Automata Learning versus Process Mining: The Case for User Journeys

Paul Kobialka, Andrea Pferscher, Bernhard K. Aichernig, Einar Broch Johnsen, Silvia Lizeth Tapia Tarifa

Abstract—With the servitization of business, understanding how users experience services becomes a crucial success factor for companies. Therefore, there is a need to include feedback from user experiences in the software engineering process. Behavioral models of user journeys, describing how users experience their interaction with a service, can provide insights and potentially improve services. In this paper, we investigate techniques that allow the automatic generation of behavioral models from user interactions with a service, recorded in an event log. We first compare two established techniques that generate behavioral models from a given event log: automata learning and process mining. Afterward, we present a novel, hybrid method that combines both automata learning and process mining methods to overcome their limitations. For the existing techniques, we present methods to learn models of user journeys and evaluate the accuracy of the resulting models. We then compare these techniques with our novel method for the automatic extraction of user journey models from the event logs of digital services. We assess the practical applicability of all techniques by evaluating real-world applications. Our results show that process mining techniques rely on expert knowledge, while automata learning techniques depend on the distribution of events in the given event log. We further show that the proposed hybrid technique combines the strengths of both process mining and automata learning, automatically selecting the best method and parameter settings for a given event log to learn very accurate models.

Index Terms—passive automata learning, process discovery, model inference, model learning, customer journeys, AALpy

1 Introduction

Software products increasingly rely on service-oriented business models [1], making user satisfaction a key factor in a business' financial success [2]. *User (or customer) journeys* [3] are models that describe how users experience a service. These models have been highly successful in explaining and improving services [4], but their construction and analysis typically require significant manual effort [5]. This paper investigates the automated, data-driven construction of behavioral models of user journeys, that are amenable to tool-driven analysis. We compare two techniques for learning such behavioral models from the system logs of services: *automata learning* (AL) [6] and *process mining* (PM) [7].

For user-centric services, it is important to feed insights about user behavior back into the continuous development process (e.g., [8]). These services are typically personalized [9], and changes to the service might directly affect user behavior. For the service provider, it is crucial to know why and in which steps of the service they lose users. Usually, if companies grow over time, so do the number of users and the variations in the service. Companies are confronted with evolving user expectations [9, 10]; their success depends on maintaining the quality of the constructed behavioral models. However, user journey models are usually generated and analyzed manually, typically with a few selected users through questionnaires, relying heavily on domain experts [11]. Thus, the creation of user journey models lacks both agility and scalability to large user bases. The process suffers from a lack of tool support and is inherently limited by the high amount of manual labor [5].

User journeys record and analyze the time sequence of interactions between a service provider and a user from the user's perspective. Users interact with a service to reach some goal, e.g., to order an item online. The steps of the user journey, called *touchpoints*, record actions or communications between a service provider and a user. Touchpoints are observable and stateful, which makes user journeys amenable to behavioral modeling. Behavioral models of user journeys describe user behavior using a finite-state representation, providing insights into an underlying user journey. A formal representation of the user behavior enables automatic analysis of properties, e.g., checking whether a user can achieve a certain task. In practice, the presence of such behavioral models is limited. Therefore, this paper investigates techniques to automatically construct behavioral models of user journeys.

Automatically constructed behavioral models of user journeys have recently been used to analyze weaknesses in service offerings by Kobialka et al. [12, 13]; the user journeys were formalized as transition systems, constructed from system logs using PM and AL techniques. Figure 1 illustrates the three-step procedure introduced by Kobialka et al. [12] to automatically generate behavioral models from logs, enabling advanced model-based analysis. In Step 1, we can either use AL or PM to create behavioral models from logs. Compared to manually constructed models, the approach enables the analysis of user journeys at a completely different scale; system logs may span from days to years of interaction between service and users, captured in possibly more than thousands of traces. Pain points in the offered service are automatically detected by (probabilistically) model checking [14, 15] the behavioral model in Step 2. Step 3 includes further visualization; e.g., *Sankey* diagrams [16] can then show us the actions most viable for improvements. The pipeline shown in Fig. 1 demonstrates that finite transition systems enable an in-depth analysis of user behavior. Analysis techniques such as model checking

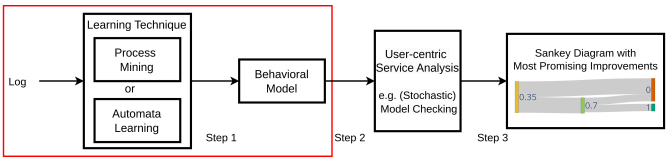


Fig. 1: Three step procedure for the creation and analysis of user journey models. This work focuses on Step 1, indicated by the red square, i.e., the automatic creation of behavioral models from event logs.

can identify bottlenecks in a service. Kobialka *et al.* [12] also use model checking to visualize user flows in a more understandable way for service providers, via Sankey diagrams.

In this paper, we focus on Step 1 in Fig. 1 and compare the two alternatives for constructing user journey models, using either PM or AL techniques. Both are well-established, but stem from different communities: PM techniques aim to gain insights into business processes, while AL techniques aim to derive the most general representation of a black-box system as a behavioral model. Nevertheless, there are similarities: *Process discovery* is a step in the PM pipeline that addresses the problem of finding a model representation from a given event log. *Passive automata learning* has a similar goal, but focuses on finding the most general representation. Recent developments in PM emphasize discovery techniques for rich behavioral models, e.g., Petri nets or process trees that allow the formalization of advanced system characteristics such as parallelism [17]. However, user journeys do not require such a feature-rich modeling formalism. As shown by Kobialka et al. [18, 19], finite transition systems model user journeys sufficiently and allow for rich analyses. Therefore, to enable realistic analyses, we compare PM and AL for learning behavioral models of user journeys.

Finite transition systems enable the application of AL techniques, which provide well-established algorithms [20, 21, 22] for learning a minimal finite-state representation of a given set of system traces. However, finding the right level of behavioral approximation defined by a generated model is *the most interesting challenge in process mining* [23]. The comparison of AL and PM techniques specifically addresses this challenge: AL targets the most general state representation in terms of overapproximation, whereas PM approaches an underapproximation. Our comparison between PM and AL shows that the accuracy of the generated model strongly depends on the underlying event log, where PM is beneficial for sparse event logs and AL for larger, well-distributed logs. In this paper, we propose a novel method, called HYBRID, that automatically applies either PM or AL depending on characteristics of the event log to obtain the most accurate model.

Contributions. The key contributions of this paper are: (1) a novel benchmark suite for the evaluation of user journey learning techniques, (2) an exhaustive comparison of different AL and PM learning setups based on the benchmark suite, (3) the novel method HYBRID that combines established AL and PM methods to extract behavioral user journey models in practice, (4) a practical evaluation that includes four different real-world case studies, and (5) a discussion of actionable insights from these experiments.

2 Related Work

Recent work on user journey modeling by Halvorsrud et al. [5, 24] emphasize the need for digital support. In their work, the *actual journeys*, reflected as user experience in the models, are manually discovered through interviews with users [4]. We here discuss previous work on automatic model generation for user journeys, based on PM and AL techniques. In contrast to all related work, our work focuses on *comparing* and *combining* PM and AL techniques. To the best of our knowledge, we are the first to propose a combination of AL and PM techniques.

Process discovery methods in PM enable a data-driven approach to generate user journey models. Positioning user journey mappings in the PM landscape, Bernard and Andritsos [25, 26, 27, 28] consider methods to abstract large numbers of user journeys into compact representations, including an XML format for user journeys [25], hierarchical clustering [26], user journey maps with different levels of granularity based on process trees [27], and a genetic algorithm to build representative user journeys [28]. Harbich et al. [29] discover user journey maps using mixtures of Markov models. Terragni and Hassani [30, 31] use PM tools to generate models of an underlying user journey for different user groups, and optimize towards specific key performance indicators. The BPI Challenge 2012 [32, 33] makes real-life event logs from a financial institution available for analysis. Verbeek [34] addresses the challenge by constructing multiple transition systems for different aspects of the event log, each with an adjusted representation. The results show the potential of analyzing transition systems to gain process-specific insights. We include the BPI challenge 2012 and 2017 [35] among the use cases for evaluating our approach. Kobialka et al. [18, 19, 13] develop PM techniques to generate and analyze weighted games that reflect the user experience from logs. This line of work demonstrates the usefulness of finite state automata for representing user journeys; hence, we use them in this paper. There are also process discovery techniques without an explicit model representation; e.g., declarative process discovery [36, 37] represents models by temporal logic formulae. These representations are not addressed in our work.

In AL, methods for passive learning can be used for event logs. The earliest work we know that uses AL for process models is by Cook and Wolf [38], who learn a model from software logs. Esparza et al. [39] proposed an active AL algorithm to learn automata from Petri nets. Their paper also discusses an extension to learning from event logs that requires comprehensive logs. Agostinelli et al. [40] evaluate AL methods for event logs and conclude that the active L^*-algorithm of Angluin [41] is not suitable for model generation, but passive algorithms perform well compared to the PM algorithm *Declare* of Pesic et al. [42], e.g., RPNI [20], MDL [43], and EDSM [44]. In contrast to this work, they assume that only traces of specific length should be included in the model. Kobialka et al. [12] also apply AL using the complete event log, but unlike this work, they learn stochastic weighted games. In a similar work, Johnsen et al. [45] show the capability of AL for dealing with large data sets, creating timed stochastic games for music streaming behavior. Wieman et al. [46] report on

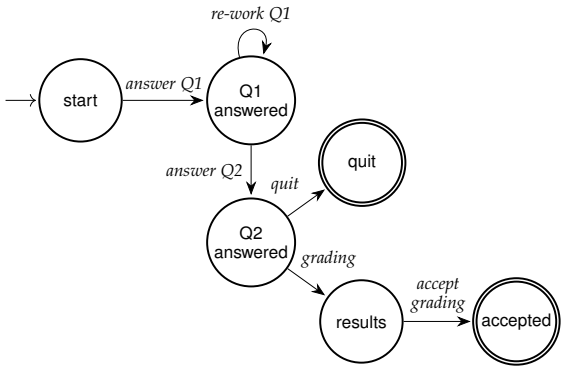


Fig. 2: Example of a transition system.

experiences from an industrial case study using passive AL to mine models of the developed software tool from test logs, and discuss practical applications of AL to improve software development. A similar empirical approach could be applied using learned user journeys to improve services.

Akin to user journeys, *software product lines* (or *software process lines*) define a base process from which multiple variations are derived. There are both PM [47, 48, 49] and AL [50] techniques to generate and analyze models of the underlying process. The PM techniques for software process lines have a similar setup as for user journey mining, i.e. using process discovery techniques to generate a model from a process log. However, the AL approaches differ: Damasceno *et al.* [50] extend AL techniques for software product lines by learning featured finite state machines over collections of product models. Software product lines are not addressed in our current work.

No related work compares PM and AL in terms of learning behavioral models of user journeys. Our paper addresses this gap by focusing on logs of varying sizes as observed in practice. No related work investigates whether learned behavioral models over- or under-approximate the given event log, and which parameter settings are adequate for user journeys. Our work examines how to set model parameters for PM and AL for sparse logs and proposes a novel HYBRID technique which applies PM on small and AL on large logs. We summarize the lessons learned from our experiments into practical insights for practitioners.

3 Preliminaries

We briefly introduce the required techniques from PM and AL, after providing some basic definitions. Given a finite set A of events, a *trace* $\sigma \in A^*$ is a finite sequential sequence over A, possibly with duplicates. An *event log* L is a multiset of traces in A^*. Let ϵ denote the empty trace. For a trace $\sigma = \langle a_0, \ldots, a_n \rangle$ with length $|\sigma| = n+1$, σ_i denotes the i^{th} event of σ (so $\sigma_i = a_i$) and $\sigma_{i:j}$ the subtrace $\langle a_i, \ldots, a_{j-1} \rangle$. If $i \geq j$, then $\sigma_{i:j} = \epsilon$. We abbreviate $\sigma_{i:|\sigma|}$ by $\sigma_{i:}$ and $\sigma_{0:i}$ by $\sigma_{:i}$.

A *transition system* (TS) [51, 52] is a tuple $TS = (\Gamma, A, E, I, T)$, where

- Γ is a finite set of states,
- A a finite set of events,
- $E \subseteq \Gamma \times A \times \Gamma$ the finite set of transitions,
- $I \subseteq \Gamma$ the finite set of initial states, and
- $T \subset \Gamma$ the finite set of final states.

A *run* in TS is an alternating sequence of states and events $\langle \mathsf{s}_0, a_0, \mathsf{s}_1, \ldots \rangle$ such that $(\mathsf{s}_i, a_i, \mathsf{s}_{i+1}) \in E$, starting from an initial state $s_0 \in I$. Let R denote the set of all runs in TS and $\pi(r) = \langle a_0, a_1, \ldots \rangle$ the projection of a run $r \in R$ to its corresponding trace of events. Let $\mathcal{L}(TS)$ denote the language induced by TS, defined by all runs that end in a final state, i.e., $\mathcal{L}(TS) = \{\pi(r) \mid r \in R \wedge \mathsf{s}_n \in T\}$. We call a trace $\sigma \in A^*$ *positive* iff $\sigma \in \mathcal{L}(TS)$, otherwise it is *negative*. TS is *deterministic* if $(\mathsf{s}, a, \mathsf{s}'), (\mathsf{s}, a, \mathsf{s}'') \in E \rightarrow \mathsf{s}' = \mathsf{s}''$. In the sequel, we assume a TS to be non-deterministic. Runs and traces highlight the difference between a transition system and an event log: in a run on the transition system both the events and the visited states are known, while the corresponding trace in the log only records events.

