This work was submitted to:

Chair of Process and Data Science (PADS - Informatik 9), RWTH Aachen University

Translucify: Enhancing Event Logs with Enabled Activities Beyond the Control-Flow Perspective

Bachelor's Thesis

Author: **Geonho Yun**

Student ID: **422305**

Advisors:	Harry H. Beyel, M.Sc. Christopher T. Schwanen, M.Sc.
Examiners:	Prof. Dr. Ir. Wil M. P. van der Aalst Prof. Dr. rer. pol. Stefan Decker

Registration Date: 2024-08-30

Submission Date: 2024-10-21

Abstract

Abstract

This is *not* the final version of the thesis paper, but a draft version needed for the theis registration. The abstract will be updated in the final version of the thesis paper.

Contents

Chapter 1

Introduction

Process mining [1] is a field of computer science aiming to reconstruct the latent process model from a manifest event data. Together with the surge of data availability, process mining has become a hot topic in the fields of data science and business intelligence.

The foundation of all process mining discovery algorithms is the so-called *event log*: A chronologically ordered list of every event recorded in a business system. Event logs have three mandatory feature attributes. *Case ID* represents which case the event belongs to. *Activity* portrays what kind of event happened. *Timestamp* logs the time of occurrence of that specific event. Table 1.1 demonstrates a simple example of an event log.

Table 1.1: Example event log.

Case ID	**Activity**	**Timestamp**	**Resource**	**Cost (Euros)**
001	Start process	2024-10-01 08:30	User A	15
001	Review data	2024-10-01 09:00	User B	30
001	Approve request	2024-10-01 09:45	User C	10
001	Send email	2024-10-01 10:15	User D	5
001	Finish process	2024-10-01 10:25	User A	20
002	Start process	2024-10-01 11:30	User A	15
002	Review data	2024-10-01 11:46	User B	30
002	Finish process	2024-10-01 12:11	User A	10
⋮	⋮	⋮	⋮	⋮

The three major branches of process mining are *process discovery*, *conformance checking*, and *process enhancement* [1]. *Process discovery* is where a process model is extracted from a submitted event log. In *conformance checking*, the quality of the discovered process model is evaluated in terms of various criteria such as fitness, precision, generalization, and simplicity [2]. Finally, in the *enhancement* phase, the idea is to extend or improve the existing process model using the data present in the event log.

1.1 Motivation

Business processes are seldom linear. Instead, they are usually a messy, chaotic, intertangled mash of activities where its golden path is meticulously hidden. On top of that, event logs have no guarantee of completeness. The quality of the process model produced by process-discovery techniques is therefore entirely dependent on the quality of its event log.

1.1.1 Insights of Enabled Activities

In an ideal world where perfect process discovery is conceivable, an event log would be *transparent*, containing metadata of structural properties of the corresponding process model, e.g., state information in a Petri net setting. Event logs, however, are usually *opaque* - one cannot identify the underlying process model straight away by solely looking at the log. One must instead utilize process-discovery algorithms to generate corresponding models. By its nature, process event logs primarily focus on what *happened*. However, they often do not take into consideration what *could have happened* instead. An event log is *translucent* if the log contains the information which alternative activities could have potentially taken place instead of the actual activity occurred in the real world. Logs of this nature are called *translucent event logs* and are extremely beneficial to enhance the quality of existing process-discovery algorithms.

Let us consider a small example as motivation. Suppose the underlying model of our business process is represented as the Petri net below.

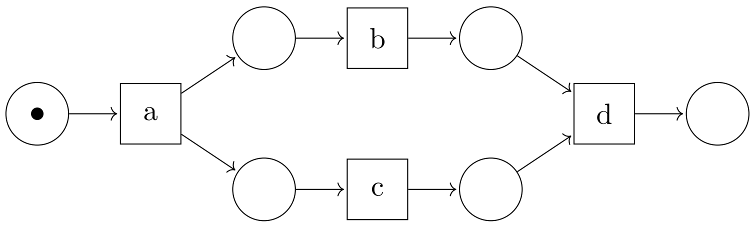


Figure 1.1: Example business model represented as a Petri net.

We play-out the model fifty times and retrieve the event log $\mathcal{L}_1 = [\langle a, b, c, d\rangle^{48}, \langle a, c, b, d\rangle^{2}]$ in the process. We can then leverage widely used process-discovery algorithms to rediscover the process model described in Figure 1.1.

This works under the assumption of log completeness, but what would happen if the trace $\langle a, c, b, d\rangle$ occurs so rarely that we weren't able to capture the behavior? Given the log subset $\mathcal{L}_2 = [\langle a, b, c, d\rangle^{50}]$, every process-discovery algorithm will return a linear process model. The translucent variant would look like $\mathcal{L}_3 = [\langle \underline{a}, \underline{b}c, \underline{c}, \underline{d}\rangle^{50}]$, where all enabled activities are listed and the executed activity is underscored. Here, the choice situation between activities b and c is clearly visible, thus preventing the linear modelling of our process.

Despite its benefits, lucent models and translucent event logs are relatively new concepts and are therefore scarcely researched. As a result, translucent event logs are hardly available in real-life process logs. This motivates us to devise novel methods to generate translucent event logs from a non-translucent event log.

1.1.2 Limitations of Previous Methods

The problem of embedding translucent information in the event log is relatively new and few methods have been proposed so far. This prompts us to explore new methods to annotate event logs with enabled activities.

Among the previously suggesting methods, one of them involves annotating the event log with activities which are in turn pattern-matched by labeling the user's system interface. This requires manual labor of labeling individual patterns and are rather unrealistic for real-life systems with a sizable variety of system interfaces.

Another method suggests replaying the event log on the provided process model and annotating the enabled activities in the log. This method is more feasible, but the quality of the annotation, i.e., the accuracy of the enabled activities, is heavily dependent on the quality of the process model. As process models returned by process-discovery algorithms, such as the Inductive Miner, tend to be underfitting, the method above will likely take a superset of enabled activities for each event, thereby reducing its accuracy.

Current methods of translucent log annotation are solely focused on the control-flow aspect, even though the data attributes of event logs also contain valuable information and have an impact on activity enablement.

1.2 Problem Statement

Taking these aforementioned aspects into account, our problem statement can be formulated as the following.

Translucent Log Extension: *Given an event log and an auxiliary process model, annotate the event log with translucent information while incorporating the data attributes into the computation.*

1.3 Research Questions

In the scope of the thesis, we define the four research questions below:

RQ1. Which techniques can detect enabled activities considering solely the log?

RQ2. Which techniques can detect enabled activities considering the log-model pair?

RQ3. How do these methods compare to each other in terms of accuracy and runtime?

RQ4. How can we design and implement an intuitive, user-friendly tool to demonstrate our results?

1.4 Research Goals

The principal research goal of this thesis paper is to explore different methods to annotate event logs with translucent information. Furthermore, by building a meaningful, easily operable end-user framework dedicated to translucent log annotation, the **Translucify** framework should be able to evaluate these methods systematically with respect to its accuracy and runtime.

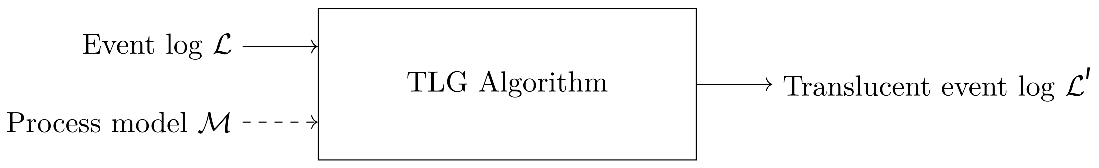

Event log $\mathcal{L}$
Process model $\mathcal{M}$
TLG Algorithm
Translucent event log $\mathcal{L}^{'}$

Figure 1.2: Workflow of Translucify. The algorithm takes an event log $\mathcal{L}$ and an optional process model $\mathcal{M}$, then returns an annotated translucent event log $\mathcal{L}^{'}$.

We hereby explicitly state that while the generated translucent log could be applied for further use, especially for process discovery to enhance previously discovered models, the exact application of generated translucent event logs is not the scope of this thesis. Instead, our sole focus lies on the log enhancement phase.

1.5 Contributions

1.6 Thesis Structure

The remainder of this thesis is structured as follows. We first present related work in Chapter 2. We then introduce basic mathematical preliminaries in Chapter 3. In Chapter 4, the main ideas of translucent event log generation algorithms are presented and thoroughly explained, followed by the implementation specifications in Chapter 5. The evaluation methods and results are displayed in Chapter 6. Finally, discussion of the result, further limitations and future perspectives are presented in Chapter 7.

Chapter 2

Related Work

The following section provides an overview of the related work regarding the thesis, primarily focusing on the fields of data-aware process mining, translucent event logs, and predictive process monitoring.

2.1 Data-Aware Process Mining

Conventional process mining algorithms exclusively focus on the control-flow aspect of the process, i.e., the activity attribute of the event log, thereby ignoring the data attributes the event log provides. Data-aware process mining, on the other hand, attempts to incorporate both the control-flow and data attributes of the event log. The prevalent repertoire is to discover decision points in the model and to annotate them with guard functions, which in turn are discovered with decision trees. This method of process enhancement is called *decision mining* [3–5] and is a well-established field of research within process mining. In [5], Petri nets with a data setting *(DPN)* are proposed, thereby expanding the notation of Petri nets. While previous papers worked with the assumption of deterministic, mutual exclusive transition behavior in decision points, they do not take into account how certain decisions cannot be modeled in a dichotomous manner. Often, one needs a softer classification assumption, stating that data attributes affect the decision probabilistically. A further extension of the concept of data-annotated Petri nets occurs in [6] by integrating stochastic information into the model. In the paper, the authors introduce the concept of Stochastic Labeled Data Petri Nets *SLDPNs* and propose a method to generate an SLDPN from a Petri net and an event log. Each transition will be mapped with its own weight function learned with the activation instances of individual transitions using logistic regression.

2.2 Translucent Event Logs

Add the new paper ActivityGen as soon as it's published

Little research has been performed on the topic of translucent event logs. Being a relatively young concept in the field of process mining, they were first hinted in 2018 by [7], where a possible event log revealing the set of enabled activities is mentioned. [8] formally introduces and defines translucent event logs and relates concepts of lucency and translucency by showing that a lucent process model can be rediscovered by using a translucent event log retrieved from the model.

Methods of creating translucent event logs are first discussed in [9]. Here, a system's screenshot is matched with the labelled activity pattern and annotated with the corresponding activities. Furthermore, a model-based approach is introduced by replaying the event log on the model and annotating enabled activities. In [10], the authors demonstrate how using information of enabled activities can be utilized to discover a process model of equipotential quality in comparison to conventional process discovery algorithms. This is done by extending the Inductive Miner [11] using newly defined translucent activity relationships. Lastly, a precision measure between a Petri net and a translucent event log is formally introduced in [12] by comparing log-enabled activities and model-enabled activities.

2.3 Predictive Process Monitoring

TODO: Make a clear distinction between the current section and methods used data-aware process mining, e.g., decision mining!

- Explain the term and list previous different approaches
- Lin et al. [13]: Next event and next data attribute prediction using RNNs
- Gunnarsson et al. [14]: Remaining trace and runtime prediction using LSTMs
- Bukhsh et al. [15]: Next activity, next event time, and remaining time prediction using transformer architecture
- General overview of the field is given in [16].

Chapter 3

Preliminaries

In the upcoming section, we will introduce basic mathematical foundations essential for the thesis. These encompass the definitions of sets, multisets, sequences, Petri nets, and event logs. We will also introduce the concept of transition systems and prefix automata. Lastly, we will provide a cursory explanation of machine learning algorithms such as multivariable regression as well as deep learning algorithms such as the transformer architecture.

Definition 3.1 (Set)**.** A set is a collection M of distinct objects. The objects in a set are called *elements* of the set. The set M is denoted as $M = \{a_1, a_2, \ldots, a_n\}$, where $a_1, a_2, \ldots, a_n$ are the elements of the set. The notation $|M|$ denotes the cardinality of M, i.e., the number of elements in M.

Throughout the thesis, we will use the variant $\mathbb{N} = \{0, 1, 2, \ldots\}$ where 0 is included in the set of natural numbers.

Definition 3.2 (Finite Sequence)**.** A finite sequence over a set A is a function with the signature $\sigma\colon S \to A$ where $S = \{0, 1, \cdots, n\} \subseteq \mathbb{N}$ and $|S| < \infty$, i.e., a finite set of initial natural numbers. The set of all finite sequences over A is denoted as A^*. The cardinality of the domain of $\sigma|S|$ is the *length* of a sequence σ and is denoted it with $|\sigma|$.

Since we are introducing the notion of sequences to represent event log traces, note that we only consider finite sequences in the scope of this thesis. Instead of the function notation, we predominantly use the list notation with angled brackets to represent sequences. For example, the sequence $\sigma\colon \{0, 1, 2, 3\} \to \{a, b, c\}, 0 \mapsto a, 1 \mapsto b, 2 \mapsto b, 3 \mapsto c$ is expressed as $\langle a, b, b, c \rangle$.

Definition 3.3 (Multiset)**.** A multiset over a set A is a function with the signature $\mu\colon A \to \mathbb{N}$, where $\mu(a)$, $a \in A$ denotes how many times the element a occurs in the multiset. A multiset over a sequence σ is a function with the signature $\mu\colon A \to \mathbb{N}$, where $\mu(a) = |\{i \mid \sigma(i) = a\}|$. The set of all multisets of a set A is denoted with $\mathbb{B}(A)$.

