1} and S_{U2} (the edit distance worsens by ca. 0.3 for both streams), while achieving the same performance as when considering all perspectives on S_{U3}. Finally, the subpar results achieved when only considering control-flow information highlight the value of taking additional perspectives into account.

Overall, the results indicate that the consideration of specific perspectives is important for some streams and not for others. However, only when considering all perspectives can the overall good performance of our approach across streams be achieved.

5.3.4. Offline results

Table 6 shows the results of our approach applied in an offline setting compared to BL_{leno} and BL_{urabe}. For BL_{urabe}, the table shows the average results across the configurations that were evaluated in the original paper [8] as well as the results of the best configuration per log.

The results show that our approach also outperforms the offline baselines by a large margin. When it comes to task identification, it achieves an edit distance of just 0.04 for L_{U1}, while the baselines fall short; BL_{urabe} achieves only 0.28 at best and BL_{leno} achieves 0.35. For L_{U2}, BL_{urabe} achieves an edit distance of 0.28 in the best case and BL_{leno} achieves 0.26, while our approach achieves 0.05; a substantial improvement of 0.23 compared to BL_{urabe} and 0.21 compared to BL_{leno}. The performance gains are even more considerable for L_{U3}, where we observe an improvement of 0.49 compared to BL_{urabe} and 0.31 compared to BL_{leno}. This again highlights the efficacy of our task-identification strategy that considers control-flow, semantic, and data information, instead of solely relying on control-flow as done by the baselines.

Interestingly, BL_{urabe} achieves good macro-level results for task categorization on L_{U1} and L_{U2} (0.97–0.99) considering the subpar task-identification results. This indicates that the baseline's post-hoc categorization generally works well, despite poor task-identification quality. For L_{U3}, for which the tasks are more similar in terms of their event classes across types, the baseline's poor identification quality cannot be compensated by its good categorization performance, yielding macro Rand and Jaccard scores of 0.52 at best. In contrast, our approach achieves high macro (0.99–1.00) and micro (0.93–0.99) scores for all three logs. Note that BL_{leno} does not cover task categorization, for which we could, thus, not compute results.

Overall, both good task-identification and task-categorization quality are required to accurately abstract a user-interaction log to a task-level event log, which our approach provides across the evaluation logs.

Table 6
Results of our approach applied in an offline setting and the offline baselines, BL_{leno} and BL_{urabe}. Note that BL_{leno} only provides task identification and that, for BL_{urabe}, we show average results across the configurations used in the original paper [8] and the results from the best configuration per log. *Perf. ident.* shows task-categorization results in case of perfect task identification. ↑ and ↓ indicate the desired direction per measure.

Log	Approach	Obj. ident.			Task ident.		Task categorization			
		Pre.↑	Rec.↑	F_1 ↑	#tasks	n.ED↓	R(mi)↑	R(ma)↑	J(mi)↑	J(ma)↑
L_{U1}	*Ours*	0.91	0.88	0.90	**202**	**0.04**	**0.98**	**0.99**	**0.97**	**0.99**
	BL_{leno}	–	–	–	50	0.35	-	–	–	–
	BL_{urabe} (avg.)	–	–	–	163	0.56	0.72	0.78	0.59	0.64
	BL_{urabe} (best)	–	–	–	131	0.28	0.90	0.98	0.85	0.97
	Perf. ident.	–	–	–	200	0.00	1.00	1.00	1.00	1.00
L_{U2}	*Ours*	0.95	0.93	0.94	**241**	**0.05**	**0.98**	**0.99**	**0.97**	**0.99**
	BL_{leno}	–	–	–	101	0.26	–	–	–	–
	BL_{urabe} (avg.)	–	–	–	190	0.54	0.69	0.72	0.48	0.53
	BL_{urabe} (best)	–	–	–	161	0.28	0.89	**0.99**	0.80	**0.99**
	Perf. ident.	–	–	–	240	0.00	1.00	1.00	1.00	1.00
L_{U3}	*Our approach*	–	–	–	**132**	**0.17**	**0.97**	**1.00**	**0.93**	**1.00**
	BL_{leno}	–	–	–	90	0.48	–	–	-	–
	BL_{urabe} (avg.)	–	–	–	25	0.67	0.26	0.36	0.11	0.28
	BL_{urabe} (best)	–	–	–	18	0.66	0.26	0.54	0.11	0.52
	Perf. ident.	–	–	–	120	0.00	1.00	1.00	1.00	1.00