Example 1 (Transition System). In the transition system shown in Fig. 2, the initial state is indicated by an arrow with an empty source, and the final states are double-lined. A run in this transition system could be $r = \langle \mathsf{start}, \mathit{answer\ Q1}, \mathsf{Q1\ answered}, \mathit{answer\ Q2}, \mathsf{Q2\ answered}, \ldots \rangle$, with the corresponding trace $\pi(r) = \langle \mathit{answer\ Q1}, \mathit{answer\ Q2}, \ldots \rangle$.

We assume that user journeys have a sequential structure; i.e., users only engage in one action at a time and logs do not contain noise. With this assumption, transition systems suffice as underlying formalism for user journeys.

3.1 Process Mining

Process mining (PM) [7] supports the model-based analysis of processes. In PM, *process discovery (PD)* techniques extract models of processes, e.g., Petri nets, from a sample of process observations, e.g., event logs. Kobialka et al. [19, 13, 53] recently applied PD techniques to automatically generate behavioral models of user journeys from event logs, based on the *directly-follows graph* (DFG) construction of van der Aalst [54]. Well-established process discovery algorithms include, e.g., approaches based on genetic algorithms [55], *Inductive Miner* [56] for learning process trees, *Alpha Miner* [57] for learning workflow nets, and *Heuristics Miner* [58] for dealing with noise. We focus on DFGs as they represent a fundamental modelling technique in process discovery. DFGs are well-suited for user journeys as they are sequential and their state-based representations can be used for, e.g., model checking [12, 19].

We review PD techniques within PM to generate DFGs for user journeys. In short, PD consists of three phases: (1) *preprocessing*, which prepares the event log; (2) *construction*, which generates the behavioral model; and (3) *postprocessing*, which prepares this model for further analysis.

Preprocessing. The event log L is modified according to a target granularity. Typically, L is filtered for outliers (i.e., rare journey variations) by removing spurious traces. Rare events may be filtered out, but this can remove too many user interactions at this early stage of the discovery process [54]. Note that a preprocessed L is still an event log.

Construction. Commonly, a DFG is a TS $G = (\Gamma, A, E, \{s_0\}, T)$ where $\Gamma = A \cup \{s_0\}$, transitions connect consecutive events, so two states $a, b \in \Gamma$ are connected, $(a, b, b) \in E$, if the log contains a trace where $a \in A$ is

directly followed by $b \in A$. The state s_0 is a designated initial state with outgoing transitions to the first event of the traces in the log, and T contains the observed last states in traces (or designated final states introduced in the preprocessing). For process analysis, the DFG can be used to highlight the most common behavior [54]. *Directly follow systems* (DFS) introduce richer state representations than DFGs by considering additional context and different context representations [23]. DFSs have been used to model the interactions between service providers and users [18, 19, 13].

Postprocessing. This phase consists of *filtering* rare transitions and *completing* the constructed DFS. The filtering of rare transitions can be done in terms of their frequency in the log, adjusting the neighboring transitions traversed by the traces iteratively if needed to preserve model soundness [54, 59]. Completing a DFS introduces common process patterns that are not found in the log; e.g., enabling event interleavings or alternative executions can be done by closing "diamond" patterns and resolving self-loops [23].

3.2 Automata Learning

AL aims to learn a minimal automaton from a finite set of system traces. AL stems from the problem of learning an unknown regular language from a set of traces. AL can be active or passive, depending on the set of traces that are actively generated by interacting with the system under learning (SUL) during learning or (passively) given traces. For user journeys, we cannot generally assume access to a SUL. Hence, we discard active AL and use passive learning to generate a behavioral model from event logs. Gold [60] showed that learning a regular language requires positive and negative traces, where positive traces are those accepted by the language and negative traces are those that are not accepted. However, Angluin [61] showed that the underlying stochastic distribution of events can be used to learn models from only positive traces. Since event logs for user journeys only contain positive traces, we learn probabilistic models. Alergia [21] is a state-merging algorithm for learning behavioral models of probabilistic systems from positive traces, where state-merging is a passive AL technique for learning behavioral models of black-box systems from a set of traces.

Let $\mathcal{L}(\mathrm{SUL}) \subseteq A^*$ be the language defined by the SUL. Given a set of traces $L \subseteq \mathcal{L}(\mathrm{SUL})$, an AL algorithm constructs a *frequency prefix tree acceptor* (FPTA) $\mathcal{T}$, where states in the tree represent events in the traces of L. Let $\sigma \ll \sigma'$ denote the reflexive prefix relation on traces, expressing that σ is a prefix of σ'. The FPTA $\mathcal{T}$ is constructed such that $\sigma \ll \sigma'$ holds for every pair of traces, where $\sigma \in A^*$ is a prefix generated from the root of $\mathcal{T}$ and $\sigma' \in L$. Transitions in $\mathcal{T}$ are labeled by the number of traces from L with the same prefix in $\mathcal{T}$.

AL merges states of similar events in $\mathcal{T}$ by assuming the Markov property; i.e., subsequent events depend only on the current event. State merges are based on an *evaluation* function that allows to assess whether states can be merged, but also to create a ranking of possible merge candidates.

Alergia [21] is a well-established passive AL algorithm based on state-merging. Alergia's evaluation function for merging states uses the Hoeffding bound [62]: two states s, s' in $\mathcal{T}$ differ if, for any event $a \in A$,

$$\left| \frac{f_a^s}{n_s} - \frac{f_a^{s'}}{n_{s'}} \right| > \sqrt{\frac{1}{2} \log \frac{2}{\alpha}} \left(\frac{1}{\sqrt{n_s}} + \frac{1}{\sqrt{n_{s'}}} \right) \tag{1}$$

holds, where $n_s \in \mathbb{N}$ is the sum of the frequencies on the outgoing transitions from state s and f_a^s the frequency of the current event a in s. The parameter $\alpha \in (0, 1]$ expresses the confidence that the distribution of events in the log reflects the system's behavior. An event log is *well-distributed* if the events in the log are statistically significant, i.e., the traces in the log follow the underlying unknown distribution of events of the SUL. High values for α express low confidence in the distribution of the event log and thus favor little merging in the FPTA, while low values favor eager merging. After state merging, the transition frequencies are converted to probabilities, reflecting the proportions of the sum of frequencies on the outgoing transitions. The result is a Markov chain [63] (see Fig. 3a). To obtain a TS from the learned Markov chain, we then erase the probabilities on the transitions and extend the transition relation by the event of the target state. In the generated TS, all states are considered final.

4 Mining Behavioral Models of User Journeys

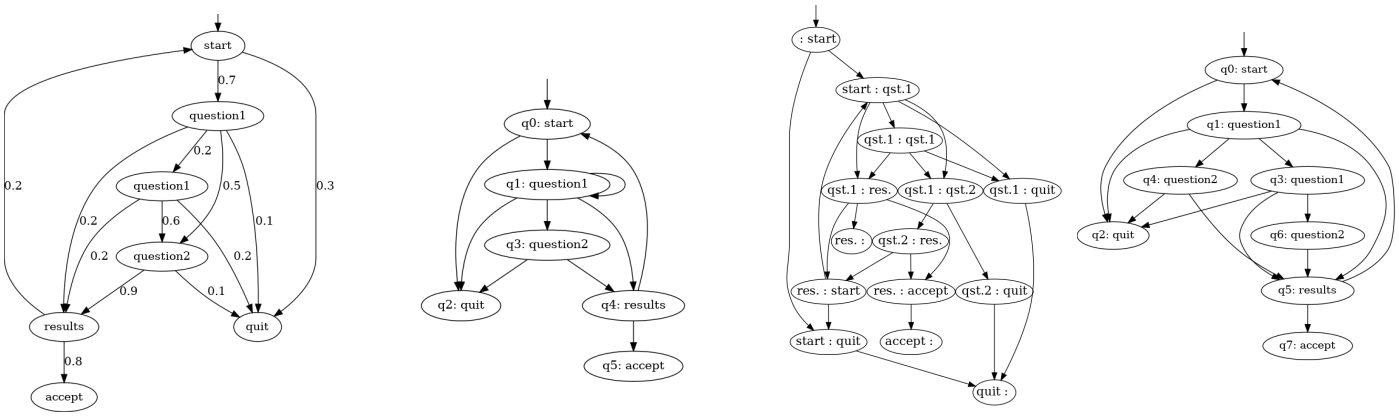
This section demonstrates by example how behavioral models for user journeys can be learned using methods from AL or PM. We also show how, depending on their parameter settings, both methods can generate the underlying model.

Example 2 (Assessment System). GrepS is a company that offers programming skill assessments as a service. The generation of user journey models from the event logs of GrepS has previously been studied by Kobialka et al. [18]. This example presents a simplified version of a user journey model, based on the GrepS service. Figure 3a shows a Markov chain of user journeys, where users are asked to answer questions to obtain a programming skill evaluation. A user journey always begins with a start event. After attempting to answering Question 1, the user can improve their answer once by repeating Question 1, or move to Question 2. Question 2 cannot be repeated. Users may quit the evaluation process or request their results at any point after starting the evaluation. After receiving the results, the user accepts them or repeats the whole questionnaire. A user journey is considered successful if the user accepts the evaluation results and unsuccessful otherwise. The Markov chain includes probabilities for subsequent events.

Example 2 already illustrates the challenge of automatically generating user journey models, since the example includes looping behavior and repeating events. Furthermore, the example also shows that user behavior may differ.

4.1 Event Logs for User Journeys

User journeys are goal-orientated processes, where users interact with a service provider to reach a certain goal, e.g., receiving a programming skill evaluation. Event logs for



(a) Ground-truth Markov chain of the assessment system (Example 2).

(b) TS from both high-confidence AL ($\alpha = 0.1$) and PM-generated DFG ($S^{[t]_1}$).

(c) PM-generated DFG with 1-event trace suffix ($S^{[t]_1}_{[h]_1}$).

(d) TS from low-confidence AL ($\alpha = 0.9$).

Fig. 3: Markov chain, learned PM and AL models of an assessment system. Different state representations (for PM) and confidence values (for AL) are used to learn models from a log with 80 traces. For simplicity, we omit transition labels.

user journeys record all interactions with different users, i.e., both successful (when the goal is reached) and unsuccessful user journeys. Therefore, a trace in an event log is a sequence of observable events that reflect a user's interaction with the service provider or actions along the user journey.

Example 3 (Event Log for the Assessment System). A possible event log for user journeys derived from Example 2:

$\langle start, question1, question2, results, accept\rangle$ (UJ-01)

$\langle start, question1, results, accept\rangle$ (UJ-02)

$\langle start, question1, question1, quit\rangle$ (UJ-03)

The traces (UJ-01) and (UJ-02) are successful user journeys, indicated by *accept* as their last event. (UJ-03) is an unsuccessful journey, where the user quits early.

4.2 Process Mining for User Journeys

We extend the DFG construction from a preprocessed event log L (Sect. 3, Phase 2) to *directly follows systems* (DFSs) by means of two functions: *trace projection* and *structure projection* [23, 7]. The purpose of the trace projection is to create a trace abstraction, while the purpose of the structure projection is to map the abstracted trace to states of a TS. Together, the two functions define the *state representations*.

Trace projections commonly (1) filter the horizon of the trace, e.g., a sliding window abstraction and (2) filter events, i.e., decide to ignore certain events. *Structure projections* define state representations from (abstracted) traces. Structure projections typically map traces to *lists*, *sets*, or *multisets*, thereby deciding how the underlying states are generalized: Lists reflect the traces as observed, multisets abstract from the order of the events, and sets abstract from their frequency. Formally, the monoid $(M, \otimes)$ over events $A \subseteq M$ with neutral element 0 defines a *structure projection* $\oplus : A^* \to M$ for trace $\sigma = \langle a_0, \ldots, a_n\rangle$, with $\oplus(\epsilon) = 0$, and $\oplus(\sigma) = \otimes(\oplus(\sigma_{0:n}), a_n)$. *State representations* [23] define states in a TS from (abstracted) traces by composing the trace and structure projections.