Similar to sequences, we preferably use the list notation with square brackets to represent multisets rather than the function notation, where the cardinality of each element is superscripted over the element. For example, the multiset over the sequence $\langle a, b, b, c \rangle$ will be represented as $[a^1, b^2, c^1]$.

Furthermore, for two multisets X and Y over A, we notate the multiset union as $X \uplus Y$, where $(X \uplus Y)(a) = X(a) + Y(a)$ for all $a \in A$. Likewise, the difference of two multisets is denoted as $X \setminus Y$, where $(X \setminus Y)(a)$ is the maximum value between $X(a) - Y(a)$ and 0 for all $a \in A$.

3.1 Event Logs

In this section, we formally define the structure of events and event logs. Furthermore, the concept of translucent event logs is introduced.

Definition 3.4 (Event)**.** $\mathcal{E}$ is the event universe. An *event* $e \in \mathcal{E}$ is a logical abstraction of a real-life process event. An event possesses multiple named attributes. We define the universe of all attribute names as $\mathcal{AN}$ and the universe of all attribute values as $\mathcal{AV}$.

Based on this, we define the attribute projection function $\pi \colon \mathcal{E} \times \mathcal{AN} \to \mathcal{AV} \cup \{\perp\}$, where π is a partial function mapping the attribute name of every event to an attribute value (otherwise a none value $\perp$). Following the convention in [1], we denote the signature $\pi(e, n)$ as $\pi_n(e)$ for all $e \in \mathcal{E}, n \in \mathcal{AN}$.

Subsequently, a collection of events form an event log of a process.

Definition 3.5 (Event Log)**.** Let $\mathcal{C}, \mathcal{A}, \mathcal{T} \subseteq \mathcal{AV}$, where $\mathcal{C}$ is the universe of case identifiers, $\mathcal{A}$ the universe of activity names, and $\mathcal{T}$ the universe of timestamps. An event log $\mathcal{L} \subseteq \mathcal{E}$ is a subset of the event universe such that for all events $e \in \mathcal{L}$:

- $\pi_{case}(e) \in \mathcal{C}$ is the case identifier,
- $\pi_{act}(e) \in \mathcal{A}$ is the activity name,
- $\pi_{time}(e) \in \mathcal{T}$ is the timestamp.

We further assume that in a conventional event log, there exists a total order $<_{time}$ on $\mathcal{L}$ such that $e <_{time} e' \iff \pi_{time}(e) < \pi_{time}(e')$ holds for all $e, e' \in \mathcal{L}$, i.e., the events are sorted by their timestamps in chronological order.

We can further group events by their case identifiers. A sequence of events with the same case identifier ordered by $<_{time}$ is called a *trace*. Note that the set of all traces of an event log is pairwise disjoint, i.e., there is no event $e \in \mathcal{E}$ which is an element of two different traces. Hence, an event log can also be represented as a set of traces.

Definition 3.6 (Trace)**.** $\mathcal{E}$ is the event universe. A *trace* is a sequence of events $\sigma = \langle e_1, e_2, \ldots, e_n \rangle \in \mathcal{E}^*$ such that $\pi_{case}(e_i) = \pi_{case}(e_{i+1})$ holds for all $i \in \{1, \ldots, n-1\}$.

However, when discussing traces, we oftentimes refer to them as a sequence of activities. This is firstly for the sake of simplicity, but also due to the fact that the control flow is considered to be the most crucial aspect in process mining, in particular when constructing process models. Event logs where each trace is solely represented as a sequence of activities is called a *simple event log*.

Definition 3.7 (Simple Event Log)**.** Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log and $\sigma \in \mathcal{L}$ a trace. We expand the attribute projection function analogously for traces as the following: $\pi_n(\sigma) = \pi_n(\langle e_1, e_2, \ldots, e_n \rangle) = \langle \pi_n(e_1), \pi_n(e_2), \ldots, \pi_n(e_n) \rangle$. $\mathcal{L}'$ is a simple event log of $\mathcal{L}$ if

$$\mathcal{L}' = [\pi_{act}(\sigma) \mid \sigma \in \mathcal{L}].$$

Since we are projecting only the activity names of each event, we lose the uniqueness of each event, resulting in losing the uniqueness of each trace as well. In the simple event log setting, we therefore need to represent the event log as a multiset of traces.

The objective of our thesis is to transform conventional event logs into translucent event logs by annotating each event $e \in \mathcal{E}$ with an additional attribute *en*. *en* specifies all activities which could have happened the moment $\pi_{act}(e)$ occurred. We formally define the notion of translucent event logs below.

Definition 3.8 (Translucent Event Log)**.** Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log. $\mathcal{L}$ is *translucent* if and only if $\pi_{en}(e) \subseteq \mathcal{A}$ and $\pi_{act}(e) \in \pi_{en}(e)$ holds for all $e \in \mathcal{L}$.

Table 3.1 displays an example translucent event log. Note that the occurred activity in the activity column is always included in the set of enabled activities.

Table 3.1: Example translucent event log table.

Case ID	**Activity**	**Timestamp**	**Resource**	**Cost**	**Enabled Activities**
001	S	2024-10-01 08:30	User A	15	{S}
001	R	2024-10-01 09:00	User B	30	{R, A}
001	A	2024-10-01 09:45	User C	10	{A, F}
001	E	2024-10-01 10:15	User D	5	{E, F}
001	F	2024-10-01 10:25	User A	20	{F}
002	S	2024-10-01 11:30	User A	15	{S}
002	R	2024-10-01 11:46	User B	30	{R, F}
002	F	2024-10-01 12:11	User A	10	{F}
⋮	⋮	⋮	⋮	⋮	⋮

3.2 Automata

In the following section, two types of automata used throughout the thesis will be defined, namely Petri nets and prefix automata.

3.2.1 Petri Nets

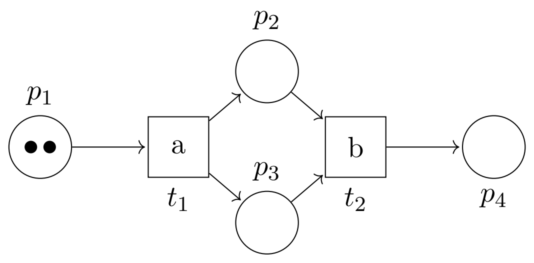


Figure 3.1: Example of a marked Petri net.

Petri nets are the standard process model used in process mining, whose biggest advantage is the ability to model concurrent systems. Transitions of a Petri net usually correspond to event activities, whereas markings of a Petri net correspond to states of the process. In the following, we begin with the definition of a Petri net and continue to expand its definition in order to express labeling transitions, marking places, and firing transitions. Finally, we augment the Petri net even further to incorporate randomness and data attributes into the model.

Definition 3.9 (Petri Net)**.** Let P, T be finite, disjoint sets, where T is a set of *places* and T set of *transitions.* A *Petri net* is a triple $N = (P, T, F)$, where $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs between places and transitions.

Figure 3.1 portrays a Petri net with four places $P = \{p_1, p_2, p_3, p_4\}$, two transitions $T = \{t_1, t_2\}$, and arcs $F = \{(p_1, t_1), (t_1, p_2), (t_1, p_3), (p_2, t_2), (p_3, t_2), (t_2, p_4)\}$. Since Petri nets are used to model the control flow of a process based on an event log, transitions should be able to represent an activity name. This is where Labeled Petri nets come into play in order to label transitions with activity names.

Definition 3.10 (Marked Labeled Petri Net)**.** Let $N = (P, T, F)$ be a Petri net. A *labeled Petri net* is an extended tuple $N = (P, T, F, \mathcal{A}, l)$, where $\mathcal{A}$ is a set of activity labels and $l : T \to \mathcal{A}$ is a labeling function. A *marked Petri net* is an ordered pair (N, M) where $M \in \mathbb{B}(P)$ is a multiset over P.

As shown in Figure 3.1, the transitions t_1 and t_2 are each labeled with activity names a and b respectively. Furthermore, place p_1 is marked with two tokens, while the rest is unmarked. The marking is therefore denoted as $M = [p_1^2, p_2^0, p_3^0, p_4^0]$.

Furthermore, there are cases where a transition does not correspond to any of the activity names in the event log, as some are, e.g., only used to model the control flow of a Petri net. These transitions are called *silent transitions* (also τ-transitions) and are denoted with the activity name τ.

Definition 3.11 (Enabled Transition)**.** Let (N, M) be a marked Petri net. We further define the $\bullet$ notation, where ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$. A transition $t \in T$ is *enabled* in M if and only if ${}^\bullet t \leq M$.

We also characterize this property using the notation $(N, M)[t\rangle$. The set of enabled transitions in a marked Petri net (N, M) is denoted as $en(N, M) = \{t \in T \mid {}^\bullet t \leq M\}$. An enabled transition can be *fired*, which removes a token from each of its input places and adds a token to each of its output places.

Definition 3.12 (Firing Rule)**.** Let (N, M) be a marked Petri net. The firing of a transition $t \in T$ is denoted as $(N, M)[t\rangle(N, M')$, where $M' = (M \setminus^\bullet t) \cup t^\bullet$.

Let $\sigma = \langle t_1, t_2, \dots, t_n\rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma\rangle(N, M')$ denotes that there exists a sequence of markings $\langle M_1 = M, M_2, \dots, M_{n+1} = M'\rangle$ so that
$(N, M_i)[t_i\rangle(N, M_{i+1})$ for all $1 \leq i \leq n$. The set of all reachable markings from M is denoted as $[N, M\rangle = \{M' \mid (N, M)[\sigma\rangle(N, M')\}$ for some $\sigma \in T^*$. Figure 3.2 takes the previously presented Petri net in Figure 3.1 and demonstrates the firing of a transition sequence $\langle t_1, t_2\rangle$.

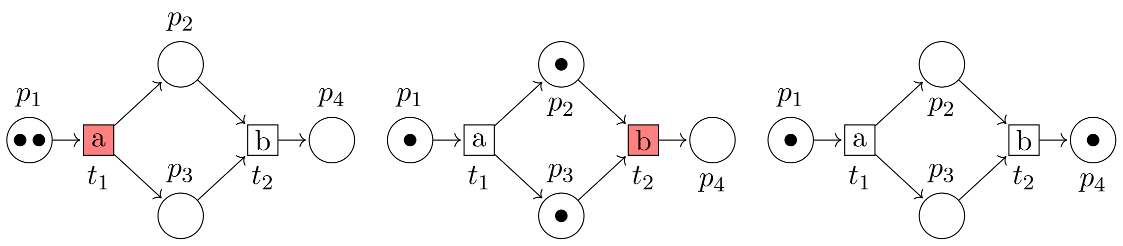


Figure 3.2: Firing of a transition sequence $\langle t_1, t_2 \rangle$ in an example Petri net.

Definition 3.13 (Lucency)**.** Let (N, M) be a marked Petri net. (N, M) is *lucent* if and only if:

$$\forall M_1, M_2 \in [N, M\rangle : en(N, M_1) = en(N, M_2) \Rightarrow M_1 = M_2.$$

As previously mentioned in Chapter 2, [6] introduces an extension of a Petri net by consolidating the stochastic properties and data states into a conventional Petri net.

In a stochastic setting, the firing probability of each transition is regulated by a single weight function. If more than one transition is enabled, the transition to be fired is selected randomly based on its weights. In other words, for a marked Petri net (N, M), there exists a weight function $w : T \to \mathbb{R}^+$ so that

$$Pr(t \mid M') = \frac{w(t)}{\sum_{t' \in en(N,M')} w(t')}$$

for all reachable markings M' and enabled transitions $t \in en(N, M')$. This notion is extended even further in [6] by allocating a separate weight function to each transition dependent on the data states. It is evident that the identical notion can be denoted as a single function for the entire Petri net for the sake of simplicity, where the function value is dependent on the Cartesian product of the transition and data state.

Definition 3.14 (Stochastic Labeled Data Petri Net)**.** Δ is the universe of data states. Let further $N = (P, T, F, \mathcal{A}, l)$ be a labeled Petri net. A tuple $(P, T, F, \mathcal{A}, l, \omega)$ is a *Stochastic Labeled Data Petri Net (SLDPN)* if and only if $(P, T, F, \mathcal{A}, l)$ is a labeled Petri net and $\omega : T \times \Delta \to \mathbb{R}^+$ is a weight function.

Analogous to the previous formulation, given a reachable marking M' and a data state $\delta \in \Delta$ in an SLDPN $(P, T, F, \mathcal{A}, l, \omega)$, the firing probability of an enabled transition $t \in en(N, M')$ is

$$Pr(t \mid M', \delta) = \frac{\omega(t, \delta)}{\sum_{t' \in en(N,M')} \omega(t', \delta)}.$$

Nur die Definition hier ist ein bisschen unverständlich, aber ein Beispiel für SLDPNs taucht eigentlich in Methods wieder auf. Soll ich hier trotzdem ein Beispiel zeigen?

3.2.2 Prefix Automata

Definition 3.15 (Transition System)**.** Let S be the set of states, A the set of activities, and $T \subseteq S \times A \times S$ the set of transitions. A *transition system* is a triple $TS = (S, A, T)$. We denote the set of initial states as S^{start} and the set of final states as S^{end}, where $S^{start}, S^{end} \subseteq S$.

show example of a TS

Transition systems are an alternative method next to Petri nets to represent processes of a system. Due to its structural property, silent transitions are absent in transition systems. This relieves us from the need to compute alignments when replaying the event log on the model.