5.3.5. Performance in online versus offline settings

As shown in the previous sections, our approach achieves consistently high results in both online and offline settings. Object-instance identification achieves the same performance in online and offline settings because this component of our approach does not involve any training on historical data. Although our task-identification component uses training for its control-flow-based segmentation checks, our experimental results reveal that these are already accurate after having seen a limited amount of historical data or are compensated by the other checks in this component (see our ablation study in Section 5.3.3). By contrast, we do observe a slight decrease in task-categorization performance when applied in online settings, since this component uses historical data to train its clustering model.

For the approach by Leno et al. [7], we observe a clear performance drop when comparing the results of the adapted online version (BL_{dfg}) to the original approach (BL_{leno}), showing that it is more suitable for offline task identification. However, we do not observe the same trend for the approach by Urabe et al. [8], which also targets offline task identification and categorization. Although its offline version (BL_{urabe}) generally outperforms the adapted online version (BL_{co-oc}) in terms of task categorization, it is noteworthy that its online versus, on average, achieves better results for task identification.

Overall, the evaluation results highlight that, regardless of the specific setting, our approach achieves good performance in recognizing task-level events from user interaction data. It consistently outperforms the baselines in both online and offline settings and, unlike existing works, allows to relate recognized events to each other.

5.3.6. Results on individual task logs

The approach by Leno et al. [7] was originally proposed for offline segmentation of logs that record executions of a single task type. To also assess our approach's performance for this purpose, we applied it on the original task logs individually. We report on the task-identification results in comparison to the offline baselines in Table 7.

Our approach generally shows good performance in terms of task identification on the individual task logs as well, achieving edit distances between 0.01 and 0.29. It outperforms the baseline by Leno et al. (BL_{leno}) on five out of the eight logs. However, BL_{leno} outperforms our approach on specific logs, particularly 1, 5, and 6, where it perfectly identifies gold-standard tasks, whereas our approach does not identify some of them correctly. This highlights BL_{leno}'s capability of identifying repetitive tasks well, which is its main goal in the context of robotic process mining (cf. Section 6.2). In contrast, BL_{urabe} falls short, identifying substantially fewer tasks than contained in the gold standard. For logs 5–8, it did not identify any segmentation points, even in its best configuration (leading to a single identified task per log). This is to be expected because this baseline identifies tasks based on contextual relatedness. As the logs at hand capture only one type of task, context changes are often too subtle for BL_{urabe} to identify segmentation points correctly.

5.3.7. Computational efficiency

Finally, we assessed the memory and response time efficiency of our approach. To assess memory efficiency, we measure the maximum memory it requires, which is the sum of the largest buffer size during runtime, the final size of the global co-occurrence matrix, the directly-follows counts between event-class sets, and the final size of the clustering model. As for response time, we measure how long it takes our approach to perform object-instance identification and task identification after an event arrives, as well as how long it takes to categorize an identified task.

We find that our approach requires less than 1% of the memory that would be needed to store all events from the streams, thus clearly demonstrating its memory efficiency. As for response time, our approach requires between 1 and 107 ms, for object-instance identification, between 2 and 4 ms for task identification, and between 40 to 150 ms for task categorization. Note that the latter is only executed once per identified task. Therefore, the response time depends on a task's length, i.e., number of interaction events it consists of. Given that the average time between user interactions is over 2.5 s in the available data, this means that our approach can easily keep up in terms of responses.

6. Related work

Our work primarily relates to research on the identification of tasks from low-level event data (Section 6.1), robotic process mining (Section 6.2), the identification of object-centric information from event data (Section 6.3), and pre-processing techniques for stream-based process mining (Section 6.4).