Example 4 (State representations). Let $\mathrm{head}_k : A^* \to A^*$ be a trace projection that maps a trace $\sigma = \langle a_0, \ldots, a_n\rangle$ to its first k events $\sigma_{:k}$, ignoring subsequent events (and assuming $k \leq n$). Similarly, let $\mathrm{tail}_k : A^* \to A^*$ map σ to the tail $\sigma_{\max(0,n-k):}$, ignoring (up to) the first k events. We define structure projection $\oplus$ on the monoid $(M, \otimes)$ of lists, where $A \subseteq M$ and $\otimes$ is list concatenation. A trace $\sigma = \langle a, b, a\rangle \in A^*$ is mapped by $\oplus(\sigma)$ to the list $[a, b, a]$ by concatenating single events. The state representation $[t]_k$, which composes the trace projection tail_k with $\oplus$, only considers the list of the last k events, i.e., states with the same last k events in sequence are considered equal. The state representation $[h]_k$, which composes the trace projection head_k with $\oplus$, equates traces if the next k events are the same, resulting in a k-step look-ahead state representation. By mapping to the monoid of sets, we obtain the state representation $\{t\}_k$ which ignores the position and frequency of the last three elements, e.g. $\{t\}_3(\sigma) = \{a, b\}$.

A DFS can be efficiently constructed by applying the DFG construction (see Sect. 3) to state representations. While states in a DFG only depend on the previous event in the trace prefix, states in the DFS depend on state representations for the trace prefix and suffix over k events, respectively. Thus, the state representation becomes sensitive to both past and future events (see Example 4). Depending on the state representation for the trace suffix, the DFS might contain several initial states.

Definition 1 (DFS from state representations). Let L be an event log over events A and let r and r' be state representations. A *directly follows system* (DFS) over L that uses r as a prefix and r' as a suffix representation, is a transition system $S^r_{r'} = (\Gamma, A, E, I, T)$ such that

- $\Gamma = \{(r(\sigma_{:k}), r'(\sigma_{k:})) \mid \sigma \in L, 0 \leq k \leq |\sigma|\}$,
- $A = \{a \mid (s, a, s') \in E\}$,
- $E = \{\langle (r(\sigma_{:k}), r'(\sigma_{k:})), \sigma_{k+1}, (r(\sigma_{:k+1}), r'(\sigma_{k+1:}))\rangle \mid \sigma \in L, 0 \leq k \leq |\sigma|\}$,
- $I = \{(r(\epsilon), r'(\sigma)) \mid \sigma \in L\}$, and
- $T = \{(r(\sigma), r'(\epsilon)) \mid \sigma \in L\}$.

DFSs can be constructed similarly to DFGs after constructing the states, by applying the state representation

functions to every event $\sigma_i \in \sigma$ for all traces in a log L. While the state computation must be applied to all σ_i, it can be efficiently computed as the horizon used in the trace projections is usually small, e.g., a finite number of previous events. Further improvements like filtering for unique traces and efficient log representations can further speed-up this computation.

Example 5 (PM for the Assessment System). We consider different state representations for an event log with 80 traces, generated by sampling the Markov chain from Example 1, where an event is sampled with the probability of its associated transition. Thus, traces in the log have different probabilities, to reflect event logs of user journeys. We select the length of the traces uniformly at random, between two and 15 events. Independent of the selected maximum length, a user journey stops after an *accept* or *quit* event. Every trace starts with an initial *start* event. Figure 3 presents DFSs for different state representations for the survey system: Fig. 3b shows the DFG, only considering the most recent event (technically, the state representation $S^{[t]_1}$). Note that we omit transition labels, but they can be obtained from the state labels as shown in Fig. 2. Figure 3c presents the DFS considering the current event and the next event (i.e., $S^{[t]_1}_{[h]_1}$). Observe that the DFG ($S^{[t]_1}$) does not recognize that the initial question can only be answered once, but displays it as a self-loop; i.e., users cannot improve their answers arbitrarily often. In contrast, the DFS $S^{[t]_1}_{[h]_1}$ covers the original model and displays that the answer to question 1 can only be improved once, thereby covering the different possibilities to answer the two questions.

DFSs satisfy the following properties: (1) every DFG is a DFS, (2) there are no dead transitions or states, i.e., every state can reach a final state, and (3) all log traces are traces of the generated DFS.

4.3 Automata Learning for User Journeys

We generate behavioral models of user journeys from event logs using passive AL. Classic passive learning techniques require a log with traces that are part of the language of the system under learning as well as traces that are not part of the language. [64]. Agostinelli et al. [40] compared different AL algorithms for mining *deterministic finite automata* of business processes. Their work separates the event log into traces that are in the language and traces that are not, according to trace length. For learning behavioral models of user journeys, this approach is less suitable since later model analysis might be interested in investigating the length of user interactions. Classifying according to the success of the user journey could also limit the insights gained, as user journey models should also allow conclusions to be drawn about the reasons for not achieving a goal. Thus, we want to learn a behavioral model whose language includes all traces of a given event log.

Angluin [61] observed that passive learning from only positive examples is possible when considering the probability distribution of the underlying event log. AL algorithms of this kind generate a Markov chain, which can then be translated to a TS by erasing the transition probabilities, as described in Sect. 3. Alergia [21] is an example of a state-merging algorithm that evaluates possible state-merges according to the Hoeffding-bound. For this, Alergia assumes that the provided log follows a certain probability distribution. However, this introduces the challenge of determining whether a given event log contains enough user journeys to distinguish states when applying AL for learning user journey models.

The Hoeffding-bound (Eq. (1)) considers α as a parameter to steer our confidence in the event log following a certain distribution. The parameter α is an arbitrary small real within $(0, 1]$ defining the error, such that a higher α expresses less confidence in the distribution of the underlying event log. For learning user journeys, we prefer a cautious state merging, i.e. a higher α, based on the following assumptions about user journeys. First, we assume that user behavior implements a regular language. The regular language defining a user journey includes in contrast to classical language identification problems, e.g., Tomita grammars [65], at least one order of magnitude more possible events in the considered alphabet. Second, user journeys frequently follow a strict sequential structure of events towards achieving the desired goal. This might be explained by the fact that the degree of freedom of actions taken by the user to interact with a service provider is usually limited and that following events are only useful for achieving a goal after a certain sequence of previous events.

Example 6 (AL for the Assessment System). Figure 3 depicts models learned by Alergia with different values for α. The event log is generated from the assessment system of Example 1. We evaluated a higher confidence ($\alpha = 0.1$), Fig. 3b, and a lower confidence ($\alpha = 0.9$), Fig. 3d, on the log with 80 traces from Example 5. Alergia generates Markov chains, which we translate into TSs. The results show that learning with $\alpha = 0.1$ (Fig. 3b) generates an overapproximation, i.e., the model describes a more general language than the ground truth. The reason is that low α promotes state merging since more states are similar according to Eq. (1). Note that this model is the same as the DFG generated in Example 5, which does not distinguish the two Question 1 events. By an increase to $\alpha = 0.9$, Alergia merges fewer states. The corresponding model (Fig. 3d) is an underapproximation; i.e., the model cannot produce all traces observable in the ground truth.

Example 6 shows that different experimental setups influence the resulting models. Thus, Examples 5 and 6 demonstrate the impact of the event log on both PM and AL. For PM, creating a DFG might lead to an overapproximation; however, selecting the right size for trace prefix and suffix requires expert domain knowledge. Therefore, PM techniques usually apply preprocessing to reduce and adapt the log. For AL, a correct model requires the right assumption about the underlying distribution of events. Otherwise, AL may learn an over- or underapproximation. The degree of generality of the model depends strongly on the underlying event log, and poses a challenge for finding appropriate parameter settings for the learning algorithm. We address these challenges in Sect. 5.

5 A Hybrid Learning Method Combining PM and AL

We introduce a novel learning method, called HYBRID, that combines AL and PM techniques to overcome the limitations of these techniques with respect to the quality of the underlying event log. This technique automatically selects the best suited method, depending on the characteristics of the log. HYBRID is a general method independent of specific PM and AL methods. It can be instantiated with any PM and AL methods that allows to infer a TS from an event log. In the following, we present HYBRID based on the PM and AL methods introduced in Sect. 4. By design, the PM method (Sect. 4.2) risks to create an underapproximation on large logs, as the selected state presentation might not enable a sufficient generalization of the behavior. In contrast, AL (Sect. 4.3) tends to learn an overapproximation, as AL techniques are designed to create the minimal automaton by merging as many states as possible. This might lead to overly general models, due to the sparsity of the provided event log. HYBRID will be validated by the experiments in Sects. 6 and 7.

HYBRID applies PM on small and AL on well-distributed event logs. The idea is to switch from PM to AL if the event log is sufficiently well distributed. For AL, state merging is guided by the confidence parameter $1 - \alpha$. Therefore, HYBRID aims to automatically switch from PM to AL if confidence in the event log being well distributed is high enough. This decision is implemented in a *certainty threshold* λ such that the model is constructed using PM for $\lambda > \alpha$, otherwise AL is used.

To estimate the certainty threshold λ, we propose a function λ_{approx} based on three different aspects of an event log L that influence our confidence α: (1) the number of different traces in an event log L [17] (i.e., the number of trace variants recorded in L: $\text{var}(L) = |\{\sigma \in A^* \mid \sigma \in L\}|$, where $\text{var}(L)$ is logarithmically scaled) and (2) the event log size $|L|$. We further introduce (3) a coefficient C_0 to adapt the function in the presence of domain knowledge. Let λ_{approx} be defined as follows:

$$\lambda_{\text{approx}} = \frac{C_0 \cdot \log_{10}(\text{var}(L))}{|L|}. \tag{2}$$

For user journeys, we set $C_0 = |A|$, accounting for the number of distinct events. Observe that for logs with a fixed event set size $|A|$, λ_{approx} shrinks rapidly in the size of the log, thus, returning a clear decision point between AL and PM. The intuition behind Eq. (2) is as follows: Small log sizes result in low confidence that the log is adequately distributed, especially for large event set sizes. In this case, we target the usage of PM techniques. However, we want to switch to AL as early as possible to avoid underapproximations caused by PM. By letting λ_{approx} shrink rapidly and α decrease with increasing log sizes, HYBRID is able to learn behavioral models of user journeys to avoid overapproximations as explained in Sect. 4.3.

When selecting α values for learning user journey models with AL, our goal is to find parameters for the AL setup that avoid both overapproximation and underapproximation in the learned models. Investigating the impact of different α values on AL, Mao et al. [66] show that with a sufficiently large event log L, $\alpha = \frac{1}{N}$ can be a good approximation, where $N = \sum_{\sigma \in L} |\sigma|$. However, we cannot generally assume that event logs for user journeys are large enough to use this approximation. Therefore, we propose an alternative α setup to what is normally done for AL, and adopt a sigmoid function

$$\alpha_{\text{approx}} = 1 - \frac{1}{1 + e^{(-|L| + C_1)\, C_2}}. \tag{3}$$

In this function, the constant C_1 shifts the inflection point, and the coefficient C_2 adapts the slope. By choosing C_1 and C_2 appropriately, the function allows an α-selection based on the size of L. The α-values that are close to 1 for very small logs L and approach $\frac{1}{N}$ for larger L.

Example 7* (α*-Approximation for AL).
For the event log with 80 traces in Example 5, Eq. (3) calculates $\alpha_{\text{approx}} = 0.49$ by setting the coefficients according to the event set size $|A| = 6$, where $C_1 = 10 \cdot |A|$ and $C_2 = \frac{1}{100 \cdot |A|}$. When using $\alpha_{\text{approx}} = 0.49$ for α, AL learns a TS which is equivalent to the TS representation of the Markov chain shown in Fig. 3a. Note that with the standard approximation $\alpha = \frac{1}{N}$ and this event log, the resulting value for α would be lower than 0.1, which would lead to an overapproximation similar to the one shown in Fig. 3b.

6 Experiments on Synthesized Benchmarks

In this section, we evaluate our hybrid method and compare it to different AL and PM configurations. For this purpose, we develop a novel benchmark suite for user journeys, which allows comparisons with "ground-truth" models. To this aim, the benchmark suite includes synthesized behavioral models of user journeys, represented as process trees, and the corresponding event logs sampled from the process trees. For AL, we evaluate different setups for the parameter α. For PM, we investigate different abstractions and state representations. By comparing with the ground-truth models, we can evaluate the precision and recall of the user journey models generated by AL and PM and assess the degree of over- or under-approximation. In this section, we answer the following research question:

RQ1: *How well do behavioral models learned by AL and PM techniques represent ground-truth models?*

6.1 Experimental Design and Setup

We implemented our evaluation framework in Jupyter Notebooks, using Python 3.10.12. The generated benchmark set and the code is available online.[1] Due to the numerous experiments, we distributed the executions over (1) a laptop with 32 GB memory and an i7-1165G7 @ 2.8 GHz Intel processor, and (2) a workstation with 128 GB memory and a 64-core AMD EPYC processor.