Definition 3.16 (Prefix Automaton)**.** Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log and $l^{state}: \mathcal{L} \times \mathbb{N} \to S$ a state representation function. $TS_{\mathcal{L},l^{state}} = (S, A, T)$ is a transition system based on $\mathcal{L}$ and l^{state} with the following properties:

- $S = \{l^{state}(\sigma, k) \mid \sigma \in \mathcal{L} \wedge 0 \leq k \leq |\sigma|\}$ the state space,
- $A = \{\sigma(k) \mid \sigma \in \mathcal{L} \wedge 1 \leq k \leq |\sigma|\}$ the set of activities of the event log,
- $T = \{(l^{state}(\sigma, k), \sigma(k+1), l^{state}(\sigma, k+1)) \mid \sigma \in \mathcal{L} \wedge 0 \leq k < |\sigma|\}$ the set of transitions,
- $S^{start} = \{l^{state}(\sigma, 0) \mid \sigma \in \mathcal{L}\}$ the set of initial states, and
- $S^{end} = \{l^{state}(\sigma, |\sigma|) \mid \sigma \in \mathcal{L}\}$ the set of final states.

Taking the simple event log $\mathcal{L}_1 = [\langle a, b, c, d\rangle^{48}, \langle a, c, b, d\rangle^{2}]$ as an example, we can construct a prefix automaton $TS_{L_1,hd}$ as follows:

- $S = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle, \langle a, b, c\rangle, \langle a, b, c, d\rangle, \langle a, c\rangle, \langle a, c, b\rangle, \langle a, c, b, d\rangle\}$
- $A = \{a, b, c, d\}$
- $T = \{(\langle\rangle, a, \langle a\rangle), (\langle a\rangle, b, \langle a, b\rangle), (\langle a, b\rangle, c, \langle a, b, c\rangle), (\langle a, b, c\rangle, d, \langle a, b, c, d\rangle), (\langle a\rangle, c, \langle a, c\rangle), (\langle a, c\rangle, b, \langle a, c, b\rangle), (\langle a, c, b\rangle, d, \langle a, c, b, d\rangle)\}$
- $S^{start} = \{\langle\rangle\}$
- $S^{end} = \{\langle a, b, c, d\rangle, \langle a, c, b, d\rangle\}$

The graphical representation is shown in Figure 3.3.

Frequently used examples of a state representation function take the prefix function hd^k as its baseline mechanism. Among diverse methods of process representation using transition systems, we focus on the list, set, and multiset representations.

TODO: Definition für Representations verfeinern

The set- and multiset-based representations of the prefix automaton $TS_{L_1,hd}$ are shown in Figures 3.4 and 3.5, respectively.

All these three variants have their own advantages and disadvantages. The list representation is the most precise, as it preserves the activity order of each trace. However, analyzing parallel situations is difficult due to the very same characteristics of order preservation.

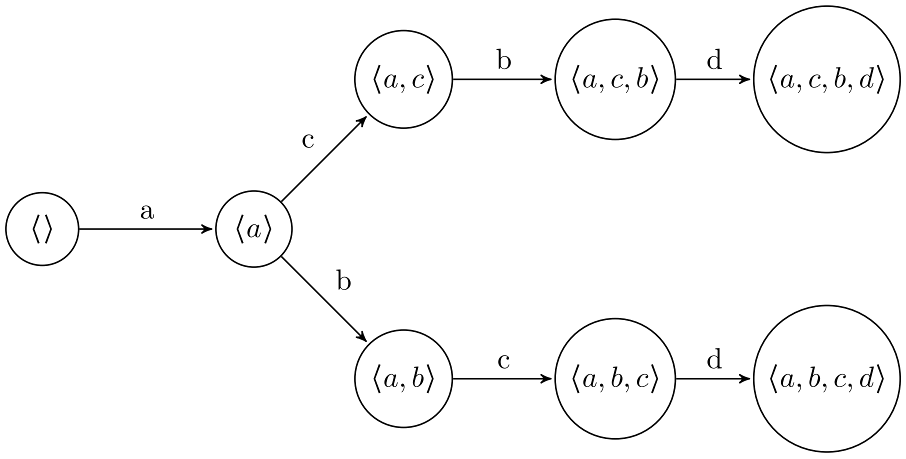

$$\begin{tikzcd}
 & & \langle a,c\rangle \arrow[r, "b"] & \langle a,c,b\rangle \arrow[r, "d"] & \langle a,c,b,d\rangle \\
\langle\rangle \arrow[r, "a"] & \langle a\rangle \arrow[ur, "c"] \arrow[dr, "b"'] & & & \\
 & & \langle a,b\rangle \arrow[r, "c"] & \langle a,b,c\rangle \arrow[r, "d"] & \langle a,b,c,d\rangle
\end{tikzcd}$$

Figure 3.3: Example of a prefix automaton $TS_{L_1,hd}$ for the simple event log $\mathcal{L}_1 = [\langle a,b,c,d\rangle^{48}, \langle a,c,b,d\rangle^{2}]$.

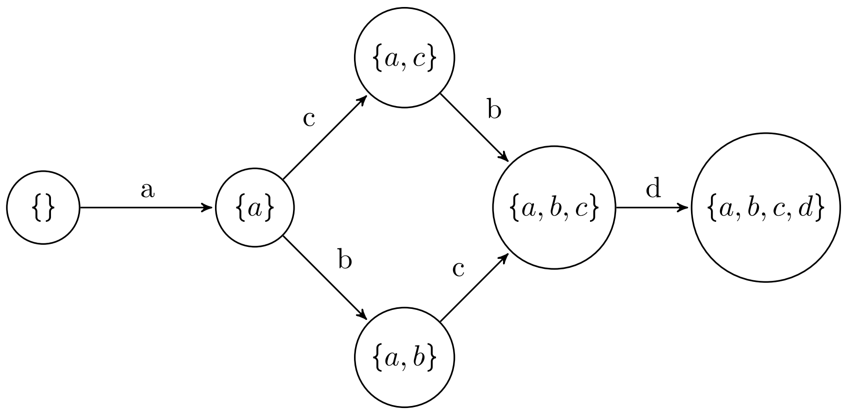

$$\begin{tikzcd}
 & & \{a,c\} \arrow[dr, "b"] & & \\
\{\} \arrow[r, "a"] & \{a\} \arrow[ur, "c"] \arrow[dr, "b"'] & & \{a,b,c\} \arrow[r, "d"] & \{a,b,c,d\} \\
 & & \{a,b\} \arrow[ur, "c"'] & &
\end{tikzcd}$$

Figure 3.4: Set-based representation of a prefix automaton $TS_{L_1,hd}$ for the simple event log $\mathcal{L}_1 = [\langle a,b,c,d\rangle^{48}, \langle a,c,b,d\rangle^{2}]$.

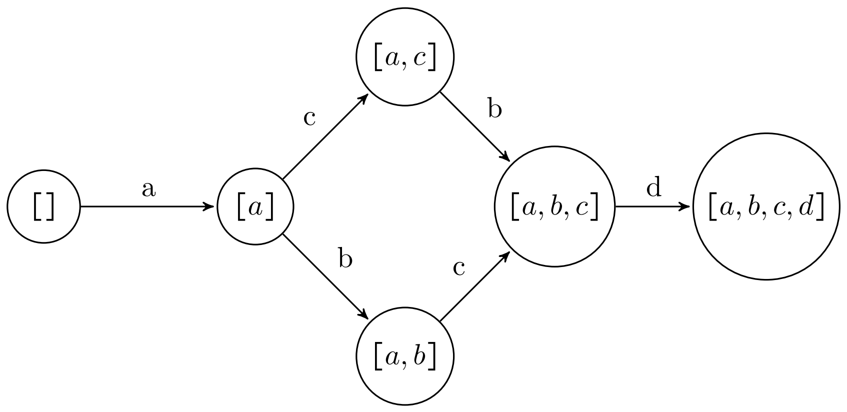

$$\begin{tikzcd}
 & & {[a,c]} \arrow[dr, "b"] & & \\
{[\,]} \arrow[r, "a"] & {[a]} \arrow[ur, "c"] \arrow[dr, "b"'] & & {[a,b,c]} \arrow[r, "d"] & {[a,b,c,d]} \\
 & & {[a,b]} \arrow[ur, "c"'] & &
\end{tikzcd}$$

Figure 3.5: Mutliset-based representation of a prefix automaton $TS_{L_1,hd}$ for the simple event log $\mathcal{L}_1 = [\langle a,b,c,d\rangle^{48}, \langle a,c,b,d\rangle^{2}]$.

Moreover, the representation fails at recognizing loops in the inherent process model and is therefore susceptible to overfitting.

The set representation, on the other hand, is more general and does not suffer from order preservation issues as the list representation does. Although the property of order negligence is practical for local loops and parallel situations, it is critical for larger loops. Consider a situation with nested loops, or a loop where all activities have been already visited in the first iteration. This would lead to drastic state simplification and the resulting information loss. Here, it is crucial to select an optimal prefix length k.

The multiset representation is a compromise between the list and set representations. However, it is still not able to correctly capture the loop behavior as each iteration would create a new set of states.

3.3 Machine Learning Algorithms

In this section, we will briefly introduce and explain the machine learning algorithms employed throughout the thesis. These include multivariable logistic regression, random forest, and the transformer architecture.

3.3.1 Multivariable Logistic Regression

Multivariable logistic regression is a statistical model used to predict the probability of a boolean or binary outcome based on multiple variables, and is commonly applied in classification tasks. Formally, we define the model as follows:

Definition 3.17 (Multivariable Logistic Regression)**.** Given a dataset $D \in (\mathbb{R}^n \times \{0,1\})^m$, find a function $f : \mathbb{R}^n \to [0,1]$ such that:

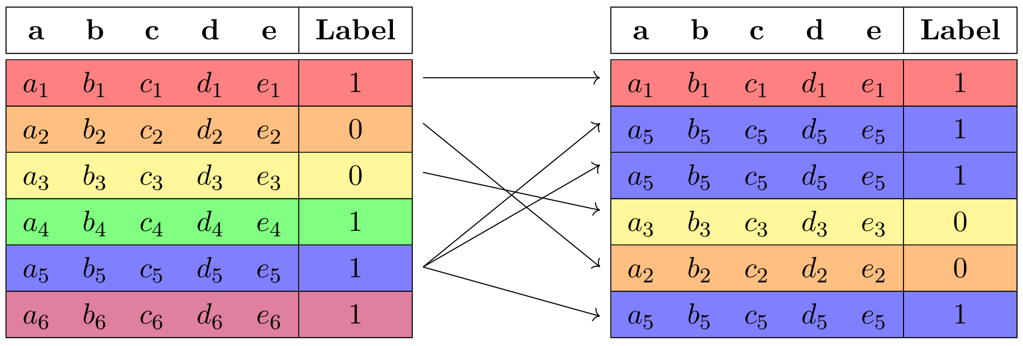
$$f(x) = \frac{1}{1 + e^{-\beta^T x}}, \text{ where}$$

$$\beta = \arg\min_{\beta} \sum_{(x,y) \in S} \ln\left(1 + e^{-y\beta^T x}\right).$$

3.3.2 Random Forest

Random Forest is an ensemble learning method that creates multiple, logically simpler decision trees and combines their predictions in order to create more accurate predictions. In contrast to conventional decision trees where the model consists of a single tree with a deep structure, Random Forests are composed of several trees with a shallow structure. The naming of the algorithm is derived from two distinct sources of randomness, namely bootstrap aggregation and random feature selection.

Bootstrap aggregation, also known as *bagging*, is a dataset creation technique in which a multitude of datasets are created by sampling from the original dataset with replacement, i.e., a sample can be selected multiple times. Each of these bagged datasets is then fed to train a decision tree. Figure 3.6 illustrates an example of a bagged dataset.



a	b	c	d	e	Label
a_1	b_1	c_1	d_1	e_1	1
a_2	b_2	c_2	d_2	e_2	0
a_3	b_3	c_3	d_3	e_3	0
a_4	b_4	c_4	d_4	e_4	1
a_5	b_5	c_5	d_5	e_5	1
a_6	b_6	c_6	d_6	e_6	1

a	b	c	d	e	Label
a_1	b_1	c_1	d_1	e_1	1
a_5	b_5	c_5	d_5	e_5	1
a_5	b_5	c_5	d_5	e_5	1
a_3	b_3	c_3	d_3	e_3	0
a_2	b_2	c_2	d_2	e_2	0
a_5	b_5	c_5	d_5	e_5	1

Figure 3.6: Example of bagging. Each row is randomly selected from the original dataset. Sampling with replacement is clearly visible through multiple instances of the fifth row.

The term random feature selection refers to the procedure of utilizing a randomly selected subspace of the feature space when finding the optimal split at a decision tree's node. This method is reported to improve the model's accuracy and make the model less prone to outliers [17]. Figure 3.7 demonstrates the process of random feature selection.

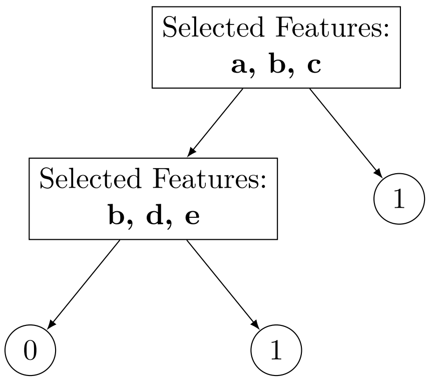


Figure 3.7: Illustration of random feature selection.