6.1. Identifying tasks from low-level event data

Various approaches have targeted the identification of tasks in low-level event data. Focusing on user interaction events, various works [5, 18,19] take a supervised approach based on the computation of alignments between user interaction logs and task models that they require as input. Tello et al. also approach task identification in a supervised manner by applying classical machine learning approaches [20], whereas Pegoraro et al. train a neural network model to segment user

Table 7
Task-identification results of our approach applied in an offline setting and the offline baselines, BL_{leno} and BL_{urabe}, on the original task logs (cf. Table 2). The actual number of tasks per log is indicated in parentheses. Note that we use the original (non-sampled) task log 1 here. For BL_{urabe}, we show average results across the configurations used in the original paper [8] and the results from the best configuration per log. ↑ and ↓ indicate the desired direction per measure.

Task Log	1 (1000 tasks)		2 (50 tasks)		3 (50 tasks)		4 (40 tasks)	
Approach	#tasks	n.ED↓	#tasks	n.ED↓	#tasks	n.ED↓	#tasks	n.ED↓
Ours	**1000**	0.01	**50**	**0.02**	**50**	**0.02**	**40**	**0.05**
BL_{leno}	**1000**	**0.00**	**50**	0.04	51	0.16	80	0.50
BL_{urabe} (avg.)	40	0.85	38	0.41	24	0.74	69	0.70
BL_{urabe} (best)	76	0.79	29	0.33	14	0.60	39	0.49
Task Log	5 (30 tasks)		6 (30 tasks)		7 (30 tasks)		8 (30 tasks)	
Approach	#tasks	n.ED↓	#tasks	n.ED↓	#tasks	n.ED↓	#tasks	n.ED↓
Ours	**30**	0.20	42	0.29	**30**	**0.14**	**26**	**0.07**
BL_{leno}	**30**	**0.00**	**30**	**0.00**	93	0.67	60	0.50
BL_{urabe} (avg.)	1	0.96	1	0.96	1	0.96	1	0.96
BL_{urabe} (best)	1	0.96	1	0.96	1	0.96	1	0.96

interaction logs [6,21]. Linn et al. [22] combine transactional data recorded by information systems with user interaction logs, to integrate interaction data with traditional process mining. Finally, our earlier work [23] aims to recognize tasks through offline self-learning of multi-perspective dependencies between low-level interactions, although this currently requires expensive pre-training on large interaction logs.

Beyond user interaction events, related approaches aim to recognize process-related tasks from so-called active-window-tracking data [24], ambient or wearable sensors [25–27], network traffic data [28,29], or low-level, server-side application logs [30]. However, due to the low-level, abstract nature of the data used by these approaches, they depend on supervised recognition strategies or even manual labeling, as opposed to our unsupervised approach.

6.2. Robotic process mining

The core idea of robotic process mining (RPM) is to discover repetitive routines from user interaction logs, which are suitable for automation [31]. Leno et al. [2] propose an RPM-pipeline, which starts from a raw user interaction log and eventually yields automatable task scripts. The so-called *segmentation stage* of this pipeline identifies which user interaction events jointly form individual tasks (yet, not their types), thus only partially solving part (1) of the transformation problem that our approach addresses. We use the corresponding approach by Leno et al. [7] as a baseline for task identification in our evaluation. The *candidate-routine-identification stage* (also covered by Leno et al. [7]) recognizes tasks that are executed in the same or similar manner. In a sense, tasks are thus categorized into candidate routines, which may be considered as types. However, this stage does not assign labels to the tasks it identifies and only takes tasks into account if they are performed frequently, thus neglecting infrequent ones. Unlike candidate-routine identification, the approach by Urabe et al. [8] categorizes all tasks that it previously identified through segmentation, yet, does not assign task labels either, thus also only partially addressing part (1) of the transformation problem (but not part (2)). We lift the segmentation strategy of this approach to an online setting and use it as a baseline in our evaluation.

Although our work and works on RPM (partially) address similar sub-problems, their overarching goals are fundamentally different. RPM aims to get in-depth insights into the execution of individual tasks with the ultimate goal of automating suitable ones, whereas we aim for a comprehensive transformation of low-level user interactions into task-level events that are usable in process mining settings.