Synthesizing User Journeys. We aim to synthesize behavioral models of user journeys, therefore, our generation incorporates structural elements that can be found in real-world user journeys. The PM library PM4PY [67] enables the randomized creation of models such as Petri

1. https://figshare.com/s/3acae9725dda09cce811

nets. PM4PY implements also a generator for process trees as proposed by Jouck and Depaire [68]. Process trees are inherently sound, can be generated stepwise, and can be restricted to the same languages as transition systems [69]. The process tree generator allows the regulation of the creation of behavioral characteristics via parameters. For the creation of user journeys, we consider the following four control-flow features: (1) sequential structures, (2) branch statements, (3) loops, and (4) duplicate event symbols. We either enable or disable these features. Considering all parameter combinations, we generate 14 different setups (excluding the setup where all features are disabled and only duplicate events are allowed). For all generated process trees, we disable certain behavioral concepts that do not apply to user journeys, e.g., parallelism and silent transitions since user behavior is assumed to be sequential and observable. Furthermore, we align the size of the event alphabet to the real-world examples presented in Sect. 7, where the number of different event symbols ranges between 20 and 30. From each generation setup, we use PM4PY to randomly generate 10 process trees, leading to 140 generated trees.

Synthesizing Event Logs. PM4PY enables the creation of event logs from process trees. A trace in the event log is generated by traversing through the process tree, where the generator selects the next event uniformly at random when several paths are enabled. To evaluate the impact of event log sizes, we create event logs for each process tree with the following number of traces: 10, 50, 100, 200, 500, and 1 000. For each event log size, we repeat the generation 10 times, which sums up to 8 400 generated event logs in the benchmark set.

Process Mining Setup. For PM, we implement the DFS construction from Def. 1 by reducing traces to the considered length of past (prefix) and future (suffix) events and choosing a state representation from set, list, and multi-set for pre- and suffixes independently. We learn DFS in three settings: (1) PM-DFG, a DFG $S^{[t]_1}$, (2) PM-LONG, a four-element prefix and suffix DFS $S^{[t]_4}_{[h]_4}$ and (3) PM-UJ, a model imitating domain knowledge by generating a DFG if it is known that no events appear twice. To choose one DFS S among the possible configurations, we evaluate the number of loops, loops_S, and states, $|\Gamma_S|$, and aim to minimize $\text{loops}_S^2 + |\Gamma_S|$, thereby squaring the number of loops to favour simpler models. Counting loops might be computationally expensive, so we search for DFS with $\text{loops}_S^2 + |\Gamma_S| < 10^6$. If no such model is found, we default to a DFG.

Automata Learning Setup. For AL, we use the implementation of Alergia for learning Markov chains in the Python library AALPY [70], version 1.4.3. The learning results for Alergia depend on the parameter α, which regulates confidence about the underlying distribution of the log. To evaluate the impact of α, we learn TS with three different α-setups: (1) AL-0.1, higher confidence $\alpha = 0.1$, (2) AL-0.9, lower confidence $\alpha = 0.9$, (3) AL-UJ, $\alpha = \alpha_{\text{approx}}$ using the approximation function in Eq. (3), with coefficients according to the alphabet size $|A|$, $C_1 = 10 \cdot |A|$ and $C_2 = \frac{1}{100 \cdot |A|}$.

Hybrid Learning Setup. For HYBRID, we switch between PM and AL based on the decision threshold λ_{approx}. We calculate λ according to λ_{approx} in Eq. (2) with $C_0 = |A|$. When PM is selected, we use PM setup (3) PM-UJ. In the case of AL, we apply the AL setup (3) AL-UJ. In both cases, we assume that these setups are the most accurate for each method as they are designated to generate user journey models from event logs.

TABLE 1: Average scores for all learning setups.

	Precision		Recall		F-Measure	
	avg	std	avg	std	avg	std
AL-0.1	0.930	0.123	**0.935**	0.152	0.919	0.123
AL-0.9	0.952	0.098	0.929	0.153	**0.929**	0.118
AL-UJ	0.943	0.108	**0.935**	0.152	**0.927**	0.119
PM-DFG	0.869	0.174	**0.935**	0.152	0.880	0.136
PM-LONG	**0.999**	0.001	0.661	0.321	0.743	0.273
PM-UJ	0.921	0.150	0.898	0.216	0.880	0.183
HYBRID	0.950	0.107	0.900	0.217	0.899	0.177

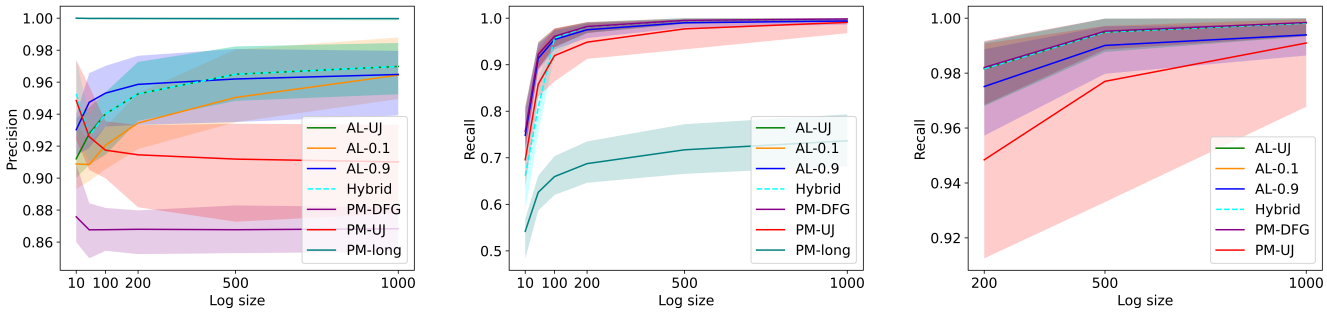
Evaluation Setup. The language difference between the generated model TS and the ground-truth model TS_{SUL} is evaluated using the language comparison technique for TSs proposed by Walkinshaw and Bogdanov [71]. For this, we create a confusion matrix [72] based on a finite test suite $\mathcal{S} \subset A^*$. For example, we classify a trace $\sigma \in \mathcal{S}$ as *false negative (FN)* if $\sigma \notin \mathcal{L}(TS)$ and $\sigma \in \mathcal{L}(TS_{\text{SUL}})$. We then calculate *precision*, *recall*, and the *F-measure* (the harmonic mean between precision and recall) following the standard definitions [71]. The generated test suite provides transition coverage for TS_{SUL} and TS. A test sequence can be written as a triple $(p, a, s) \in \mathcal{S}$, where $p \in A^*$ is a sequence leading to a state, $a \in A$ is the event of currently considered transition, and $s \in A^*$ is a randomly generated suffix of length $[0, n_{\text{len}})$. For each transition, we generate n_σ traces. For the evaluation, we set n_σ to the difference in the number of states between TS_{SUL} and TS, but at least to two. We set $n_{\text{len}} = 2$ since the probability of generating true negative traces is high.

6.2 Results

Figure 4 shows the results of the learning setups for the generated benchmark sets, for a total of 58 800 learned TSs. Table 1 shows the aggregated values. The precision results for the AL experiments, Fig. 4a, show that precision increases with the size of the event log, especially for AL-0.1 and AL-UJ. For the recall, Figs. 4b and 4c, AL-0.9 tends to underapproximate the log, since it is below AL-0.1 and AL-UJ. The function α_{approx} for AL-UJ creates a good approximation for learning models.

For PM, the DFG already contains all possible transitions on small event logs. Therefore, it does not improve with more samples. Compared to the DFG, the models generated with AL-0.1 overapproximate less, since the precision of AL-0.1 is higher. The results show that considering a long prefix of events, PM-LONG models create an underapproximation, i.e., the models overfit the traces in the event log. The PM-UJ technique based on domain knowledge achieves the best approximation for all considered PM techniques, yet tends to learn an overapproximation compared to AL.

For HYBRID, we observe that selecting the setups PM-UJ and AL-UJ has been the right choice, as both setups are best within their learning technique. Furthermore, switching between PM and AL based on λ_{approx} provides a well-balanced solution. In Fig. 4a, we can clearly see that HYBRID



(a) Average precision (b) Average recall (c) Average recall (zoom $200 - 1000$); here, AL-0.1 and PM-DFG overlap

Fig. 4: Results for the six evaluated learning setups on the synthesized benchmark set.

overcomes the limitations of AL and PM. For small log sizes, HYBRID adapts PM-UJ and, therefore, avoids overapproximating with AL-UJ. However, we switch exactly at the intersection of PM-UJ and AL-UJ, avoiding underapproximations as shown in Fig. 4b.

To answer **RQ1**, we conclude that AL benefits from large event logs to learn accurate models. The sigmoid approximation α_{approx} achieves highly accurate models on average. For PM, the state representation is crucial to avoid underapproximations, preferably exploiting domain knowledge. By combining AL-UJ and PM-UJ, HYBRID learns accurate models independent of the event log size.

6.3 Threats to Validity

We developed our own benchmark to simulate user journeys, as we assume that the existing benchmarks for AL and PM do not reflect the behavioral aspects of the user journeys. The random generation of ground-truth models still bears the risk of creating user journeys that might be unlikely in practice. Therefore, we evaluate AL and PM also on real-world case studies in Sect. 7. In addition, although the test method covers the transitions, it does not exhaustively test all behavior. We assume that transition coverage sufficiently tests critical behavior of user journeys. Other testing methods, e.g., W-Method [73, 74] or probably approximately correct (PAC) testing [75], would test many negative traces due to the restrictive sequential structure of user journeys. HYBRID uses a syntactic criterion to decide between PM and AL technique, where the variance in the event log is compared to its size. This criterion can be extended to include the semantics of the event log; e.g., Back et al. [76] propose entropies specifically for event logs.

The performance of HYBRID depends on the AL and PM methods that it uses. We chose Alergia for the AL part as it is a well-established algorithm with a maintained implementation in publicly available learning libraries such as AALPY [70]. Alergia naturally supports the use of the confidence parameter α to decide between AL and PM. However, in the literature on AL benchmarking [77], *Evidence Driven State Merging* (EDSM) [44] shows favorable performance over Alergia for classical grammatical inference problems.

Recently, a general state-merging approach based on EDSM was included in AALPY [70, 78]. A general *scoring* function to rank possible merges enables the algorithm to favour merges with the largest support in the underlying log. This implementation allows to combine evidence-drive state merging with other state-merging based algorithms, such as Alergia.

In further experiments (not reported in this paper), we tested EDSM as defined by Lang *et al.* [44] and EDSM in combination with Alergia, instantiated with default parameters, and it did not achieve a better performance than our reported results in Sect. 6.2. We envision that for better performance, EDSM would also require a parameter optimization, as we have done in AL-UJ. Additionally, the scoring function could be adapted to include domain knowledge to further adjust state merges in user journey models.

7 Experiments on Real-World Case Studies

We now evaluate the applicability of AL and PM techniques in practice. As no ground-truth models exist, we here compare behavioral similarities between the models learned by AL, PM, and HYBRID. Our results demonstrate that the quality of learned models depends on the sparsity of the event log, and that preprocessing may be crucial in practice. In summary, we address how AL and PM techniques scale to real-world user journeys by addressing the following research questions:

RQ2: *What is the impact of sparse event logs on learning with PM and AL?*
RQ3: *How do the models learned by PM and AL for real-world user journeys compare?*

7.1 Real-World Case Studies

For the practical evaluation of how the proposed AL and PM techniques scale to real-world case studies, we considered four different event logs. The event logs are constructed from datasets by grouping events into traces and ordering the events in a trace by their timestamps.

GrepS. Event log for user journeys from the company GrepS, for conducting programming skill evaluations with job applicants; this case study is taken from Kobialka et al. [13]. The user journey consists of (i) a sign-up phase, (ii) a task-solving phase where the users are presented with a range of real-world programming tasks, and (iii) a review phase, where users are presented with their evaluation and

are asked to approve their results. The provided log contains 33 traces.

BPIC12, BPIC17a & BPIC17b. BPIC, the Business Process Intelligence Challenge, is organized regularly by the IEEE Task Force on Process Mining.[2] We use the BPIC12 [32] and BPIC17 [35] event logs from a Dutch financial institution. The logs provide detailed user interactions, e.g. phone calls, between a bank (the service provider) and applicants (the user) in a loan application process. We preprocess the BPIC12 and BPIC17 event logs following setups from the literature [33, 79, 19]. BPIC17 records a change in the underlying behavior, a so-called *concept drift* [80]. Thus, we split BPIC17 into BPIC17a and BPIC17b containing traces before and after the concept drift, respectively. BPIC12 contains 5 053, BPIC17a 10 746 and BPIC17b 12 344 traces. With preprocessing, BPIC17a reduces to 5 515 and BPIC17b 6 989 traces; BPIC12 remains at the same size.

7.2 Experimental Design and Setup

To find adequate state representations for AL and PM for the real-world case studies, we exploit the insights gained in the benchmark evaluation presented in Sect. 6, and used the same parameter setup for HYBRID. Note that when applying AL and PM to logs, the techniques have different semantic definitions of states in their learned TSs: AL defines states by the distribution over the observed future states, while PM defines states over a state representation function including, e.g., a trace prefix and suffix.

In practice, model performance and later analysis benefit from small models with a feasible number of loops. For PM, we propose to generate models with the state representations in question and choose a possible small model with a feasible number of loops. This approach was approximated by minimizing the sum of loops and states in Sect. 6. For AL, low values for α promote state merging, whereas a higher α prevents merging. Since user journeys tend to follow sequential structures and loops should be reduced to a necessary minimum, we use a more restrained state merging for learning user journeys than the parameter setting proposed in the literature, as discussed in Sect. 6.