After the set of decision trees is trained, the model then aggregates the predictions of individual trees to compute the final prediction. This is done by, e.g., taking a majority vote of the prediction among the set of trees. For a deeper theoretical background, we refer to the papers [18] and [17].

3.3.3 Transformer Architecture

Compared to earlier models such as RNNs or LSTMs mentioned in Chapter 2, the transformer is a newer form of architecture first introduced in 2017 with [19]. The centerpiece of transformers is the self-attention mechanism, enabling the model to express interdependent semantic context between word tokens.

When given an input sequence, the transformer first splits up the input sequence into word embeddings known as tokens. Each token t_i is then multiplied with parameter matrices W^Q, W^K, W^V in order to obtain its query, key, and value vectors q_i, k_i, v_i respectively. Note that the parameter matrices are learned during the training phase.

For each token t_i, we then compute the attention score w.r.t. all other tokens in the sequence by taking the dot product of the query vector q_i and the key vector of all tokens $k_1, \cdots, k_n$. The scalar product is also described as the *compatibility function* of the query-key pairs. The attention scalar is ultimately employed as weights for the value vectors $v_1, \cdots, v_n$, whose weighted sum is a representation of t_i where the contextual information of peer tokens are imputed.

It is often characterized by the equation

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \text{ where}$$

$$Q = \begin{pmatrix} | & | & & | \\ q_1 & q_2 & \ldots & q_n \\ | & | & & | \end{pmatrix}, K = \begin{pmatrix} | & | & & | \\ k_1 & k_2 & \ldots & k_n \\ | & | & & | \end{pmatrix}, V = \begin{pmatrix} | & | & & | \\ v_1 & v_2 & \ldots & v_n \\ | & | & & | \end{pmatrix}$$

are the query, key, and value matrices, respectively containing the query, key, and value vectors of each token as matrix columns, and d_k is the dimension of the query and key matrices. Figure 3.8 provides a graphical representation of the self-attention mechanism.

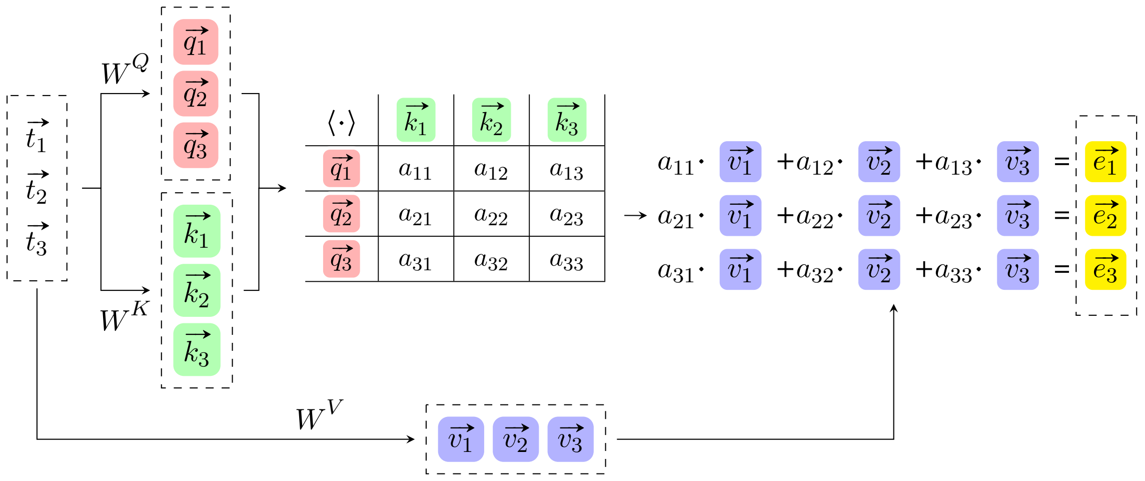

$\vec{t_1}$
$\vec{t_2}$
$\vec{t_3}$
W^Q
W^K
W^V
$\vec{q_1}$
$\vec{q_2}$
$\vec{q_3}$
$\vec{k_1}$
$\vec{k_2}$
$\vec{k_3}$
$\langle\cdot\rangle$
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
$a_{11}\cdot \vec{v_1} + a_{12}\cdot \vec{v_2} + a_{13}\cdot \vec{v_3} = \vec{e_1}$
$a_{21}\cdot \vec{v_1} + a_{22}\cdot \vec{v_2} + a_{23}\cdot \vec{v_3} = \vec{e_2}$
$a_{31}\cdot \vec{v_1} + a_{32}\cdot \vec{v_2} + a_{33}\cdot \vec{v_3} = \vec{e_3}$
$\vec{v_1}$ $\vec{v_2}$ $\vec{v_3}$

Figure 3.8: An example illustration of the self-attention mechanism with three input tokens.

For further reference, the definitive paper on transformer models is [19].

Chapter 4

Translucifying the Event Log

In the upcoming chapter, we discuss the methodology of *Translucify* in detail. We begin with a general framework overview and discuss the specific approaches individually.

4.1 Framework Overview

In order to look deeper into the main problem of this thesis, we first formally define our problem as described below.

Definition 4.1 (Translucent Log Extension Problem)**.** Given an event log $\mathcal{L} \subseteq \mathcal{E}$ as input, produce a translucent event log $\mathcal{L}'$ where the set of enabled activities are added as attributes.

There is a variant of the *Translucent Log Extension Problem*, where a process model is provided along with the event log.

Definition 4.2 (Translucent Log Extension Problem - Process Model Variant)**.** Given an event log $\mathcal{L}$ and a process model $\mathcal{M}$ as inputs, produce a translucent event log $\mathcal{L}'$ where the set of enabled activities are added as attributes.

The second variant differs from the first, as the model is meant to act as a reference model; the set of enabled activities is intended to be constrained by the process model. Note that the function of $\mathcal{M}$ is to provide an upper bound on the set of enabled activities, and the log is there to provide further constraints to enrich the model. Of particular interest are parallel and choice situations, since we are able to deduce supplementary patterns not demonstrated in the process model using log data. We name the first variant defined in Definition 4.1 as *Bottom-up Translucent Log Extension Problem*, whereas the second variant in Definition 4.2 is named as *Top-down Translucent Log Extension Problem.*

Table 4.1: A table of available model-based options for *Translucify.*

	Petri Net	**Prefix Automaton**
Logistic Regression	LR with Petri nets	LR with prefix automata
Random Forests	RF with Petri nets	RF with prefix automata

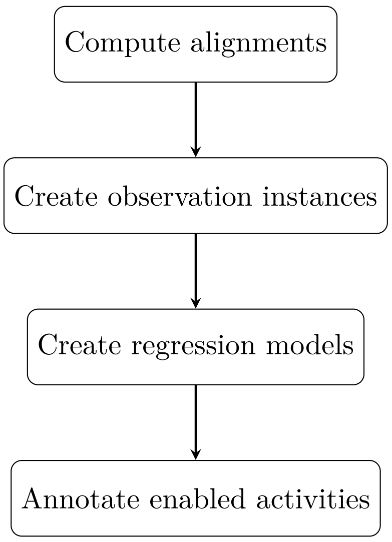


Figure 4.1: Stages of multivariable regression on an SLDPN.

4.2 Petri Net-based Approaches

The basic notion of Petri nets was introduced in the previous Chapter 3. In this section, we will discuss how Petri nets are employed to tackle the *Translucent Log Extension Problem.*

Given a Petri net and an event log, we utiltize the alignment-based appraoch presented in [9] as its baseline algorithm. After computing the alignment for each trace, the program replays the alignment on the reachabiliy graph trace by trace in order to circumvent the silent transitions. For each activity, the algorithm then augments the event log with the corresponding transitions situated in outgoing arcs of the current state. After creating a basic translucent event log, the log can be refined by further algorithms.

...

When annotating the event log with enabled activities, it might make sense to assume that not all activities enabled in the model are enabled in reality. The model might be too permissive to guarantee its fitness to the event log, and the set of enabled activities depicted in the model should therefore be considered as an upper bound of the actual set. Here, instead of applying a hard-line policy by adding guards to the transitions as done in the field of decision mining, we can utilize a probabilistic approach to filter out the activities. An important question would be how we should compute the transition probability of the model. This is where the SLDPN model comes into play.

A method to incorporate the data attributes is to implement multivariable regression. Similar to the setting in [6], we construct a training data set for each transition of a Petri net consisting of data attributes and a boolean label indicating whether the transition was executed given the data attributes as input. We then perform a regression analysis on the training data set for each transition. The resulting dictionary of transitions and regression functions can be employed to filter out transitions lying below a certain probability threshold p in each decision point during replay.

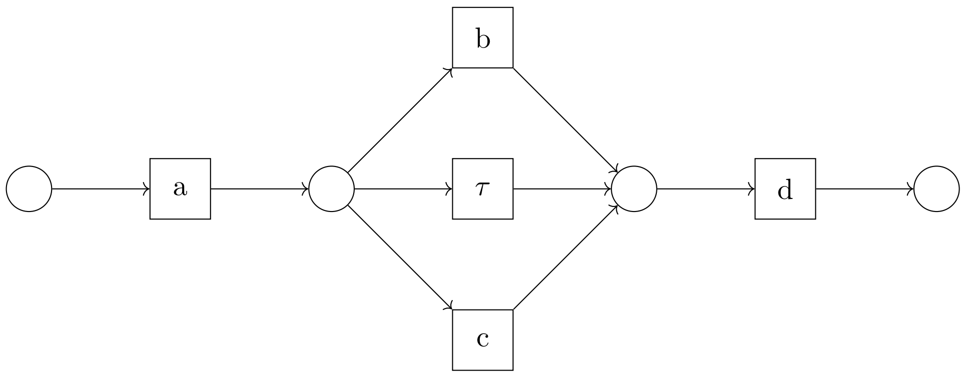


Figure 4.2: Example Petri net for multivariable regression.

Table 4.2: Example event log for multivariable regression.

Case ID	Activity	Timestamp	Family History	Amount
101	a	2024-08-01 08:00:00	yes	30
101	d	2024-08-01 08:30:00	yes	50
102	a	2024-08-02 09:00:00	no	0
102	b	2024-08-02 09:45:00	no	40
102	d	2024-08-02 10:30:00	no	56

Computing Alignments

A process model does not always guarantee a perfect fitness of the event log. Sometimes, infrequent traces are considered as noise and are subsequently ignored. Furthermore, process models contain more than one unique fitting path for each trace in various occasions. It is therefore crucial in Petri nets to first compute the best-fitting model trace for each log trace using *alignments* [20].

(theoretical background & explanation on how alignments usually work)

Computing alignments can be thought as choosing an alignment function $f_{align}\colon \mathcal{L} \to T^*$, where given a log trace $\sigma \in \mathcal{L}$ as input, f_{align} returns a sequence of transitions $\langle t_1, \ldots, t_n \rangle$ as output, corresponding to the optimal model trace computed by a chosen alignment algorithm.

In our scenario, however, we need to take data attributes into account as well, which are necessary for the regression analysis. For this, the appropriate data attributes of the log trace must be attached to the model trace. Intuitively, since event logs are records of a system, we assume that the data state is the data snapshot immediately after the event occurrence. Moreover, we assume that silent transitions do not modify the data state. Data states of silent transitions in the model trace are therefore identical to the most recently occurred data state of a named transition recorded in the event log.

We therefore extend the alignment function to $f_{align}\colon \mathcal{L} \to (T \times \Delta)^*$, where Δ is the set of data attributes found in the corresponding event $e \in \sigma$. Given a log trace $\sigma \in \mathcal{L}$ and the corresponding data attributes $d \in \Delta$, $f_{align}(\sigma) = \langle (t_1, d_1), \ldots, (t_n, d_n) \rangle$ returns a sequence of transitions as output.

Using the example event log in Table 4.2 and the model in Figure 4.2, the alignment for the trace $\langle a, d\rangle$ would be $((a,(yes,30)),(t1,(yes,30)),(d,(yes,50)))$, and the alignment for trace $\langle a,b,d\rangle$ would be $((a,(no,0)),(b,(no,40)),(d,(no,56)))$.

Creating Observation Instances

Given an aligned model trace, our goal is to generate a labeled training data set for each transition in the Petri net model. In other words, the goal of this stage is to find the function $f_{observe}\colon T \to (\Delta \times \{0,1\})^*$.

We iterate over the event log and replay the Petri net using the model trace $f_{align}(\sigma)$ $\forall \sigma \in \mathcal{L}$. For each Petri net marking M_i where $(N, M_0)[\langle t_1, \ldots, t_{i-1}\rangle\rangle(N, M_i)$ and the corresponding current data state d_i, we first compute the enabled transitions $en(M_i)$ and update the values of $f_{observe}$ as the following: $f_{observe}(t_j) \leftarrow f_{observe}(t_j) \cup \{(d_i, c)\}$ for all $t_j \in en(M_i)$, where

$$c = \begin{cases} 1 & \text{if } t_i = t_j \\ 0 & \text{otherwise.} \end{cases}$$

Looking at our running example, the observation instance function $f_{observe}$ would be as follows:

$$f\colon t \mapsto \begin{cases} \Big[((yes,30),1),((no,0),1)\Big], & t = a \\ \Big[((yes,30),0),((no,0),1)\Big], & t = b \\ \Big[((yes,30),0),((no,0),0)\Big], & t = c \\ \Big[((yes,30),1),((no,40),1)\Big], & t = d \end{cases}$$

Create regression models

After creating the observation instances, we train a logistic regression model for each transition in the Petri net model. The goal of this stage is to find the function $f_{regress}\colon T \times \Delta \to [0,1]$, which returns the probability of a transition being enabled given the data attributes.