6.3. Identifying object-centric information in event data

Relating task-level events to each other by identifying to which case they belong has been addressed by several works [32]. The problem of inferring missing object information from event data as well as the conversion of classical event logs into object-centric logs have also been investigated [33,34]. Berti et al. propose approaches to extract object-centric event logs from SAP systems and relational databases [35]. Finally, the extraction of object-centric information from knowledge graphs has been researched [36]. However, these techniques assume that events are already on the task level and that they can operate in an offline setting, whereas our approach overcomes both of these assumptions.

6.4. Stream-based pre-processing of event data

Research on stream-based pre-processing in process mining mostly focuses on cleaning noisy event streams. Van Zelst et al. [37] filter a stream based on estimates of how likely new events belong to real process behavior, whereas Hassani et al. [38] filter noise by extracting frequent sequential patterns from an event stream before applying streaming process discovery. Finally, Awad et al. [39] propose an approach to resolve situations in which events arrive in an incorrect order on a stream. However, these techniques assume arriving events to be on the task level, even though, in practice, streaming data is commonly at a lower-level of abstraction, such as taken into account by our approach.

7. Conclusion

This section summarizes the results of our work (Section 7.1), discusses its main limitations (Section 7.2), and provides an outlook on future work (Section 7.3).

7.1. Summary

In this paper, we proposed an automated approach for recognizing task-level events from user interaction data, which works in a fully unsupervised manner. It segments user interaction data to identify tasks, categorizes these according to their type, and relates tasks to each other via object instances it extracts from the low-level events. In this manner, our approach creates task-level events that meet the requirements of process mining settings as they relate to a process-level activity and to a process execution.

We demonstrated our approach's efficacy in recognizing task-level events from user interaction data through an experimental evaluation and showed that it outperforms three streaming baselines and two offline RPM approaches on this task. Furthermore, we found that, in most cases, our approach also outperforms these RPM approaches in solving the task-identification problem they target. This shows the benefit of the semantic and data perspectives considered by our approach, which go beyond the control-flow perspective exclusively used by these existing works.

7.2. Limitations

Our work is subject to certain limitations, which relate to the approach itself and its evaluation.

As for our approach, a key limitation is that it assumes that tasks are executed sequentially, i.e., one task must be completed before another one is started. This is naturally a limiting assumption as it is well-imaginable that users switch between tasks in their daily work. Especially knowledge workers in office settings commonly work on several tasks in an interleaving manner [24]. However, it is important to remark that this limitation so far applies to all unsupervised approaches that recognize tasks from user interaction data, because it is highly challenging and may impose additional data requirements to be solved properly.

As for our evaluation, we acknowledge that the considered user interaction data may impact generalizability of the results. Although this data covers a variety of task types, was obtained from different sources, and goes beyond the evaluation data used in other work [8], it does not capture a user's real sequence of process-related and overhead tasks conducted during a workday. Therefore, we plan to conduct further experiments as soon as more suitable data becomes available.

Furthermore, the generalizability of the sets of completion actions and keywords for overhead actions (used during task identification) remains to some degree uncertain given the available data. While these sets are generic, stem from established design guidelines, and occur across different types of graphical user interfaces, we cannot guarantee their completeness for any user interaction stream or log. Moreover, as there is no comprehensive set of UI-objects, we created such a set (used during object-instance identification). This creation may have been biased through the knowledge of the evaluation data, yet, also includes a broad range of objects beyond those that occur in this data. However, all of these sets our approach employs can easily be adjusted and extended so that they, for instance, cover domain-specific applications and different languages.

7.3. Future work

We see several promising directions on how to improve our approach in future work. First, our approach currently does not consider event timestamps that may help make correct segmentation decisions, e.g., when there are large time differences between events [40]. Therefore, we aim to incorporate a strategy based on time differences into our task-identification component. Furthermore, while our object-instance-identification component can accurately identify process-related object instances, it cannot identify relations among their types. We plan to recognize higher-level semantic relations between object types, such as that an *address* belongs to a *customer* and that a *customer* places *orders*, to further refine the relations between task-level events. Finally, we aim to recognize task-level events from user interaction data when facing interleaving task executions. This may be supported through the detection of data values and object instances that allow us to infer inter-relations between non-consecutive events, yet, due to its challenging nature, solving this problem also requires entirely new conceptual developments.