Learning Setup. For the state representation in PM, we chose state representations for trace prefix and suffix which are a good compromise between the number of states and the number of loops. For AL, we evaluate the impact of the confidence parameter α by considering two setups: (1) α set by α_{approx} in Eq. (3) using similar alphabet-size coefficients as in Sect. 6, and (2) low confidence ($\alpha = 0.9$). For HYBRID, we use the setup of PM-UJ and AL-UJ, in case PM or AL is selected, respectively. We set $\lambda = \lambda_{\text{approx}}$ similar as in Sect. 6. To automatically decide between AL and PM in HYBRID, we compare $\lambda_{\text{approx}} > \alpha_{\text{approx}}$ (see Sect. 5). Note that we limit the comparisons between certainty threshold λ_{approx} and confidence α_{approx} to two digits.

For each case study, we consider two different event logs, with and without preprocessing: (1) the *full event log* keeps single-occurrence traces and does not enumerate duplicate events. (2) the *preprocessed event log* removes all single-occurrence traces and enumerates iterative duplicated events, e.g., Offer 1, . . . , Offer 2, etc. In the sequel, we refer to different experimental setups by a string stating the learning technique, an indication if the log is preprocessed, and special algorithm parameters, e.g., AL-FULL-0.9 would refer to AL on the non-preprocessed event log with $\alpha = 0.9$, and AL-UJ to AL on the preprocessed event log with the computed $\alpha = \alpha_{\text{approx}}$ (cf. Eq. (3)). The experiments were run in the evaluation framework described in Sect. 6.1, using only machine (1).

We do not explicitly highlight in the following plots the individual results for HYBRID as it completely overlaps with either PM-UJ or AL-UJ. PM-UJ is always selected in the GrepS case study, and AL-UJ in all three BPIC case studies. This selection corresponds to our intention to use PM for sparse logs and AL for well-distributed logs.

RQ2. To investigate the impact of sparse event logs on learning, we split the event logs into two disjoint logs: a training log and a test log. The training log is used to learn the TS and the test log to evaluate the generality of the learned model. For this purpose, we consider training and test suites with different proportions of the full event logs going into the training log and into the test log. We divide the event logs with proportions ranging from 0.4 to 0.9 of the traces going into the training log, in 0.1 increments (thus, each increment corresponds to 10% of the traces in the log). Due to the randomness in the division of logs, ten training and test suites are created for each proportion–wise split. We then run the traces of the test log on the learned model. We say that a trace σ in the test suite *fails* for a TS if $\sigma \notin \mathcal{L}(TS)$, otherwise it is accepted. We consider event logs with and without preprocessing.

RQ3. The results reported in Sect. 6 and Tbl. 2 suggest that the compared learning techniques yield different TSs. To compare the models, we analyze the number of overlapping failed traces for the different TSs. For this, we run all failed traces from TS A on TS B and vice versa. However, these test-based experiments are biased towards the *generality* of models, where a general model accepts more traces from an event log. Therefore, we also investigate the differences in the defined languages by the learned TSs. To quantify the overlap between the accepted languages of the two models, we define one model to be the ground truth and test the *recall* of the other model by creating a transition coverage test suite as described in Sect. 6. If we do the same in both directions, we can see the overlap between the languages. We assume that TS A is more general than TS B if A accepts many traces generated by B, but B accepts fewer traces generated by A.

As an additional metric to evaluate the generality of a TS, we use the average frequency of traversing a transition by running all traces. The traversal frequency of a transition is the sum of appearances of the transition in the set of runs generated when running all traces in an event log.

7.3 Results

Table 2 shows the execution time and the number of states of the learned TSs for the different learning setups in all case studies. The computation times are measured over one execution as the algorithms are determinstic. We considered an event log (without preprocessing, indicated by "full") and a preprocessed event log. Note that all learned TSs

2. https://www.tf-pm.org/

TABLE 2: Execution time and the number of states of the learned TSs for the different AL and PM techniques.

	AL-0.9 full log		AL full log		PM full log		Events	AL-0.9 preprocessed		AL preprocessed		PM preprocessed		Events
	time (s)	# states	time (s)	# states	time (s)	# states		time (s)	# states	time (s)	# states	time (s)	# states	
GrepS	$4.2 \cdot 10^{-3}$	37	$1.1 \cdot 10^{-3}$	19	$1.4 \cdot 10^{-3}$	20	25	$3.9 \cdot 10^{-3}$	40	$1.2 \cdot 10^{-3}$	28	$1.1 \cdot 10^{-3}$	29	25
BPIC12	0.11	84	0.03	32	0.12	384	18	0.21	108	0.04	42	0.12	707	18
BPIC17a	1.79	156	0.11	39	1.09	337	23	0.06	68	0.03	43	0.46	143	26
BPIC17b	2.67	194	0.13	42	0.31	394	23	0.12	88	0.04	40	0.13	168	26

TABLE 3: States and loops of DFS $S^{\{t\}_r}_{[h]_c}$ for BPIC17a for row and column indices r and c; loops are given in parenthesis.

Prefix length	Suffix length 0	1	2	3
1	27 (103)	79 (46)	156 (5)	265 (0)
2	69 (88)	**143 (5)**	248 (0)	380 (0)
3	123 (16)	226 (0)	356 (0)	512 (0)
4	186 (0)	311 (0)	467 (0)	632 (0)

define a language that includes all traces in the event logs. The results show little execution-time differences, but Alergia's iterative state-merging had an impact on the BPIC17 case studies. Comparing the two AL setups, we observe that setting $\alpha = \alpha_{\text{approx}}$ (cf. Eq. (3)) yields TSs with fewer states than $\alpha = 0.9$. The models generated with PM-UJ are mostly larger in the number of states, which relates to the construction of DFSs.

To choose state representations for PM-UJ, we compare different horizons for trace projection and structures for state projection. Table 3 presents state and loop numbers for the different trace projections when using *set* structures for prefixes and list structures for suffixes on the BPIC17a event log. Table 3 shows that prefix representations of length 2 and suffix representations of length 1 result in a reasonable number of states and loops (cf. parenthesis), still maintaining sufficient generality. We repeated this process for all case studies, selecting individual state representations, leveraging domain knowledge from Bautista et al. [33], Rodrigues et al. [79], Kobialka et al. [19].

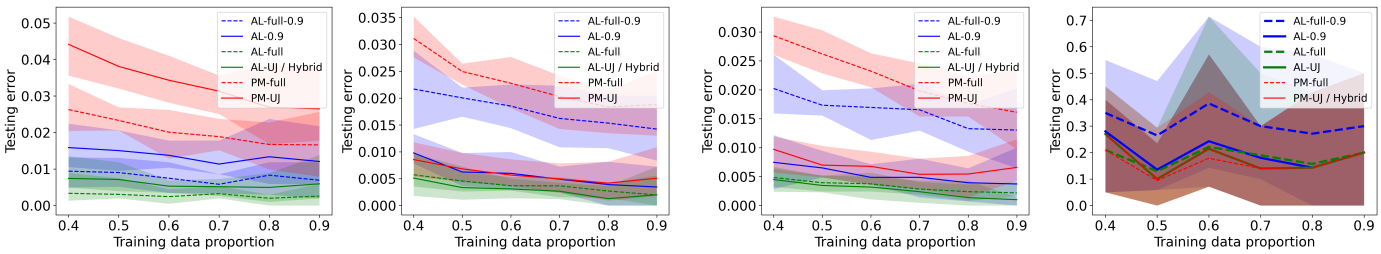
To evaluate the quality of the learned models under a lack of ground-truth models, we conduct additional experiments in **RQ2** and **RQ3**.

RQ2. To evaluate the impact of sparse logs, we conducted a growing sliding window experiment, splitting the available traces into a training and a test log. Figure 5 displays each iteration with the percentage of test traces that failed in the corresponding case study; the lines represent the average over the 10 models learned, the surrounding hue indicates maximum and minimum results. In general, all techniques failed fewer traces when provided with larger training logs. The models generated with AL and an $\alpha = 0.9$ serve as a baseline for AL-UJ and PM-UJ to evaluate *over-approximations*, as AL with $\alpha = 0.9$ should represent an *underapproximation*. Overall, we see that the testing error for AL is low, which might be an indicator for an over-approximation. The results also show that PM has similar performance as AL with $\alpha = 0.9$, which means that models learned with PM tend to represent an under-approximation. Furthermore, AL methods generate more general models on sparse logs than the PM method. For BPIC17, the models created by AL-0.9 and PM-UJ have comparable performance with similar generalizations on unseen test traces. Since the event log of GrepS is very small, training with only a subset bears a high risk that the learned model misses behavior.

The results reveal that both AL-0.9 and PM-UJ benefit from preprocessing. These results highlight that suitable preprocessing enables PM to successfully cope with sparsity. The HYBRID method correctly selects PM-UJ on smaller event logs, such as the log provided for GrepS. On larger event logs, such as the logs for the BPIC case studies, HYBRID selects AL-UJ, which achieves better generalizing approximations according to Fig. 5. To answer **RQ2**, our experiments show that sparse event logs—such as the log for GrepS—can be challenging for AL, while PM creates accurate models already on sparse logs, especially if the log is preprocessed. If the sparsity of the event logs is reduced—such as in the logs for BPIC—AL learns models with few failing tests, where PM might not generalize well enough on large event logs.

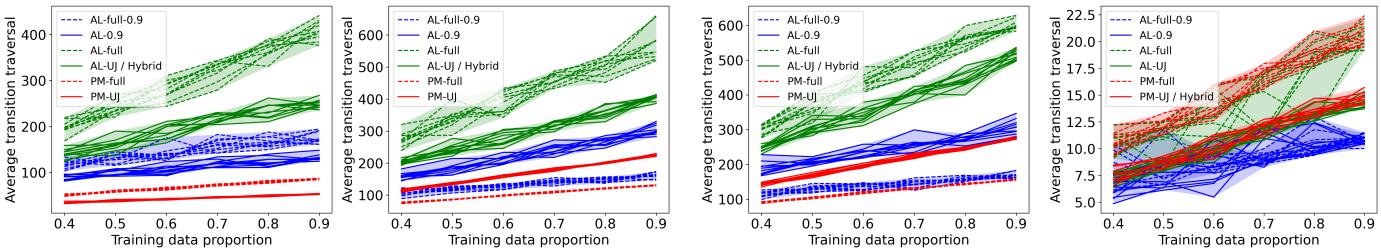
RQ3. The results reported in Sect. 6 and Tbl. 2 indicate that the evaluated techniques yield different TSs. To compare the differences between the models, we investigated the overlapping failing traces. Table 4 shows the average number of overlapping failing traces between models trained on the same log; e.g., for BPIC12-full, the average failing traces for AL-UJ is two, we see approximately the same number of failing traces for AL-0.9 and PM-UJ in the same column. For AL-0.9 and PM-UJ, we see that 8.4 and 16.9 traces fail on each technique, respectively. However, only 5.2 failing traces overlap for AL-0.9 and PM-UJ. In general, Tbl. 4 reports a similar trend between the full and preprocessed logs. The results show that approximately all failing traces of AL-UJ models also fail for AL-0.9 and PM-UJ models. Recall from Sect. 3 that a TS TS_A defines *more general behavior* than another TS TS_B if $\mathcal{L}(TS_B) \subseteq \mathcal{L}(TS_A)$ holds. This indicates that the AL-UJ models represent an overapproximation of the PM-UJ and AL-0.9 models. Further, the number of overlapping tests is around half of the total number, which suggests that there are behavioral differences between the PM-UJ and AL-0.9 models.

Table 5 shows the average recall values, with the row indicating the assumed ground-truth models; i.e., these models represent the basis for the assessment of the test verdict. The columns specify the tested model. We observe that the models generated by the same technique differ; e.g., on average the BPIC12 AL-FULL models accept a proportion of 0.76 of the traces generated by the ground-truth AL-FULL models. The results further support our assumption about overapproximation: AL-FULL accepts many traces generated by other models, but traces generated by the AL-FULL model are more likely to fail on other models. By preprocessing the log, AL tends to create traces that are more likely to be in the language of the other models. The AL-0.9 and PM-UJ models produced traces that were accepted by most other mod-



(a) BPIC12. (b) BPIC17a. (c) BPIC17b. (d) GrepS.

Fig. 5: Proportion of failed test traces for the case studies.



(a) BPIC12. (b) BPIC17a. (c) BPIC17b. (d) GrepS.

Fig. 6: Average transition frequencies for the case studies.

TABLE 4: Average overlapping failing traces with training log proportion 0.8.