Annotating the log with enabled activities

After training a logistic regression model for each transition, the goal of this stage is to utilize the regression functions and the alignments to add enabled activities to events listed in each trace. In other words, we need to compute a function $f_{annotate}\colon \mathcal{L} \times [0,1] \to \mathcal{L}'$, where given a log trace $\sigma \in \mathcal{L}$ and a threshold $t \in [0,1]$, $f_{annotate}$ returns a translucent trace σ' where $\pi_{en}(e) \in \mathcal{P}(\mathcal{A})$ for all trace events $e \in \sigma$.

4.3 Random Forest-based Approaches

We do the same thing as above, but instead of using logistic regression, we use random forests.

4.4 Transition System-based Approaches

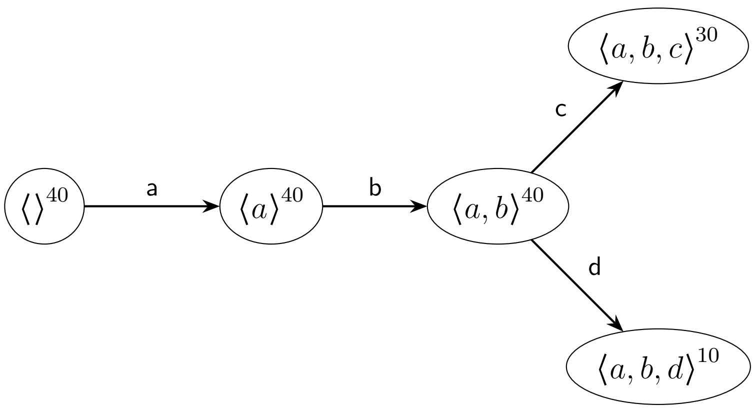
In this section, we discuss how transition systems, in particular prefix automata, can be exploited as an additional process model for the *Translucent Log Extension Problem.*

One of the advantages of utilizing transition systems over Petri nets is the lack of silent transitions. When trying to replay a trace on a Petri net in the presence of silent transition, the algorithm must decide which path of the firing sequence it should take by computing alignments with a certain cost function, then taking the path with the minimal cost. This can be computationally expensive and time-consuming, especially for larger models where the number of silent transitions is high. Transition systems can benefit us in this regard.

4.4.1 Frequency-Annotated Prefix Automata

Definition 4.3 (Frequency-Annotated Prefix Automaton)**.** Let $\mathcal{L} \subseteq \mathcal{E}$ be a simple event log. A *Frequency-Annotated Prefix Automaton (FAPA)* is a tuple $PA_{freq} = (S, A, T, f)$, where (S, A, T) is a prefix automaton $TS_{L,hd}$ following the Definition 3.16 and $f\colon S \to \mathbb{N}, \sigma_{pref} \mapsto |\{\sigma \in \mathcal{L} \mid \sigma_{pref} \sqsubseteq \sigma\}|$ is a frequency labeling function.

Let us look at a small example. Consider the simple event log $\mathcal{L} = [\langle a, b, c\rangle^{30}, \langle a, b, d\rangle^{10}]$. The resulting FAPA is depicted in 4.3. The frequency labeling function f is represented as in the usual superscript notation used in trace multiset of simple event logs.



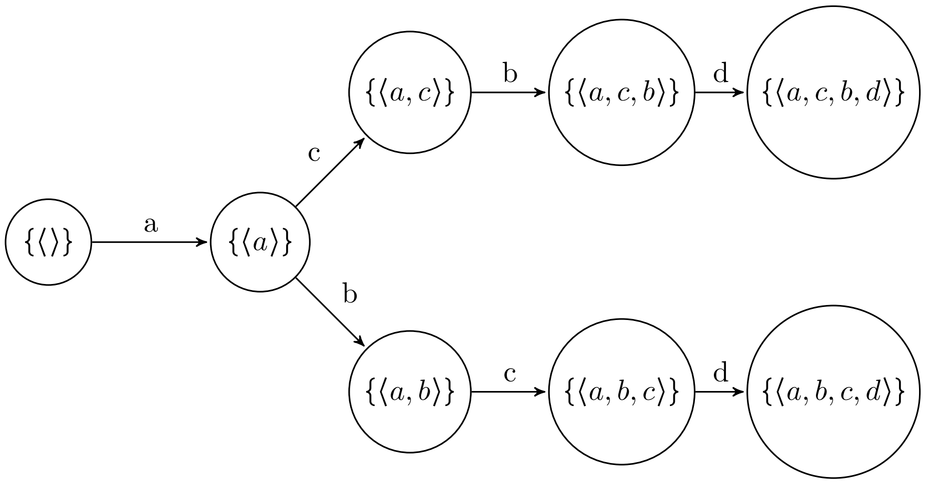
$\langle\rangle^{40}$
a
$\langle a\rangle^{40}$
b
$\langle a,b\rangle^{40}$
c
$\langle a,b,c\rangle^{30}$
d
$\langle a,b,d\rangle^{10}$

Figure 4.3: Resulting FAPA from $\mathcal{L}$.

Given a certain threshold, a simple algorithm would be to iterate over each trace in the event log and annotating the enabled activities with transitions lying above the threshold. For an example threshold value of $t = 0.5$, the resulting simple translucent event log would be: $[\langle \underline{a}, \underline{b}, \underline{c}\rangle^{30}, \langle \underline{a}, \underline{b}, c\underline{d}\rangle^{10}]$. Note that annotating enabled activities using conventional prefix automata can be considered as a special case of this problem with threshold $t = 0$.

4.4.2 Merging States

As discussed in the previous Section 3.2.2, list-based prefix automata can precisely describe the behavior of an event log, but it fails at creating a generalized process model due to its inability of recognizing loops and parallel behavior.



$\{\langle\rangle\}$
a
$\{\langle a\rangle\}$
c
b
$\{\langle a,c\rangle\}$
b
$\{\langle a,c,b\rangle\}$
d
$\{\langle a,c,b,d\rangle\}$
$\{\langle a,b\rangle\}$
c
$\{\langle a,b,c\rangle\}$
d
$\{\langle a,b,c,d\rangle\}$

Figure 4.4: Generated state-merged prefix automaton based on Figure 3.3.

In order to overcome this issue, we introduce a merge operator for the list-based prefix automata. The merge operator is defined as the following:

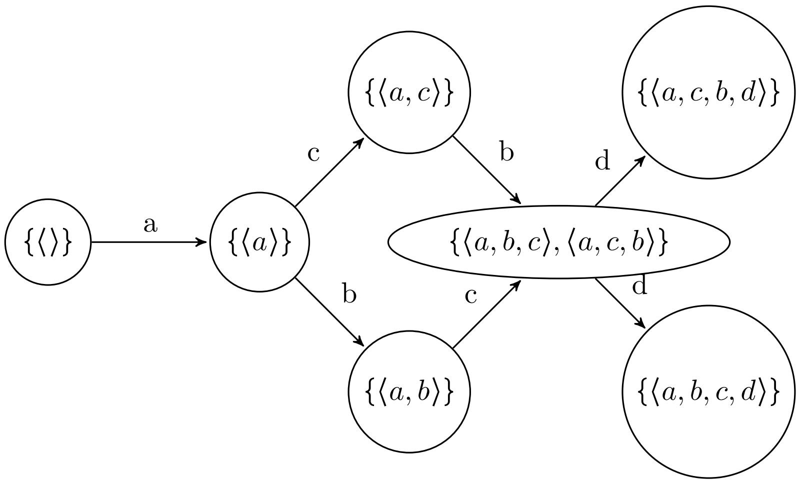
Definition 4.4 (State Merging)**.** Let $PA = (S, A, T, f)$ be a prefix automaton. A *State-merged prefix automaton* of PA is a prefix automaton $PA_{\succ} = (S_{\succ}, A_{\succ}, T_{\succ}, f_{\succ})$, where $S' \subseteq \mathcal{P}(S), A_{\succ} = A, T' \subseteq S' \times A \times S'$, and $f_{\succ} = f$.

Let $s_1, s_2 \in S_{\succ}$ be two distinct states of a state-merged prefix automaton. $PA_{\succ} \underset{s_1,s_2}{\ni\!\!-} PA'_{\succ}$ denotes that the state merge operator $\ni\!\!-$ applied on a state-merged prefix automaton $PA'_{\succ}$ together with states s_1, s_2 returns a new state-merged prefix automaton $PA'_{\succ} = (S'_{\succ}, A'_{\succ}, T'_{\succ}, f'_{\succ})$ with following properties:

- $S'_{\succ} = (S_{\succ}/\{s_1, s_2\}) \cup s'$, where s' is a new state with $s' = s_1 \cup s_2$.
- $A'_{\succ} = A_{\succ}$.
- $T'_{\succ} = (T_{\succ}/\{(s_\alpha, a, s_\beta) \mid a \in A_{\succ}, s_\alpha \in \{s_1, s_2\} \vee s_\beta \in \{s_1, s_2\}\}) \cup \{(s', a, s) \mid (s_1, a, s) \in T_{\succ} \vee (s_2, a, s) \in T_{\succ}\} \cup \{(s, a, s') \mid (s, a, s_1) \in T_{\succ} \vee (s, a, s_2) \in T_{\succ}\}$.
- $f'_{\succ} = f_{\succ}$.

Using the previous prefix automaton in Figure 3.3 as an example, we first create a state-merged prefix automaton as follows in Figure 4.4.

After merging the states $\{\langle a, c, b\rangle\}$ and $\{\langle a, b, c\rangle\}$, the resulting state-merged prefix automaton is depicted in Figure 4.5. The new state $\{\langle a, b, c\rangle, \langle a, c, b\rangle\}$ inherits the incoming and outgoing transitions of the merged states.



$\{\langle\rangle\}$
a
$\{\langle a\rangle\}$
c
b
$\{\langle a,c\rangle\}$
$\{\langle a,b\rangle\}$
b
c
$\{\langle a,b,c\rangle,\langle a,c,b\rangle\}$
d
d
$\{\langle a,c,b,d\rangle\}$
$\{\langle a,b,c,d\rangle\}$

Figure 4.5: State-merged prefix automaton after merging states $\{\langle a,c,b\rangle\}$ and $\{\langle a,b,c\rangle\}$ together.

State merging brings two advantages. Firstly, the state space can be significantly reduced depending on the operator's preference. Secondly, behaviors which were previously impossible to capture using conventional prefix automata, such as loops or parallel executions, can now be represented in the state-merged prefix automaton.

After state merging, we can utilize the resulting automaton for multivariate regression in the identical manner as described in the Petri net-based approach in section 4.2.

4.5 Transformer-based Approaches

Buttom-up approaches aims to discover enabled activities without the help of a process model. ...

A simple approach is to utilize the *directly-follows matrix* of an event log. Given an event log $\mathcal{L}$, we compute the set of unique activities and generate a next-activity matrix $\mathbf{A}$, where the entry $\mathbf{A}_{ij}$ represents the number of times activity j follows activity i. We can then easily transform $\mathbf{A}$ into a probability matrix $\mathbf{A}^{'}$ by dividing each row by the sum of the row. We then filter out the results by a certain threshold p and add the entries surviving the threshold to each event trace. This algorithm can serve as baseline for further extensions. We formally define the directly-follows matrix as follows:

Definition 4.5 (Directly-follows Matrix)**.** Let $\mathcal{L}$ be an event log. Let further $U = \pi_{act}(L) = \bigcup\limits_{\sigma\in\mathcal{L}} \{\pi_{act}(e) \mid \pi_{act}(e) \in \sigma\}$ be the set of unique activities in $\mathcal{L}$. $\mathbf{A}$ is a *directly-follows matrix* of $\mathcal{L}$, if:

- $\mathbf{A} \in \mathbb{N}^{n\times n}$, where $n = |U|$, i.e., the number of unique activities in $\mathcal{L}$.
- Let $\{U_j\}_{j\in J}$ be an indexed family of U with $J = \{1,2,\dots,n\}, n = |U|$. $\mathbf{A}_{ij} = \sum\limits_{\sigma\in\mathcal{L}} \Big|\{i \mid 1 \le i \le |\sigma|, \sigma(i) = U_i \wedge \sigma(i+1) = U_j\}\Big|$, i.e., the absolute frequency activity U_i is directly followed by activity U_j.

Given the directly-follows matrix as a baseline, there are several applications to extend the matrix. One of the most straightforward approaches is transforming the matrix into a Markov matrix.

Definition 4.6 (Probabilistic Directly-follows Matrix)**.** Let $\mathbf{A} = (\mathbf{a_{ij}})$ be a directly-follows matrix of an event log $\mathcal{L}$. $\mathbf{A}' = (a'_{ij})$ is a *probabilistic directly-follows matrix* of $\mathcal{L}$ if

$$a'_{ij} = \frac{a_{ij}}{\sum\limits_{k=1}^{n} a_{ik}}.$$

- Furthermore, we can utilitze a deep-learning based black-box approach. The issue with implementing supervised learning algorithms is that we need a labeled training dataset. In our case, this would be a preexisting translucent event log, which is unavailable in our setting due to missing enabled activities data. We can cirucmvent the problem by training the model using the next activity information as label, as this information is available in every event log. Since most learning algorithms would not return a single value but an underlying probability distribution of possible outcomes, we can substitute the final *argmax* operation with selecting a threshold p and returning all labels lying above it.