CRediT authorship contribution statement

Adrian Rebmann: Writing – review & editing, Writing – original draft, Validation, Software, Resources, Methodology, Data curation, Conceptualization. **Han van der Aa:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The implementation, data collection, evaluation pipeline, and raw results can be found through our repository linked in Section 5.

References

[1] L. Abb, J.-R. Rehse, A reference data model for process-related user interaction logs, in: BPM, Springer, 2022, pp. 57–74.

[2] V. Leno, A. Polyvyanyy, M. Dumas, M. La Rosa, F.M. Maggi, Robotic process mining: vision and challenges, Bus. Inf. Syst. Eng. 63 (3) (2021) 301–314.

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

[4] S. Agostinelli, M. Lupia, A. Marrella, M. Mecella, Reactive synthesis of software robots in RPA from user interface logs, Comput. Ind. 142 (2022) 103721.

[5] A. Marrella, T. Catarci, Measuring the learnability of interactive systems using a Petri net based approach, in: Proceedings of the 2018 Designing Interactive Systems Conference, 2018, pp. 1309–1319.

[6] M. Pegoraro, M.S. Uysal, T.-H. Hülsmann, W.M. van der Aalst, Uncertain case identifiers in process mining: A user study of the event-case correlation problem on click data, in: BPMDS, Springer, 2022, pp. 173–187.

[7] V. Leno, A. Augusto, M. Dumas, M. La Rosa, F.M. Maggi, A. Polyvyanyy, Identifying candidate routines for robotic process automation from unsegmented UI logs, in: ICPM, IEEE, 2020, pp. 153–160.

[8] Y. Urabe, S. Yagi, K. Tsuchikawa, H. Oishi, Task clustering method using user interaction logs to plan RPA introduction, in: BPM, Springer, 2021, pp. 273–288.

[9] A. Burattin, Streaming process mining, in: Process Mining Handbook, Springer, 2022.

[10] A. Rebmann, H. van der Aa, Unsupervised task recognition from user interaction streams, in: International Conference on Advanced Information Systems Engineering, Springer, 2023, pp. 141–157.

[11] A. Bifet, R. Gavalda, G. Holmes, B. Pfahringer, Machine learning for data streams: with practical examples in MOA, MIT Press, 2018.

[12] L. Abb, C. Bormann, H. van der Aa, J.R. Rehse, Trace clustering for user behavior mining, in: ECIS 2022 Research Papers. 34, 2022.

[13] M. Honnibal, I. Montani, S. van Landeghem, A. Boyd, Spacy: industrial-strength natural language processing in python, 2020, https://spacy.io.

[14] IBM, Carbon Design System - Action Labels, 2022, URL https://carbondesignsystem.com/guidelines/content/action-labels/.

[15] F. Cao, M. Ester, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in: International Conference on Data Mining, SIAM, 2006, pp. 328–339.

[16] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: KDD, AAAI Press, 1996, pp. 226–231.

[17] S. Agostinelli, A. Marrella, L. Abb, J. Rehse, Mastering robotic process automation with process mining, in: Business Process Management - 20th International Conference, BPM 2022, MÜNster, Germany, September 11-16, 2022, Proceedings, in: Lecture Notes in Computer Science, vol. 13420, Springer, 2022, pp. 47–53.

[18] S. Agostinelli, A. Marrella, M. Mecella, Automated segmentation of user interface logs, in: Robotic Process Automation, De Gruyter Oldenbourg, 2021, pp. 201–222.

[19] S. Agostinelli, Automated segmentation of user interface logs using trace alignment techniques, in: ICPM Doctoral Consortium/Tools, 2020, pp. 13–14.

[20] G. Tello, G. Gianini, R. Mizouni, E. Damiani, Machine learning-based framework for log-lifting in business process mining applications, in: BPM 2019, Vienna, Austria, September 1–6, 2019, Proceedings 17, Springer, 2019, pp. 232–249.

[21] M. Pegoraro, M.S. Uysal, T.-H. Hülsmann, W.M. van der Aalst, Resolving uncertain case identifiers in interaction logs: A user study, 2022, arXiv preprint arXiv:2212.00009.