	BPIC12-full			**BPIC17a-full**			**BPIC17b-full**			**GrepS-full**			**BPIC12**			**BPIC17a**			**BPIC17b**			**GrepS**		
	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ
AL-UJ	2.0	1.9	2.0	5.8	4.3	5.0	5.9	5.0	5.5	1.1	1.0	1.0	5.0	4.5	5.0	1.4	1.2	0.8	1.9	1.6	1.8	1.0	1.0	1.0
AL-0.9	1.9	8.4	5.2	4.3	33.0	12.6	5.0	32.8	16.1	1.0	1.9	1.0	4.5	13.5	10.8	1.2	4.3	2.0	1.6	5.5	3.5	1.0	1.0	1.0
PM-UJ	2.0	5.2	16.9	5.0	12.6	39.5	5.5	16.1	43.6	1.0	1.0	1.0	5.0	10.8	27.2	0.8	2.0	4.6	1.8	3.5	7.6	1.0	1.0	1.0

TABLE 5: Average recall over all generated model pairs with training log proportion 0.8.

test / **base**	**BPIC12-full**			**BPIC17a-full**			**BPIC17b-full**			**GrepS-full**			**BPIC12**			**BPIC17a**			**BPIC17b**			**GrepS**		
	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ	AL-UJ	AL-0.9	PM-UJ
AL-UJ	0.76	0.72	0.73	0.96	0.73	0.57	0.95	0.69	0.52	0.85	0.75	0.84	0.64	0.62	0.71	0.95	0.82	0.84	0.95	0.82	0.84	0.88	0.75	0.88
AL-0.9	0.90	0.87	0.81	0.96	0.88	0.74	0.96	0.87	0.72	0.87	0.83	0.88	0.75	0.75	0.74	0.92	0.93	0.85	0.98	0.95	0.91	0.89	0.84	0.89
PM-UJ	0.99	0.96	0.93	0.94	0.78	0.92	0.98	0.84	0.93	0.84	0.76	0.84	0.98	0.90	0.92	0.95	0.87	0.97	0.97	0.91	0.97	0.88	0.75	0.88

els. On average proportions of 0.86 and 0.90 of the traces are positive, respectively, which are the lowest recall values observed. We assume that AL-0.9 and PM-UJ approach a similar level of approximation. These results show that preprocessing can prevent AL from overapproximating the event log and could be used for sparse logs with a high variety.

The analysis of our second metric on transition traversal frequency indicates that AL-UJ models define more general languages. Figure 6 compares the average transition frequency when running the training log for each of the 10 trained models of each proportion-wise split with training log proportions between 0.4 and 0.9. The results show that the runs performed on the AL-UJ model traverse the same transitions more often. The average values are increasing. This indicates that that a larger training log did not necessarily lead to models with more states, but to more general models. The average transition frequency of the PM-UJ model grows at the same rate as AL-0.9. A similar observation applies to the full event logs. However, AL-0.9 seems to create more general models than PM-UJ. Further, the AL-FULL models have the highest fluctuations in the average traversals, while the PM-UJ models are the most constant. This highlights that the behavior defined by models learned by AL techniques highly depend on the underlying event log, and that PM techniques are more robust to minor fluctuations in the event log. In summary, we answer **RQ3** by observing that models learned by AL are more general and have fewer states, while models created with PM are less general and have more states.

7.4 Threats to Validity

To avoid a biased evaluation, we considered four case studies with four event logs ranging in size from 33 to several thousand traces. The case studies record digital services assuming that users do not perform actions simultaneously.

For **RQ2**, the results for GrepS seem to not align with the BPIC results. We believe this is due to the small event log provided for GrepS, where subsets of the event log are likely to not include all behavioral aspects. Furthermore, we note that the evaluation for **RQ2** favors models that define a more general behavior. Our experiments use a preprocessing adjusted for user journeys, originally introduced by Kobialka et al. [81]. Our results show that appropriate preprocessing has a strong influence on the PM and AL models. To answer **RQ3**, we used a similar conformance-testing technique based on transition coverage as in Sect. 6. We assume that transition coverage is sufficient to estimate the language overlap as exhaustive testing methods were not computationally feasible, see Sect. 6.3. Although complimentary in size, the real-world case studies in our experiments all exhibit a fairly protocol-like behavior, where users aim for a final goal. We believe this reflects the shape of most user journeys; user journeys in which a large set of actions could be chosen in near arbitrary and repetitive order lie outside of the conducted experiments.

8 Discussion on Actionable Insights

In software engineering research, we aim for tools that support and improve the development process. The success of service-oriented software solutions depends on the user experience, which needs to be considered in the software engineering process. This paper compared AL and PM, two techniques that provide insights into user behavior via transition systems. Our extensive evaluation shows that both techniques are capable of accomplishing this task. Still, the question remains which technique should be applied in practice to a given event log?

Actionable Insights. The HYBRID method combines practical insights from our experiments, and is recommended for learning behavioral models of user journeys in practice: HYBRID automatically selects PM or AL considering that (1) PM requires domain knowledge and is well-suited for small event logs, and (2) AL requires an adapted confidence parameter estimation and larger logs. HYBRID should be applied with a preprocessing of the event log, independent of the learning technology it selected. Further, HYBRID can be instantiated with any PM and AL algorithm, leveraging hand-curated evaluation functions for the given log. The following discussion provides further explanations for these findings.

For real-world applications, there exists no ground-truth model. To evaluate the ability of AL and PM to adequately capture the ground truth (**RQ1**, cf. Sect. 6), we created a synthetic benchmark suite. The results in Sect. 6 suggest that PM techniques are beneficial for sparse logs when expert domain knowledge is available, i.e., knowledge about the interdependence of past and future events. Our evaluation investigated different state representations for PM. Without a suitable state representation, models generated by PM methods risk to not define the behavior adequately. Therefore, we suggest to apply PM techniques for the early stages of software engineering, when only few user interactions are recorded. AL can generate user-journey models even from complex event logs, by taking into account the underlying distribution of events in the log in the form of confidence parameter α. Our results showed that the approximation of α proposed by Mao et al. [66] allows too many states to be merged, creating overapproximating models. We proposed a confidence-parameter approximation α_{approx} that supports the creation of models that are neither over- nor underapproximations. As AL benefits from large, well-distributed event logs, AL is better suited for later stages in the software engineering process.

The results from four real-world case studies (Sect. 7) are consistent with the findings from the synthetic benchmark (Sect. 6). In **RQ2** (cf. Sect. 7), we investigated the learning algorithms for different levels of sparsity in the underlying event log. Our results show that AL struggles for very sparse event logs, but creates accurate models on larger event logs. We observed that PM could handle sparse event logs, but learns underapproximations for large event logs. In **RQ3** (cf. Sect. 7), we compared the behavioral differences between the learned models, reinforcing the previous findings. For PM, we used domain knowledge to learn models established for the case studies. Still, the resulting models generalized less than models learned by other techniques.

For AL, the learned models overapproximated the behavior. Using the proposed approximation α_{approx}, prevented AL from creating overapproximations. Note that α_{approx} is optimized for the goal-oriented behavior observed in user journeys, with limited repetitions of events in a trace.

Finally, user journeys have particular characteristics, which PM techniques take into account by preprocessing the event log. Our evaluation showed that these preprocessing techniques are also beneficial for AL, as the learned models from preprocessed event logs achieve often higher accuracy.

Further Analysis Steps and Managerial Implications. We briefly consider analysis steps once a user journey model has been learned from the event logs. Postprocessing of learned models is common in PM to utilize manual model analyses: For example, observed patterns are completed by introducing missing transitions, transitions between states are merged, and states are merged into groups representing, e.g., the different phases of the process [23]. Furthermore, transition systems can be transformed into other process models, such as Petri nets [23]. Leemans [82] compares techniques for generating process models from very large logs and provides practical recommendations. Furthermore, Aichernig et al. [83] show that AL-generated models can provide a baseline for model-based analysis and verification such as testing or model-checking. Hence, it is important to know whether the learned model represents an over- or under-approximation. Finally, AL models can be refined through counterexamples showing behavioral differences between the learned model and the system under learning, adapting the idea of Walkinshaw et al. [84].

These analyses can be exploited from a managerial perspective to assess service changes and usage trends. As HYBRID is an automated technique, realistic models of the current user journeys can be constructed in a timely fashion. Leveraging automated analysis techniques [12], the models constructed by HYBRID can support managerial decision-making processes when further developing a service.

Log Quality. Throughout this work, we implicitly assumed that the used logs were of sufficient quality. However, event log quality is essential for applying process

mining techniques successfully but in practise, most event logs are "incomplete, noisy, and imprecise" [85]. Quality aspects of event logs are actively researched in process mining, e.g., tthe one to five star rating proposed by the "Process Mining Manifesto" [86]. Preprocessing (cf. Sect. 3) can enhance later analysis results by improving the quality of the log [87]. Marin-Castro and Tello-Leal [87] review preprocessing techniques and highlight that there is not one preferable preprocessing technique, rather, multiple techniques are needed for addressing several quality aspects of the given event log. Recently, the representativeness of event logs for the underlying process has been studied [88, 89, 90] by formalizing the notion of completeness for event logs. All these works complement our proposed method.

9 Conclusion

This paper presents a comprehensive evaluation of passive AL and PM for learning user journey models, with the following main contributions: (1) A novel benchmark suite for generating ground-truth models of user journeys with corresponding event logs. (2) A comprehensive evaluation of AL and PM techniques, using the benchmark set. The results show that both techniques can generate user journey models, but both have limitations that depend on the given event log. (3) A novel method, Hybrid, to automatically select between available AL and PM algorithms depending on the properties of a given event log. Our evaluation shows that Hybrid can support software engineering processes with a very accurate model independent of the event log. (4) An evaluation of the practical feasibility of applying these methods to real-world user journeys. Evaluating the investigated learning techniques based on their performance on sparse event logs, we learn that sparse logs are not suited for AL, while PM handles them well. Larger event logs improve AL results, while PM struggles to create models that generalize well. Comparing the behavioral differences between the learned models, we observe that AL learns models that overapproximate the behavior; i.e., the models describe behavior that is not observable in practice. However, PM creates underapproximations. We further discuss actionable insights for applying AL or PM to learn real-world user journeys. For AL, we see a strong dependence on the distribution in the event log. Therefore, the parameters of the learning algorithm must be carefully selected to learn accurate models. For PM, domain knowledge about the underlying service is required to find an adequate state representation of the learned model. Both methods may benefit from preprocessing the event log, ideally exploiting domain knowledge.

Future Work. For future work, we plan to investigate the use of different model-based verification techniques, e.g. model-checking, to analyze user behavior based on the learned models. Such automated analyses may create valuable insights into the service design and user experience for improving services. Furthermore, it is interesting to investigate how models learned with AL can be integrated with established process analysis steps used in PM. This might also include the consideration of other case studies from PM beyond user journeys such as software product lines, e.g., as presented by Damasceno *et al.* [50].

10 Data Availability

An artifact to replicate the presented results is available at https://figshare.com/s/3acae9725dda09cce811.

References

[1] S. Vandermerwe and J. Rada, "Servitization of business: Adding value by adding services," *Eur. Management Journal*, vol. 6, no. 4, pp. 314–324, Dec. 1988. [Online]. Available: https://doi.org/10.1016/0263-2373(88)90033-3

[2] C. Fornell, S. Mithas, F. V. Morgeson, and M. S. Krishnan, "Customer satisfaction and stock prices: High returns, low risk," *Journal of Marketing*, vol. 70, no. 1, pp. 3–14, Jan. 2006. [Online]. Available: https://doi.org/10.1509/jmkg.70.1.003.qxd

[3] M. S. Rosenbaum, M. L. Otalora, and G. C. Ramírez, "How to create a realistic customer journey map," *Business Horizons*, vol. 60, no. 1, pp. 143–150, 2017. [Online]. Available: https://doi.org/10.1016/j.bushor.2016.09.010

[4] R. Halvorsrud, K. Kvale, and A. Følstad, "Improving service quality through customer journey analysis," *Journal of Service Theory and Practice*, vol. 26, no. 6, pp. 840–867, Nov. 2016. [Online]. Available: https://doi.org/10.1108/JSTP-05-2015-0111

[5] R. Halvorsrud, F. Mannhardt, E. B. Johnsen, and S. L. Tapia Tarifa, "Smart journey mining for improved service quality," in *IEEE Intl. Conf. on Services Computing (SCC 2021)*. IEEE, 2021, pp. 367–369. [Online]. Available: https://doi.org/10.1109/SCC53864.2021.00051

[6] F. W. Vaandrager, "Model learning," *Communications of the ACM*, vol. 60, no. 2, pp. 86–95, 2017. [Online]. Available: https://doi.org/10.1145/2967606

[7] W. M. P. van der Aalst, *Process Mining - Data Science in Action*, 2nd ed. Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-662-49851-4

[8] C. Pahl, "Adaptive development and maintenance of user-centric software systems," *Inf. Softw. Technol.*, vol. 46, no. 14, pp. 973–986, 2004. [Online]. Available: https://doi.org/10.1016/j.infsof.2004.04.004

[9] R. T. Rust and M.-H. Huang, "The service revolution and the transformation of marketing science," *Marketing Science*, vol. 33, no. 2, pp. 206–221, 2014. [Online]. Available: https://doi.org/10.1287/mksc.2013.0836

[10] K. N. Lemon and P. C. Verhoef, "Understanding customer experience throughout the customer journey," *Journal of marketing*, vol. 80, no. 6, pp. 69–96, 2016. [Online]. Available: https://doi.org/10.1509/jm.15.0420

[11] A. Følstad and K. Kvale, "Customer journeys: a systematic literature review," *Journal of service theory and practice*, vol. 28, no. 2, pp. 196–227, 2018. [Online]. Available: https://doi.org/10.1108/JSTP-11-2014-0261

[12] P. Kobialka, A. Pferscher, G. R. Bergersen, E. B. Johnsen, and S. L. Tapia Tarifa, "Stochastic games for user journeys," in *26th Intl. Symposium on Formal Methods (FM 2024)*, ser. Lecture Notes in Computer Science, vol. 14934. Springer, 2025, pp. 167–186. [Online]. Available: https://doi.org/10.1007/978-3-031-71177-0_12

[13] P. Kobialka, S. L. Tapia Tarifa, G. R. Bergersen, and E. B. Johnsen, "User journey games: Automating user-centric analysis," *Software & System Modeling*, 2024. [Online]. Available: https://doi.org/10.1007/s10270-024-01148-2

[14] E. M. Clarke, "Model checking," in *Foundations of Software Technology and Theoretical Computer Science*. Springer, 1997, pp. 54–56.