Chapter 5

Implementation

The program accepts an event log and an optional process model, e.g. a Petri net, as inputs and returns a corresponding translucent event log as output. Users have the option to select from various methods of log generation, will be specified below. Mainly, these methods can be classified in two categories: top-down approaches which require a Petri net, and bottom-up approaches which solely need the event log.

In order to provide a user-friendly interface, we implement a web application. The following sections describe the software specification, architecture, and its features. The comprehensive source code is publically available on GitHub[1][2].

5.1 Software Architecture

In the following sections, we briefly describe the technical architecture used for *Translucify*. The architecture can be partitioned into two main components, the backend and the frontend. The backend houses the actual algorithms used to extend the event log with enabled activities, while the frontend provides a UI for user interaction.

5.1.1 Backend

The backend was implemented with Python using the Flask[3] framework. Flask is fitting for our use case as it is a lightweight framework providing a minimalistic set of features needed to build API calls.

Behind the API handlers, the backend logic is dependent on `PM4Py` [21], a novel, rapidly established process mining library written for the Python programming language. Other than `PM4Py`, standard libraries necessary for machine learning and data processing, such as `NumPy` [22], `Pandas` [23], and `Scikit-learn` [24] were used. The choice of using Python for building the backend was an intentional decision in order to incorporate these libraries natively into the backend.

For the database, SQLAlechemy[4] was used as an object relational mapping (ORM) tool

[1]https://github.com/biscimus/Translucify

[2]https://github.com/biscimus/Translucify-Transformer-Service

[3]www.flask.palletsprojects.com/en/3.0.x/

[4]www.sqlalchemy.org

to model the database. It was chosen due to its seamless compatibility with Flask using the Flask-SQLAlchemy extension and its ease of usage.

The database stores event log entities and their corresponding translucent event log entities. Since the event logs tend to be large, the actual event logs are stored in the file system, while the database only keeps track of the metadata, such as the file path, name, and type of the event logs. Translucent event log entities have an extra attribute `is_ready` in order to indicate whether the computation is complete. The database schema is depicted in Figure 5.1.

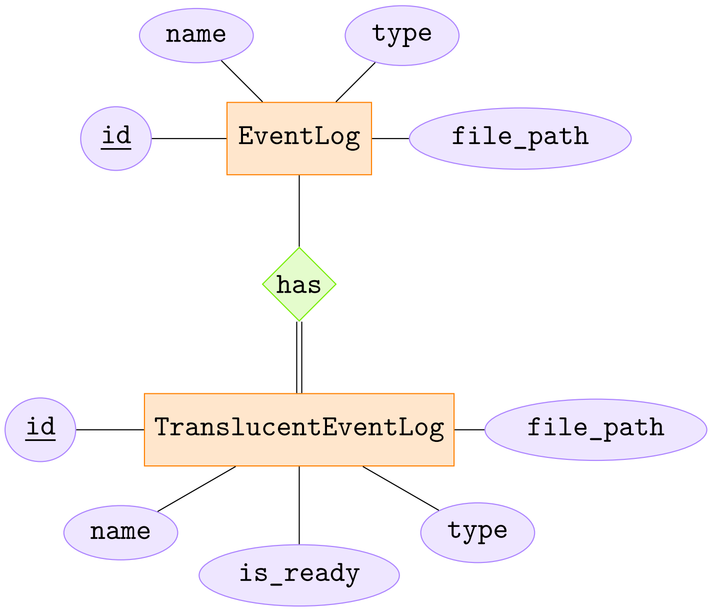


Figure 5.1: Entity-relationship diagram of the database schema.

5.1.2 Frontend

The frontend is a web application built with React [25]. The application leverages Mantine [5] as its UI library, TanStack Router [6] for routing and TanStack Query [7] for asynchronous state management. The frontend application is responsible for sending & receiving requests & responses with the backend, parsing & rendering the data, and finally to provide a user-friendly interface for the user to interact with the application.

The web application will start with an interface where the user can upload an event log, with a choice between CSV and XES. After uploading the event log, different methods of translucent log annotation will be available. The user can select the method and the corresponding necessary parameters. After the log generation is complete, the user can download the log either in a CSV or an XES format.

[5]https://mantine.dev/
[6]https://tanstack.com/router/latest
[7]https://tanstack.com/query/latest

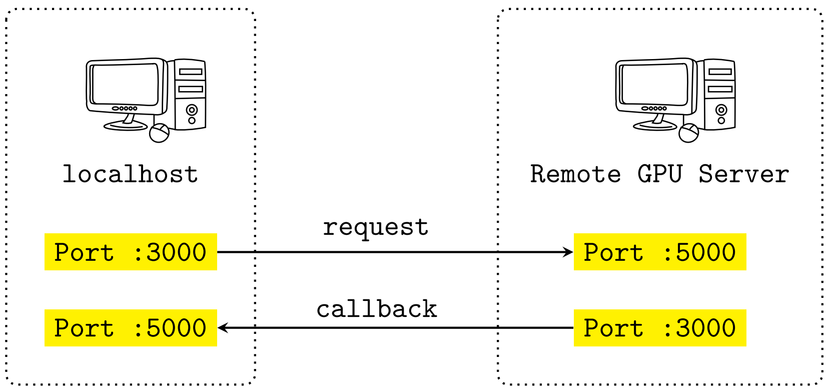


Figure 5.2: SSH Port Forwarding.

5.2 Software Features

The following section outlines some technical features employed to implement the software architecture. It also describes the sequence flows within application between the frontend and the backend.

5.2.1 GPU Computing

For the transformer variant, in order to train the transformer model, a GPU access was necessary due to the high computational requirements. In a local setting, the program would take an extraordinary amount of computing time, or in other cases simply terminate due to insufficient memory.

In order to solve this issue, a connection to the remote PADS HPC Cluster needed to be established. Inside the cluster, a Flask microservice was set up analogously to the local setup. Using SSH tunneling, all requests regarding the transformer model were then forwarded to the remote server. After the computations are finished, the results from the remote server were subsequently delivered back to the main application running in the local server. Figure 5.2 illustrates the SSH port forwarding setup.

A screenshot of the remote service is available in Figure 5.3.

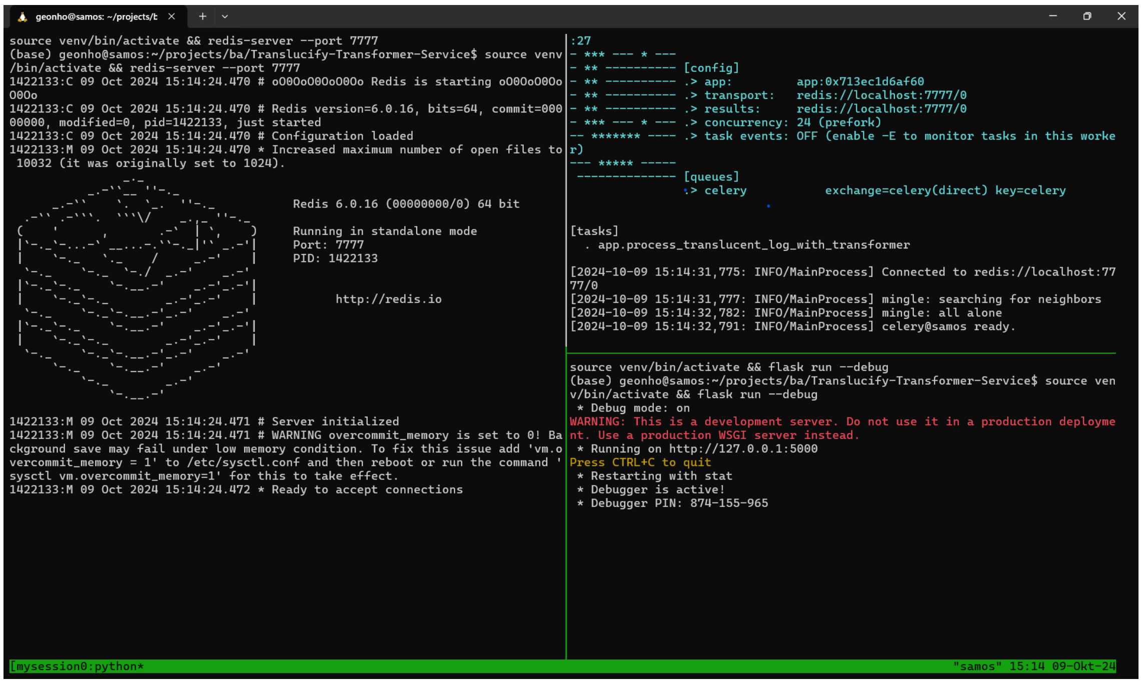
5.2.2 Asynchronous Task Distribution

All requests from the frontend are processed by request handlers of Flask. A particular issue we encountered is this case was the excessive computation time for each algorithm run in the backend. As a result, when attempting to run the algorithms directly on the request handlers in Flask, the server would block all other requests until the current computation was finished, making the application unresponsive and bluntly unusable. To mitigate this issue, we incorporated a distributed task queue using Redis[8] and Celery[9].

Celery is a distributed system to split up computation in tasks and to assign these to subprocesses, known as workers. Redis is used as a *message broker* so that the Celery client can delegate the tasks to the workers. Upon receiving a request, the Flask server

[8]https://redis.io/

[9]https://pypi.org/project/celery/



```
geonho@samos: ~/projects/b
```

```
source venv/bin/activate && redis-server --port 7777
(base) geonho@samos:~/projects/ba/Translucify-Transformer-Service$ source venv
/bin/activate && redis-server --port 7777
1422133:C 09 Oct 2024 15:14:24.470 # oO0OoO0OoO0Oo Redis is starting oO0OoO0Oo
O0Oo
1422133:C 09 Oct 2024 15:14:24.470 # Redis version=6.0.16, bits=64, commit=000
00000, modified=0, pid=1422133, just started
1422133:C 09 Oct 2024 15:14:24.470 # Configuration loaded
1422133:M 09 Oct 2024 15:14:24.470 * Increased maximum number of open files to
 10032 (it was originally set to 1024).
                _._
           _.-``__ ''-._
      _.-``    `.  `_.  ''-._           Redis 6.0.16 (00000000/0) 64 bit
  .-`` .-```.  ```\/    _.,_ ''-._
 (    '      ,       .-`  | `,    )     Running in standalone mode
 |`-._`-...-` __...-.``-._|'` _.-'|     Port: 7777
 |    `-._   `._    /     _.-'    |     PID: 1422133
  `-._    `-._  `-./  _.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |           http://redis.io
  `-._    `-._`-.__.-'_.-'    _.-'
 |`-._`-._    `-.__.-'    _.-'_.-'|
 |    `-._`-._        _.-'_.-'    |
  `-._    `-._`-.__.-'_.-'    _.-'
      `-._    `-.__.-'    _.-'
          `-._        _.-'
              `-.__.-'

1422133:M 09 Oct 2024 15:14:24.471 # Server initialized
1422133:M 09 Oct 2024 15:14:24.471 # WARNING overcommit_memory is set to 0! Ba
ckground save may fail under low memory condition. To fix this issue add 'vm.o
vercommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command '
sysctl vm.overcommit_memory=1' for this to take effect.
1422133:M 09 Oct 2024 15:14:24.472 * Ready to accept connections
```

```
:27
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         app:0x713ec1d6af60
- ** ---------- .> transport:   redis://localhost:7777/0
- ** ---------- .> results:     redis://localhost:7777/0
- *** --- * --- .> concurrency: 24 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worke
r)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . app.process_translucent_log_with_transformer

[2024-10-09 15:14:31,775: INFO/MainProcess] Connected to redis://localhost:77
77/0
[2024-10-09 15:14:31,777: INFO/MainProcess] mingle: searching for neighbors
[2024-10-09 15:14:32,782: INFO/MainProcess] mingle: all alone
[2024-10-09 15:14:32,791: INFO/MainProcess] celery@samos ready.
```

```
source venv/bin/activate && flask run --debug
(base) geonho@samos:~/projects/ba/Translucify-Transformer-Service$ source ven
v/bin/activate && flask run --debug
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployme
nt. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 874-155-965
```

```
[mysession0:python*                                        "samos" 15:14 09-Okt-24
```

Figure 5.3: Screenshot of the remote service running on the HPC Cluster. In each of the three separate tmux terminals, the Redis server (left), the Celery worker (top right), and the Flask server (bottom right) are running.

will pass on the task to the Celery client, which will then push the task to a Redis Message Queue. The Celery workers will then pick up the task from the queue and execute it. Compared to the setup where all tasks are handled directly in the request handlers, this setup allows for a non-blocking behavior of the server, as the tasks are executed in the background by other processes, thereby preventing the application from freezing.

The overall application architecture is depicted in Figure 5.4.

5.2.3 Multivariable Regression With Petri Nets

After the user selection, the client sends a request to the server to obtain the columns of the event log. After receiving the list of columns, the user subsequently selects a set of data columns to be used for the multivariable regression, while simultaneously indicating whether the columns are numerical or categorical. After the selection is complete, the client sends a request to the server to initiate the computation with corresponding columns. After the computation is terminated, the backend flips the `is_ready` boolean attribute of the translucent event log entity.

The complete sequence is depicted in Figure 5.5.