[22] C. Linn, P. Zimmermann, D. Werth, Desktop activity mining-a new level of detail in mining business processes, in: WS der INFORMATIK 2018-Architekturen, Prozesse, Sicherheit und Nachhaltigkeit, Köllen Druck+ Verlag GmbH, 2018, pp. 245–258.

[23] A. Rebmann, P. Pfeiffer, P. Fettke, H. van der Aa, Multi-perspective identification of event groups for event abstraction, in: Process Mining Workshops. ICPM 2022, Springer, 2023, pp. 31–43.

[24] I. Beerepoot, D. Barenholz, S. Beekhuis, J. Gulden, S. Lee, X. Lu, S. Overbeek, I. van de Weerd, J.M. van der Werf, H.A. Reijers, A window of opportunity: Active window tracking for mining work practices, in: ICPM, IEEE, 2023, pp. 57–64.

[25] L. Chen, J. Hoey, C.D. Nugent, D.J. Cook, Z. Yu, Sensor-based activity recognition, IEEE Trans. Syst. Man Cybern. 42 (6) (2012) 790–808.

[26] A. Rebmann, A. Emrich, P. Fettke, Enabling the discovery of manual processes using a multi-modal activity recognition approach, in: BPM Workshops, Springer, 2019, pp. 130–141.

[27] A. Rebmann, S. Knoch, A. Emrich, P. Fettke, P. Loos, A multi-sensor approach for digital twins of manual assembly and commissioning, Procedia Manuf. 51 (2020) 549–556.

[28] G. Engelberg, M. Hadad, P. Soffer, From network traffic data to business activities: A process mining driven conceptualization, in: BPMDS 2021, Springer, 2021, pp. 3–18.
[29] M. Hadad, G. Engelberg, P. Soffer, From network traffic data to a business-level event log, in: International Conference on Business Process Modeling, Development and Support, Springer, 2023, pp. 60–75.
[30] B. Fazzinga, S. Flesca, F. Furfaro, L. Pontieri, Process mining meets argumentation: Explainable interpretations of low-level event logs via abstract argumentation, Inf. Syst. 107 (2022) 101987.
[31] M. Dumas, M. La Rosa, V. Leno, A. Polyvyanyy, F.M. Maggi, Robotic process mining, in: Process Mining Handbook, Springer International Publishing Cham, 2022, pp. 468–491.
[32] K. Diba, K. Batoulis, M. Weidlich, M. Weske, Extraction, correlation, and abstraction of event data for process mining, Wiley Interdiscipl. Rev. 10 (3) (2020) 1–31.
[33] A. Swevels, R. Dijkman, D. Fahland, Inferring missing entity identifiers from context using event knowledge graphs, in: International Conference on Business Process Management, Springer, 2023, pp. 180–197.
[34] A. Rebmann, J.-R. Rehse, H. van der Aa, Uncovering object-centric data in classical event logs for the automated transformation from XES to OCEL, in: BPM, 2022, pp. 11–16.
[35] A. Berti, G. Park, M. Rafiei, W.M. van der Aalst, A generic approach to extract object-centric event data from databases supporting SAP ERP, J. Intell. Inf. Syst. (2023) 1–23.
[36] J. Xiong, G. Xiao, T.E. Kalayci, M. Montali, Z. Gu, D. Calvanese, Extraction of object-centric event logs through virtual knowledge graphs, in: 35th International Workshop on Description Logics, DL 2022, Haifa, Israel, August 7-10, 2022, 2022.
[37] S. van Zelst, M. Fani Sani, A. Ostovar, R. Conforti, M.L. Rosa, Filtering spurious events from event streams of business processes, in: CAiSE, Springer, 2018, pp. 35–52.
[38] M. Hassani, S. Siccha, F. Richter, T. Seidl, Efficient process discovery from event streams using sequential pattern mining, in: SSCI, IEEE, 2015, pp. 1366–1373.
[39] A. Awad, M. Weidlich, S. Sakr, Process mining over unordered event streams, in: ICPM, IEEE, 2020, pp. 81–88.
[40] G. Bernard, A. Senderovich, P. Andritsos, Cut to the trace! process-aware partitioning of long-running cases in customer journey logs, in: CAiSE, Springer, 2021, pp. 519–535.