[15] C. Baier, L. de Alfaro, V. Forejt, and M. Kwiatkowska, "Model checking probabilistic systems," in *Handbook of Model Checking*. Springer, 2018, pp. 963–999. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8_28

[16] P. Riehmann, M. Hanfler, and B. Froehlich, "Interactive Sankey diagrams," in *Symposium on Information Visualization (InfoVis 2005)*. IEEE Computer Society, 2005, pp. 233–240. [Online]. Available: https://doi.org/10.1109/INFVIS.2005.1532152

[17] W. M. P. van der Aalst, "Foundations of process discovery," in *Process Mining Handbook*. Springer, 2022, pp. 37–75. [Online]. Available: https://doi.org/10.1007/978-3-031-08848-3_2

[18] P. Kobialka, S. L. Tapia Tarifa, G. R. Bergersen, and E. B. Johnsen, "Weighted games for user journeys," in *20th Intl. Conf. Software Engineering and Formal Methods (SEFM 2022)*, ser. Lecture Notes in Computer Science, vol. 13550. Springer, 2022, pp. 253–270. [Online]. Available: https://doi.org/10.1007/978-3-031-17108-6_16

[19] P. Kobialka, E. B. Johnsen, F. Mannhardt, and S. L. Tapia Tarifa, "Decision boundaries in user journey event logs," *Process Science*, vol. 2, no. 1, p. 27, 2025. [Online]. Available: https://doi.org/10.1007/s44311-025-00026-4

[20] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. New York, NY, USA: Cambridge University Press, 2010. [Online]. Available: https://doi.org/10.1017/CBO9781139194655

[21] R. C. Carrasco and J. Oncina, "Learning stochastic regular grammars by means of a state merging method," in *2nd Intl. Colloquium on Grammatical Inference and Applications (ICGI-94)*, ser. Lecture Notes in Computer Science, vol. 862. Springer, 1994, pp. 139–152. [Online]. Available: https://doi.org/10.1007/3-540-58473-0_144

[22] S. Verwer and C. A. Hammerschmidt, "FlexFringe: Modeling software behavior by learning probabilistic automata," *CoRR*, vol. abs/2203.16331, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.16331

[23] W. M. P. van der Aalst, V. A. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software & System Modeling*, vol. 9, no. 1, pp. 87–111, 2010. [Online]. Available: https://doi.org/10.1007/s10270-008-0106-z

[24] R. Halvorsrud, O. R. Sanchez, C. Boletsis, and M. Skjuve, "Involving users in the development of a modeling language for customer journeys," *Software & System Modeling*, vol. 22, no. 5, pp. 1589–1618, 2023. [Online]. Available: https://doi.org/10.1007/s10270-023-01081-w

[25] G. Bernard and P. Andritsos, "A process mining based model for customer journey mapping," in *Forum and Doctoral Consortium Papers at the 29th Intl. Conf. on Advanced Information Systems Engineering (CAiSE 2017)*, ser. CEUR Workshop Proceedings, vol. 1848. CEUR-WS.org, 2017, pp. 49–56. [Online]. Available: https://ceur-ws.org/Vol-1848/CAiSE2017_Forum_Paper7.pdf

[26] ——, "CJM-ex: Goal-oriented exploration of customer journey maps using event logs and data analytics," in *BPM 2017 Demo Track and BPM 2017 Dissertation Award*, ser. CEUR Workshop Proceedings, vol. 1920. CEUR-WS.org, 2017. [Online]. Available: https://ceur-ws.org/Vol-1920/BPM_2017_paper_172.pdf

[27] ——, "CJM-ab: Abstracting customer journey maps using process mining," in *Information Systems in the Big Data Era (CAiSE Forum 2018)*, ser. Lecture Notes in Business Information Processing, vol. 317. Springer, 2018, pp. 49–56. [Online]. Available: https://doi.org/10.1007/978-3-319-92901-9_5

[28] ——, "Contextual and behavioral customer journey discovery using a genetic approach," in *23rd Eur. Conf. on Advances in Databases and Information Systems (ADBIS 2019)*, ser. Lecture Notes in Computer Science, vol. 11695. Springer, 2019, pp. 251–266. [Online]. Available: https://doi.org/10.1007/978-3-030-28730-6_16

[29] M. Harbich, G. Bernard, P. Berkes, B. Garbinato, and P. Andritsos, "Discovering customer journey maps using a mixture of Markov models," in *7th Intl. Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2017)*, ser. CEUR Workshop Proceedings, vol. 2016. CEUR-WS.org, 2017, pp. 3–7. [Online]. Available: http://ceur-ws.org/Vol-2016/paper1.pdf

[30] A. Terragni and M. Hassani, "Analyzing customer journey with process mining: From discovery to recommendations," in *6th Intl. Conf. on Future Internet of Things and Cloud (FiCloud 2018)*. IEEE Computer Society, 2018, pp. 224–229. [Online]. Available: https://doi.org/10.1109/FiCloud.2018.00040

[31] ——, "Optimizing customer journey using process mining and sequence-aware recommendation," in *34th Symposium on Applied Computing (SAC 2019)*. ACM, 2019, pp. 57–65. [Online]. Available: https://doi.org/10.1145/3297280.3297288

[32] B. F. van Dongen, "BPI challenge 2012," Dataset, 4TU.ResearchData.nl, 2012. [Online]. Available: https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204

[33] A. D. Bautista, L. Wangikar, and S. M. K. Akbar, "Process mining-driven optimization of a consumer loan approvals process," in *BPM 2012 Workshops*, ser. Lecture Notes in Business Information Processing, vol. 132. Springer, 2012, pp. 219–220. [Online]. Available: https://doi.org/10.1007/978-3-642-36285-9_24

[34] H. M. W. E. Verbeek, "BPI challenge 2012: The transition system case," in *Business Process Management Workshops (BPM 2012)*, ser. Lecture Notes in Business Information Processing, vol. 132. Springer, 2012, pp. 225–226. [Online]. Available: https://doi.org/10.1007/978-3-642-36285-9_27

[35] B. F. van Dongen, "BPI challenge 2017,"

Dataset, 4TU.ResearchData.nl, 2017. [Online]. Available: https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884

[36] C. D. Ciccio and M. Mecella, "On the discovery of declarative control flows for artful processes," *ACM Trans. Manag. Inf. Syst.*, vol. 5, no. 4, pp. 24:1–24:37, 2015. [Online]. Available: https://doi.org/10.1145/2629447

[37] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst, "Efficient discovery of understandable declarative process models from event logs," in *24th Intl. Conf. on Advanced Information Systems Engineering (CAiSE 2012)*, ser. Lecture Notes in Computer Science, vol. 7328. Springer, 2012, pp. 270–285. [Online]. Available: https://doi.org/10.1007/978-3-642-31095-9_18

[38] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," *ACM Trans. Softw. Eng. Methodol.*, vol. 7, no. 3, pp. 215–249, 1998. [Online]. Available: https://doi.org/10.1145/287000.287001

[39] J. Esparza, M. Leucker, and M. Schlund, "Learning workflow Petri nets," *Fundam. Informaticae*, vol. 113, no. 3-4, pp. 205–228, 2011. [Online]. Available: https://doi.org/10.3233/FI-2011-607

[40] S. Agostinelli, F. Chiariello, F. M. Maggi, A. Marrella, and F. Patrizi, "Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis," *Information Systems*, vol. 114, p. 102180, 2023. [Online]. Available: https://doi.org/10.1016/j.is.2023.102180

[41] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987. [Online]. Available: https://doi.org/10.1016/0890-5401(87)90052-6

[42] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, "DECLARE: full support for loosely-structured processes," in *Intl. Enterprise Distributed Object Computing Conf. (EDOC 2007)*. IEEE Computer Society, 2007, pp. 287–300. [Online]. Available: https://doi.org/10.1109/EDOC.2007.14

[43] P. W. Adriaans and C. J. H. Jacobs, "Using MDL for grammar induction," in *8th Intl. Colloquiumon Grammatical Inference: Algorithms and Applications (ICGI 2006)*, ser. Lecture Notes in Computer Science, vol. 4201. Springer, 2006, pp. 293–306. [Online]. Available: https://doi.org/10.1007/11872436_24

[44] K. J. Lang, B. A. Pearlmutter, and R. A. Price, "Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm," in *4th Intl. Colloquium on Grammatical Inference (ICGI-98)*, ser. Lecture Notes in Computer Science, vol. 1433. Springer, 1998, pp. 1–12. [Online]. Available: https://doi.org/10.1007/BFb0054059

[45] E. B. Johnsen, P. Kobialka, A. Pferscher, and S. L. Tapia Tarifa, "Nudging strategies for user journeys: Take a path on the wild side," in *Real Time and Such - Essays Dedicated to Wang Yi to Celebrate His Scientific Career*, ser. Lecture Notes in Computer Science, vol. 15230. Springer, 2025, pp. 42–63. [Online]. Available: https://doi.org/10.1007/978-3-031-73751-0_6

[46] R. Wieman, M. F. Aniche, W. Lobbezoo, S. Verwer, and A. van Deursen, "An experience report on applying passive learning in a large-scale payment company," in *Intl. Conf. on Software Maintenance and Evolution (ICSME 2017)*. IEEE Computer Society, 2017, pp. 564–573. [Online]. Available: https://doi.org/10.1109/ICSME.2017.71

[47] F. R. Blum, J. Simmonds, and M. C. Bastarrica, "Software process line discovery," in *Intl. Conf. on Software and System Process*, 2015, pp. 127–136. [Online]. Available: https://doi.org/10.1145/2785592.2785605

[48] V. A. Rubin, C. W. Günther, W. M. P. van der Aalst, E. Kindler, B. F. van Dongen, and W. Schäfer, "Process mining framework for software processes," in *Intl. Conf. on Software Process (ICSP 2007)*, ser. Lecture Notes in Computer Science, vol. 4470. Springer, 2007, pp. 169–181. [Online]. Available: https://doi.org/10.1007/978-3-540-72426-1_15

[49] H. Chemingui, I. Gam, R. Mazo, C. Salinesi, and H. B. Ghézala, "Product line configuration meets process mining," in *Intl. Conf. on ENTERprise Information Systems (CENTERIS 2019) / Intl. Conf. on Project MANagement (ProjMAN 2019) / Intl. Conf. on Health and Social Care Information Systems and Technologies (HCist 2019)*, ser. Procedia Computer Science, vol. 164. Elsevier, 2019, pp. 199–210. [Online]. Available: https://doi.org/10.1016/j.procs.2019.12.173

[50] C. D. N. Damasceno, M. R. Mousavi, and A. da Silva Simão, "Learning from difference: an automated approach for learning family models from software product lines," in *23rd Intl. Systems and Software Product Line Conf. (SPLC 2019)*. ACM, 2019, pp. 10:1–10:12. [Online]. Available: https://doi.org/10.1145/3336294.3336307

[51] M. Nielsen, G. Rozenberg, and P. S. Thiagarajan, "Elementary transition systems and refinement," *Acta Informatica*, vol. 29, no. 6/7, pp. 555–578, 1992. [Online]. Available: https://doi.org/10.1007/BF01185561

[52] G. D. Plotkin, "A structural approach to operational semantics," *J. Log. Algebraic Methods Program.*, vol. 60-61, pp. 17–139, 2004.