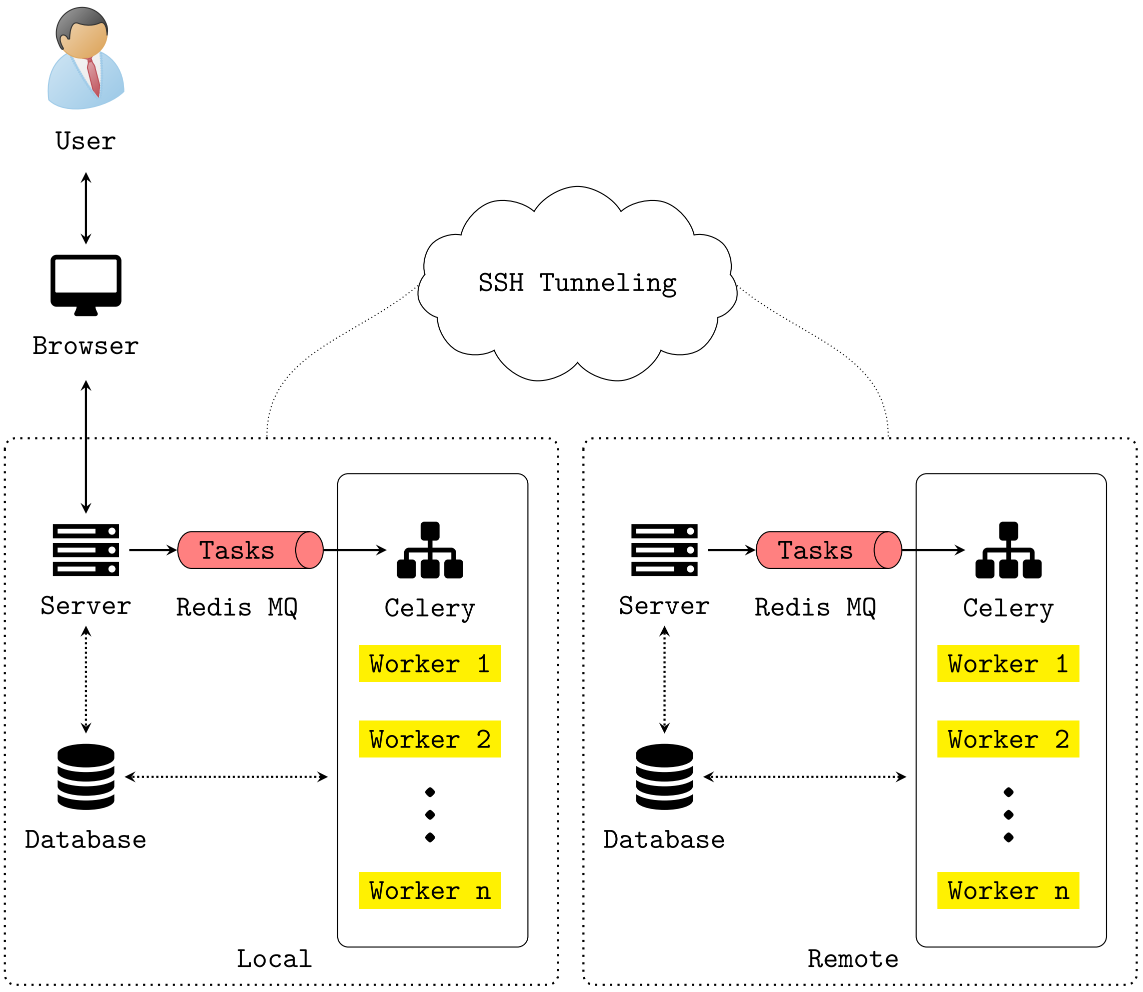


Figure 5.4: Application Architecture

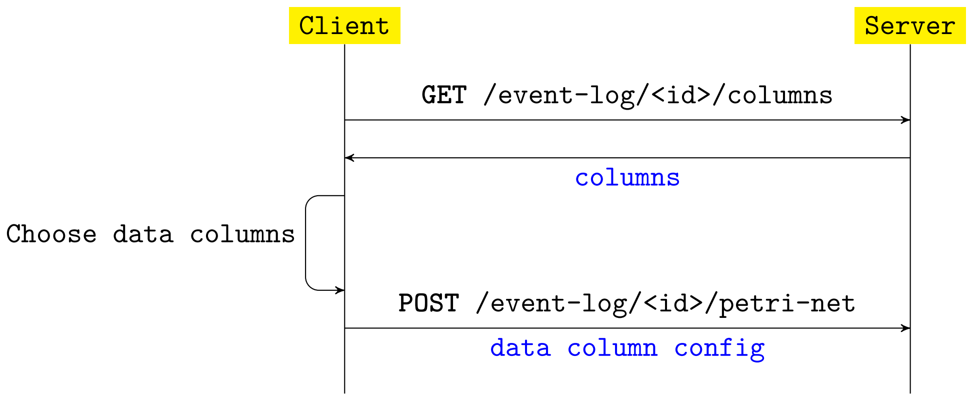


Figure 5.5: Client-server sequence diagram for the Petri net variant.

5.2.4 Multivariable Regression with Prefix Automata

The user sequence for multivariable regression with prefix automata is analogous to the Petri net variant for the most part. An additional step, however, is required in order to fine-tune the prefix automaton. After the data column selection step, the server computes a prefix automaton. The user then is able to fine-tune the prefix automaton by merging states. After the user is satisfied with his custom fine-tuned automaton, the client sends a request to the server to initiate the regression computation.

The complete sequence is depicted in Figure 5.6.

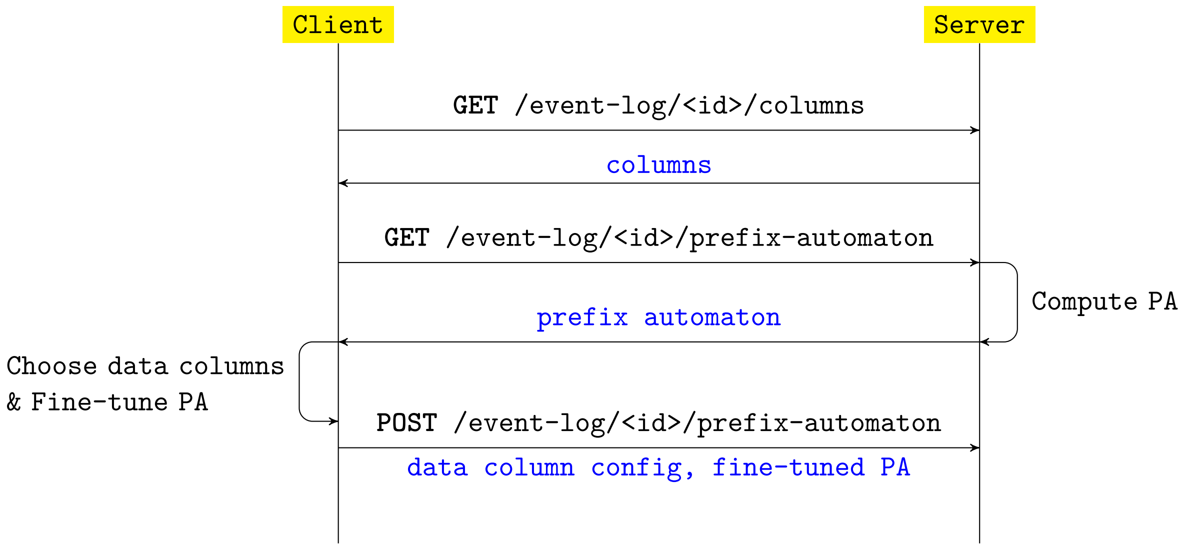

Client
Server
GET /event-log/<id>/columns
columns
GET /event-log/<id>/prefix-automaton
Compute PA
prefix automaton
Choose data columns
& Fine-tune PA
POST /event-log/<id>/prefix-automaton
data column config, fine-tuned PA

Figure 5.6: Client-server sequence diagram for the Prefix automaton variant.

5.2.5 Random Forests

The Scikit-learn library was used to implement the random forest model. Unlike the classical random forest, the random forest model in Scikit-learn uses decision trees which returns a probability vector, in which each element represents the probability of each activity class. The forest then averages the probability vectors of all trees to formulate a prediction instead of a simple majority vote.

5.3 User Workflow

In the upcoming section, we will describe the user workflow of *Translucify* with the help of screenshots from the application.

5.3.1 Uploading an Event Log

Upon opening the application, the user faces the simple landing page as in Figure 5.7. After pressing the `Go to your logs` button, the user will be navigated to the home page, where an overview of previously uploaded event logs are provided. The list of all uploaded event logs is constantly available as a sidebar on the left side. Furthermore, an upload button is located above the event log table, allowing the user to upload a fresh event log. The home page is depicted in Figure 5.8.

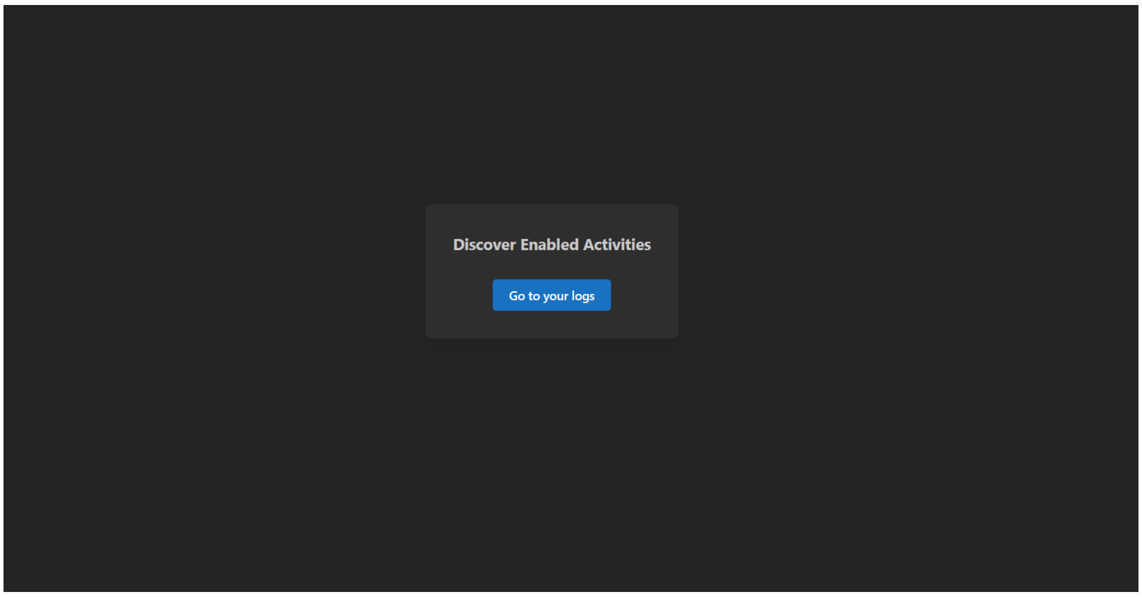
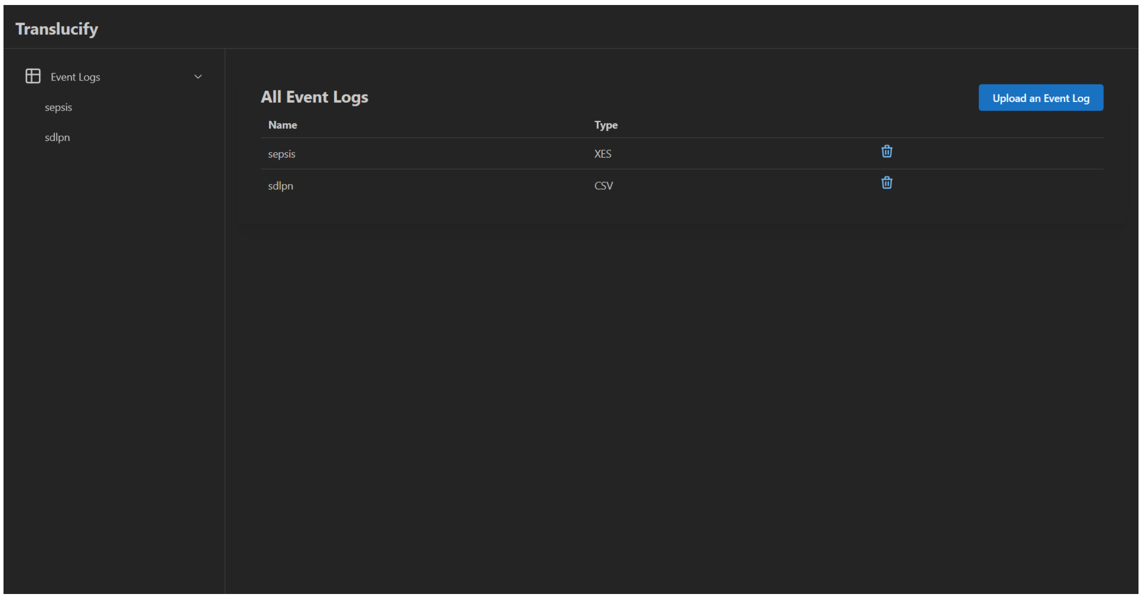


Figure 5.7: Landing page of *Translucify*.