[53] P. Kobialka, R. Schlatte, G. R. Bergersen, E. B. Johnsen, and S. L. Tapia Tarifa, "Simulating user journeys with active objects," in *Active Object Languages: Current Research Trends*, ser. Lecture Notes in Computer Science. Springer, 2024, vol. 14360, pp. 199–225. [Online]. Available: https://doi.org/10.1007/978-3-031-51060-1_8

[54] W. M. P. van der Aalst, "A practitioner's guide to process mining: Limitations of the directly-follows graph," in *Intl. Conf. on ENTERprise Information Systems (CENTERIS 2019) / Intl. Conf. on Project MANagement (ProjMAN 2019) / Intl. Conf. on Health and Social Care Information Systems and Technologies (HCist 2019)*, ser. Procedia Computer Science, vol. 164. Elsevier, 2019, pp. 321–328. [Online]. Available: https://doi.org/10.1016/j.procs.2019.12.189

[55] A. K. A. de Medeiros, A. J. Weijters, and W. M. van der Aalst, "Genetic process mining: an experimental evaluation," *Data mining and knowledge discovery*, vol. 14, no. 2, pp. 245–304, 2007. [Online]. Available: https://doi.org/10.1007/s10618-006-0061-7

[56] S. J. Leemans, D. Fahland, and W. M. Van Der Aalst, "Discovering block-structured process models from event logs-a constructive approach," in *Intl. conf. on applications and theory of Petri nets and concurrency*. Springer, 2013, pp. 311–329. [Online]. Available: https://doi.org/10.1007/978-3-642-38697-8_17

[57] W. Van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004. [Online]. Available: https://doi.org/10.1109/TKDE.2004.47

[58] A. J. Weijters, W. M. van Der Aalst, and A. A. De Medeiros, "Process mining with the Heuristics-Miner algorithm," Technische Universiteit Eindhoven, BETA publicatie : working papers 166, 2006.

[59] S. J. Leemans, E. Poppe, and M. Wynn, "Directly follows-based process mining: A tool," in *ICPM demo track 2019*, ser. CEUR Workshop Proceedings, vol. 2374. CEUR-WS.org, 2019, pp. 9–12. [Online]. Available: https://ceur-ws.org/Vol-2374/paper3.pdf

[60] E. M. Gold, "Complexity of automaton identification from given data," *Inf. Control.*, vol. 37, no. 3, pp. 302–320, 1978. [Online]. Available: https://doi.org/10.1016/S0019-9958(78)90562-4

[61] D. Angluin, "Identifying languages from stochastic example," Yale University, Tech. Rep. 614, Mar. 1988. [Online]. Available: http://www.cs.yale.edu/publications/techreports/tr614.pdf

[62] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426. [Online]. Available: https://doi.org/10.1007/978-1-4612-0865-5_26

[63] J. R. Norris, *Markov chains*, ser. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, 1998.

[64] E. M. Gold, "Language identification in the limit," *Inf. Control.*, vol. 10, no. 5, pp. 447–474, 1967. [Online]. Available: https://doi.org/10.1016/S0019-9958(67)91165-5

[65] M. Tomita, "Dynamic construction of finite automata from examples using hill-climbing," in *Conf. of the Cognitive Science Society*, 1982, pp. 105–108.

[66] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen, "Learning deterministic probabilistic automata from a model checking perspective," *Mach. Learn.*, vol. 105, no. 2, pp. 255–299, 2016. [Online]. Available: https://doi.org/10.1007/s10994-016-5565-9

[67] A. Berti, S. van Zelst, and D. Schuster, "PM4Py: A process mining library for Python," *Software Impacts*, vol. 17, p. 100556, 2023. [Online]. Available: https://doi.org/10.1016/j.simpa.2023.100556

[68] T. Jouck and B. Depaire, "PTandLogGenerator: A generator for artificial event data," in *BPM 2016 Demo Track*, ser. CEUR Workshop Proceedings, vol. 1789. CEUR-WS.org, 2016, pp. 23–27. [Online]. Available: https://ceur-ws.org/Vol-1789/bpm-demo-2016-paper5.pdf

[69] ——, "Generating artificial data for empirical analysis of control-flow discovery algorithms: A process tree and log generator," *Business & Information Systems Engineering*, vol. 61, pp. 695–712, 2019. [Online]. Available: https://doi.org/10.1007/s12599-018-0541-5

[70] E. Muškardin, B. K. Aichernig, I. Pill, A. Pferscher, and M. Tappler, "AALpy: an active automata learning library," *Innov. Syst. Softw. Eng.*, vol. 18, no. 3, pp. 417–426, 2022. [Online]. Available: https://doi.org/10.1007/s11334-022-00449-3

[71] N. Walkinshaw and K. Bogdanov, "Automated comparison of state-based software models in terms of their language and structure," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 2, pp. 13:1–13:37, 2013. [Online]. Available: https://doi.org/10.1145/2430545.2430549

[72] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, 2009. [Online]. Available: https://doi.org/10.1016/j.ipm.2009.03.002

[73] M. P. Vasilevskii, "Failure diagnosis of automata," *Cybernetics*, vol. 9, no. 4, pp. 653–665, 1973. [Online]. Available: https://doi.org/10.1007/BF01068590

[74] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. Software Engineering*, vol. 4, no. 3, pp. 178–187, 1978. [Online]. Available: https://doi.org/10.1109/TSE.1978.231496

[75] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, 1984. [Online]. Available: https://doi.org/10.1145/1968.1972

[76] C. O. Back, S. Debois, and T. Slaats, "Entropy as a measure of log variability," *Journal on Data Semantics*, vol. 8, pp. 129–156, 2019. [Online]. Available: https://doi.org/10.1007/s13740-019-00105-3

[77] A. Soubki and J. Heinz, "Benchmarking state-merging algorithms for learning regular languages," in *Intl. Conf. on Grammatical Inference (ICGI 2023)*, ser. Machine Learning Research, vol. 217. PMLR, 2023, pp. 181–198. [Online]. Available: https://proceedings.mlr.press/v217/soubki23a.html

[78] B. von Berg and B. K. Aichernig, "Extending AALpy with passive learning: A generalized state-merging approach," in *Intl. Conf. on Computer Aided Verification*. Springer, 2025, pp. 127–140. [Online]. Available: https://doi.org/10.1007/978-3-031-98685-7_6

[79] A. M. B. Rodrigues, C. F. P. Almeida, F. B. Saraiva, Daniel D. G. Moreira, G. M. Spyrides, G. Varela, I. T. Krieger, Gustavo M. Peres, L. F. Dantas, M. Lana, O. E. Alves, R. Franca, R. A. Q. Neira, S. F. Gonzalez, W. P. D. Fernandes, S. D. J. Barbosa, M. Poggi, and H. Lopes, "Stairway to value: mining a loan application process," in *13th Intl. Workshop on Business Process Intelligence*, 2017. [Online]. Available: https://www.win.tue.nl/bpi/lib/exe/fetch.php?media=2017:bpi2017_winner_academic.pdf

[80] J. N. Adams, S. J. van Zelst, L. Quack, K. Hausmann, W. M. P. van der Aalst, and T. Rose, "A framework for explainable concept drift detection in process mining," in *19th Intl. Conf. on Business Process Management (BPM 2021)*, ser. Lecture Notes in Computer Science, vol. 12875. Springer, 2021, pp. 400–416. [Online]. Available: https://doi.org/10.1007/978-3-030-85469-0_25

[81] P. Kobialka, F. Mannhardt, S. L. Tapia Tarifa, and E. B. Johnsen, "Building user journey games from

multi-party event logs," in *3rd Intl. Workshop on Event Data and Behavioral Analytics (EdbA 2022)*, ser. Lecture Notes in Business Information Processing, vol. 468. Springer, 2022. [Online]. Available: https://doi.org/10.1007/978-3-031-27815-0_6

[82] S. J. Leemans, "Robust process mining with guarantees." Ph.D. dissertation, Technische Universiteit Eindhoven, 2017.

[83] B. K. Aichernig, W. Mostowski, M. R. Mousavi, M. Tappler, and M. Taromirad, "Model learning and model-based testing," in *Machine Learning for Dynamic Software Analysis: Potentials and Limits*, ser. Lecture Notes in Computer Science, vol. 11026. Springer, 2018, pp. 74–100. [Online]. Available: https://doi.org/10.1007/978-3-319-96562-8_3

[84] N. Walkinshaw, J. Derrick, and Q. Guo, "Iterative refinement of reverse-engineered models by model-based testing," in *Second World Congress on Formal Methods (FM 2009)*, ser. Lecture Notes in Computer Science, vol. 5850. Springer, 2009, pp. 305–320. [Online]. Available: https://doi.org/10.1007/978-3-642-05089-3_20

[85] R. J. C. Bose, R. S. Mans, and W. M. Van Der Aalst, "Wanna improve process mining results?" in *Symposium on computational intelligence and data mining (CIDM 2013))*. IEEE, 2013, pp. 127–134. [Online]. Available: https://doi.org/10.1109/CIDM.2013.6597227

[86] W. Van Der Aalst, A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs *et al.*, "Process mining manifesto," in *Intl. conf. on business process management*. Springer, 2011, pp. 169–194. [Online]. Available: https://doi.org/10.1007/978-3-642-28108-2_19

[87] H. M. Marin-Castro and E. Tello-Leal, "Event log preprocessing for process mining: a review," *Applied Sciences*, vol. 11, no. 22, p. 10556, 2021. [Online]. Available: https://doi.org/10.3390/app112210556

[88] J. Pei, L. Wen, H. Yang, J. Wang, and X. Ye, "Estimating global completeness of event logs: A comparative study," *IEEE Transactions on Services Computing*, vol. 14, no. 2, pp. 441–457, 2018. [Online]. Available: https://doi.org/10.1109/TSC.2018.2805912

[89] M. Kabierski, M. Richter, and M. Weidlich, "Addressing the log representativeness problem using species discovery," in *5th Intl. Conf. on Process Mining (ICPM 2023)*. IEEE, 2023, pp. 65–72. [Online]. Available: https://doi.org/10.1109/ICPM60904.2023.10272004

[90] A. Karunaratne, A. Polyvyanyy, and A. Moffat, "The role of log representativeness in estimating generalization in process mining," in *6th Intl. Conf. on Process Mining (ICPM 2024)*. IEEE, 2024, pp. 33–40. [Online]. Available: https://doi.org/10.1109/ICPM63005.2024.10680679

Paul Kobialka is a postdoctoral researcher in the Reliable Systems group at the Department of Informatics at the University of Oslo, where he received his Ph.D. in 2025. Previously, he finished his Master degree in Computer Science at RWTH Aachen University in 2021. He is researching data-driven formal methods, i.e. problems where formal guarantees are necessary but formal models are too complex to construct traditionally. In his current work, he applies formal methods to improve user experiences in digital services (email: paulkob@ifi.uio.no).

Andrea Pferscher is a postdoctoral researcher at the Department of Informatics at the University of Oslo. She received her PhD from Graz University of Technology in 2023. In her PhD thesis, she investigated automata learning techniques for analyzing and testing the safety-critical behavior of network components. In her current position, she has broadened her focus to the application of formal methods for digital twins on health and environmental systems. She has been a member of different Austrian, European and Norwegian research projects. (email: andreapf@ifi.uio.no).

Bernhard K. Aichernig is a full professor of Formal Methods at Johannes Kepler University Linz (JKU), Austria, where he leads the Institute of Formal Models and Verification (FMV). His research focuses on the foundations of software engineering for dependable and trustworthy systems, with interests in automated falsification, verification, and modelling. Current topics include automata learning, learning-based testing, and the integration of symbolic and subsymbolic AI. He is the author of more than 140 scientific publications. Until April 2025, he was affiliated with Graz University of Technology. From 2002 to 2006, he held a faculty position at the United Nations University in Macao, China. He served on the board of Formal Methods Europe from 2004 to 2016. Prof. Aichernig holds a habilitation in Practical Computer Science and Formal Methods, a doctorate, and a Diplom-Ingenieur degree, all from Graz University of Technology. (email: bernhard.aichernig@jku.at).

Einar Broch Johnsen is a professor at the Department of Informatics, University of Oslo, where he leads the Reliable Systems research group. He is active in formal methods for distributed and concurrent systems, including object-oriented and actor languages, manycore computing, cloud computing and digital twins. He is one of the main developers of the ABS modeling language. He has been prominently involved in many national and European research projects; in particular, he was the strategy director of Sirius, a center for research-driven innovation on scalable data access, he was the coordinator of the EU FP7 project Envisage (2013–2016) on formal methods for cloud computing and the scientific coordinator of the EU H2020 project HyVar (2015–2018) on hybrid variability systems. Einar Broch Johnsen is member of IFIP WG2.2 "Formal Description of Programming Concepts". He was board member of Sintef ICT (2009–2015). He is currently member of the Scientific Council of the dScience centre at University of Oslo, board member of Formal Methods Europe, editorial board member of the journals Formal Aspects of Computing and Journal of Logical and Algebraic Methods in Programming, and steering committee member of several conference series (email: einarj@ifi.uio.no).

Silvia Lizeth Tapia Tarifa (Ph.D. 2014) is an associate professor at the Department of Informatics, University of Oslo. Her research activities are rooted in the application and foundations of formal methods, which touch on the study, understanding and development of formal methods, focusing on abstractions that benefit the specification, design and implementation of complex systems, the development of model-centric methods for distributed and component-based systems, the use of model-centric methods in various problem domains and the combination of formal methods with various existing methods, e.g., semantic technology, process mining, etc. In 2017, she won a young research talent grant, awarded by the Research Council of Norway. She has been contributing to several EU and Norwegian research projects (email: sltarifa@ifi.uio.no).