Translucify

Event Logs
sepsis
sdlpn

All Event Logs

Upload an Event Log

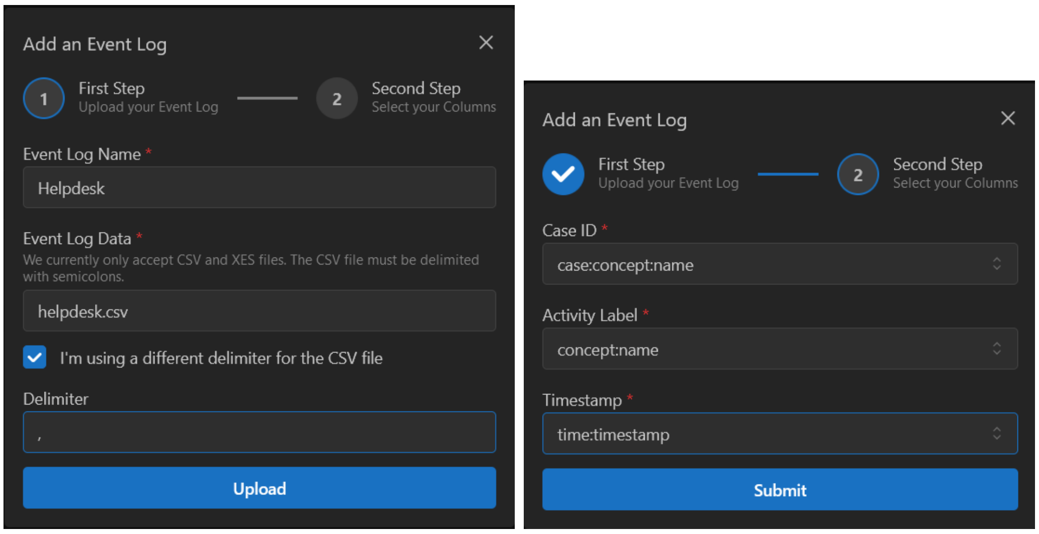
Name	Type
sepsis	XES
sdlpn	CSV

Figure 5.8: Home page of *Translucify*.

Pressing the `Upload an Event Log` button opens a modal used for the event log upload.

The modal wizard consists of two steps. In the first step, the user is prompted to upload the event log data, which can either be in a CSV or an XES format. A name is also a required field for the event log. Although *Translucify* assumes the CSV delimiter to be a semicolon, the user can still specify a custom delimiter if necessary.

The wizard moves onto the second step after clicking on the `Upload` button. In the background, the frontend sends a request to the backend in order to save the event log in the database, upon which the backend reads the event log data and returns the column list of the event log. Here, the user can manually select the three fundamental columns for the event log: case ID, activity, and timestamp. After pressing the `Submit` button, the user will be subsequently prompted to the event log dashboard of the newly uploaded event log. Figure 5.9 illustrates the event log upload modal.



Event Log Name *
Helpdesk

Event Log Data *
We currently only accept CSV and XES files. The CSV file must be delimited with semicolons.
helpdesk.csv

☑ I'm using a different delimiter for the CSV file

Delimiter
,

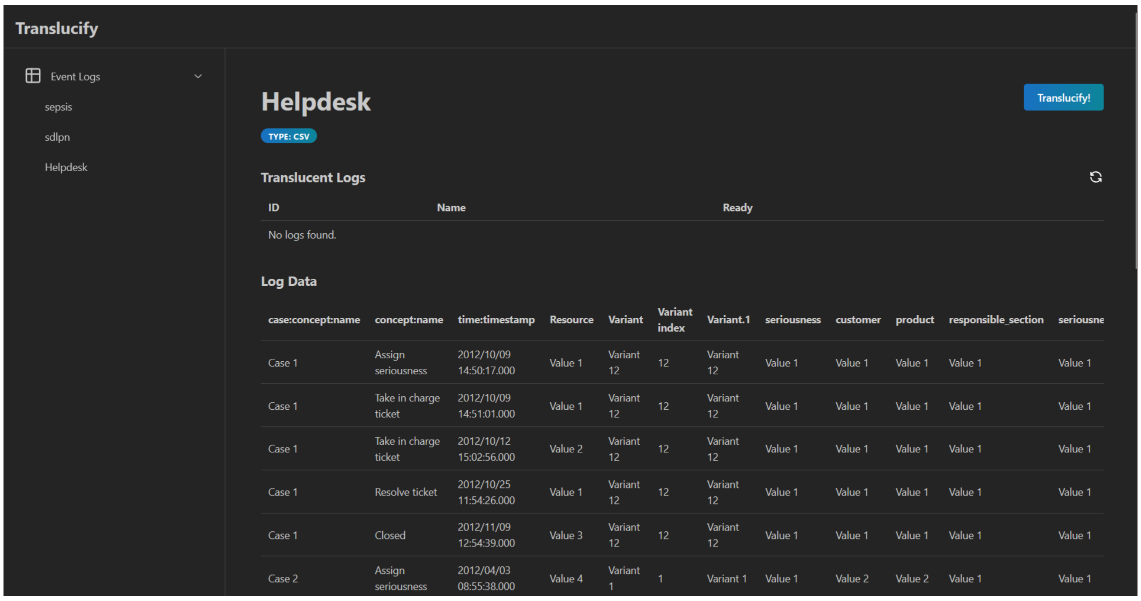
Case ID *
case:concept:name

Activity Label *
concept:name

Timestamp *
time:timestamp

Figure 5.9: Event log upload modal consisting of a two-step wizard.

5.3.2 Event Log Dashboard



Helpdesk

TYPE: CSV

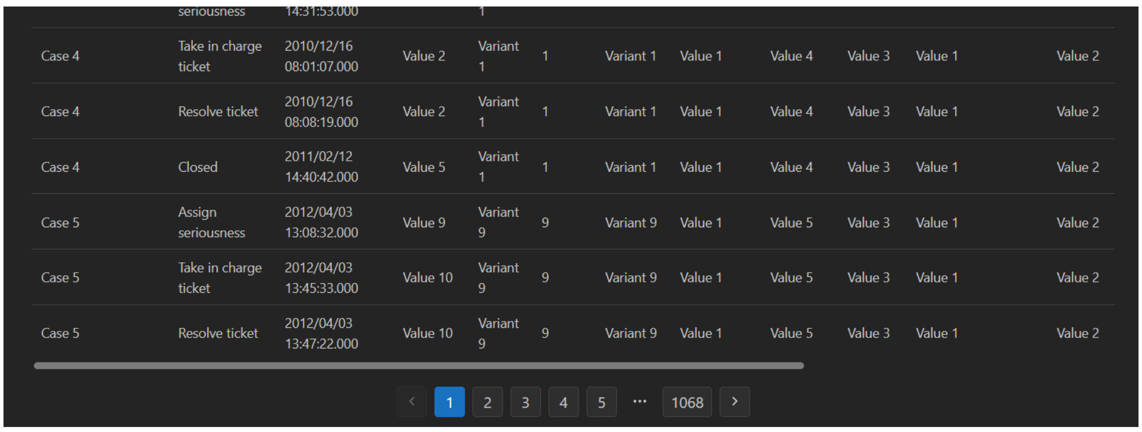
Translucent Logs

ID	Name	Ready
No logs found.		

Log Data

case:concept:name	concept:name	time:timestamp	Resource	Variant	Variant index	Variant.1	seriousness	customer	product	responsible_section	seriousne
Case 1	Assign seriousness	2012/10/09 14:50:17.000	Value 1	Variant 12	12	Variant 12	Value 1	Value 1	Value 1	Value 1	Value 1
Case 1	Take in charge ticket	2012/10/09 14:51:01.000	Value 1	Variant 12	12	Variant 12	Value 1	Value 1	Value 1	Value 1	Value 1
Case 1	Take in charge ticket	2012/10/12 15:02:56.000	Value 2	Variant 12	12	Variant 12	Value 1	Value 1	Value 1	Value 1	Value 1
Case 1	Resolve ticket	2012/10/25 11:54:26.000	Value 1	Variant 12	12	Variant 12	Value 1	Value 1	Value 1	Value 1	Value 1
Case 1	Closed	2012/11/09 12:54:39.000	Value 3	Variant 12	12	Variant 12	Value 1	Value 1	Value 1	Value 1	Value 1
Case 2	Assign seriousness	2012/04/03 08:55:38.000	Value 4	Variant 1	1	Variant 1	Value 1	Value 2	Value 2	Value 1	Value 1

Figure 5.10: Event log dashboard.



	seriousness	14:31:53.000		1							
Case 4	Take in charge ticket	2010/12/16 08:01:07.000	Value 2	Variant 1	1	Variant 1	Value 1	Value 4	Value 3	Value 1	Value 2
Case 4	Resolve ticket	2010/12/16 08:08:19.000	Value 2	Variant 1	1	Variant 1	Value 1	Value 4	Value 3	Value 1	Value 2
Case 4	Closed	2011/02/12 14:40:42.000	Value 5	Variant 1	1	Variant 1	Value 1	Value 4	Value 3	Value 1	Value 2
Case 5	Assign seriousness	2012/04/03 13:08:32.000	Value 9	Variant 9	9	Variant 9	Value 1	Value 5	Value 3	Value 1	Value 2
Case 5	Take in charge ticket	2012/04/03 13:45:33.000	Value 10	Variant 9	9	Variant 9	Value 1	Value 5	Value 3	Value 1	Value 2
Case 5	Resolve ticket	2012/04/03 13:47:22.000	Value 10	Variant 9	9	Variant 9	Value 1	Value 5	Value 3	Value 1	Value 2

Figure 5.11: Horizontal scroll and pagination feature for the raw log data.

The event log dashboard provides the comprehensive information regarding the specific event log instance and consists of two main sections.

The first section is the translucent log table, in which every generated and to-be-generated translucent event log is registered, listing information such as the generation method and completion status. The download feature enabled the user to download the translucent event log data directly from the table.

The second section is the event log table, which displays the raw event log data in form of a table. During development, we observed frequent browser crashes due to the extensive data size in the table when testing with large event logs. In order to mitigate this, a pagination feature was added to the table. Moreover, the table is horizontally scrollable for event logs with a large column count. Figures 5.10 and 5.11 demonstrate the event log dashboard and the pagination feature, respectively.

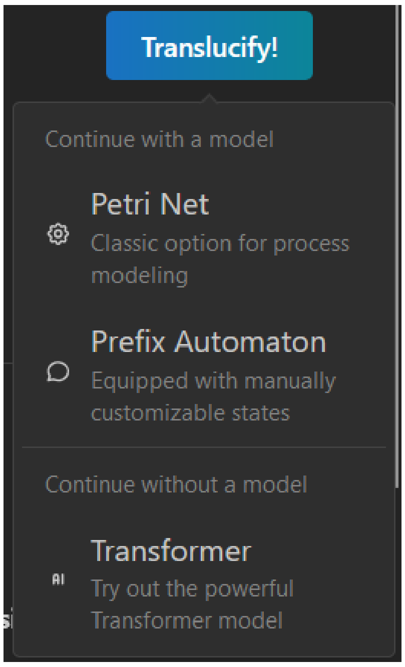
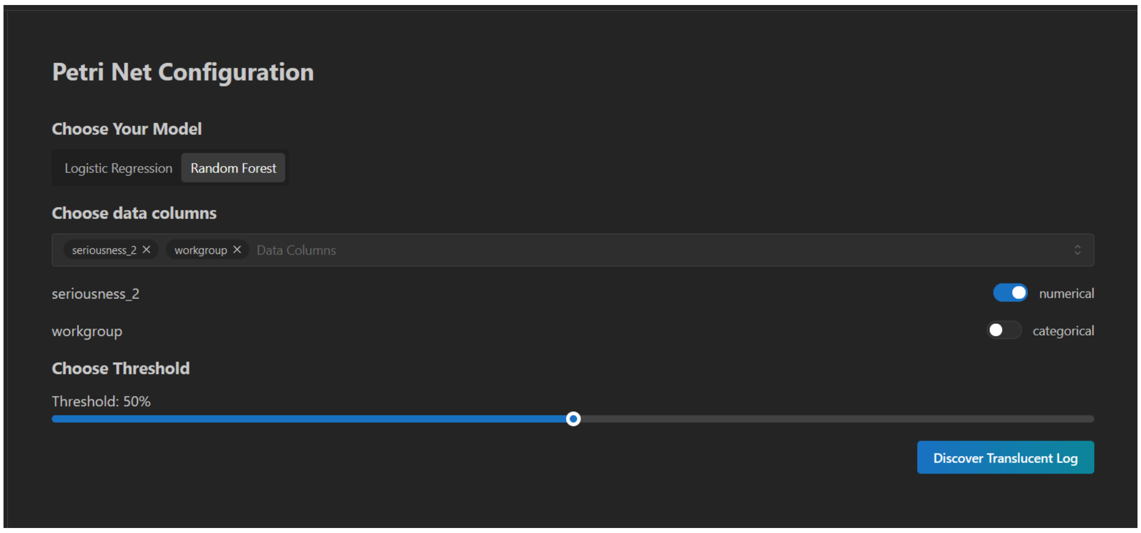

Translucify!
Continue with a model
Petri Net
Classic option for process modeling
Prefix Automaton
Equipped with manually customizable states
Continue without a model
AI
Transformer
Try out the powerful Transformer model

Figure 5.12: Choice popup.

Upon clicking the `Translucify!` button, the user needs to select between three log generation options: Petri net, prefix automaton, and transformer. Pressing each button navigates the user to the corresponding configuration page. The popup is depicted in Figure 5.12.

5.3.3 Configuration Pages

Petri net and prefix automaton configuration pages share an identical component. The user is first prompted to choose a classifier model between random forests and multivariable regression. Next, the user can select the feature columns to be used for data analysis and subsequently indicate whether the columns are numerical or categorical. Lastly, the user can select the probability threshold for the enabled activities.



Petri Net Configuration

Choose Your Model

Logistic Regression [✓ Random Forest]

Choose data columns

seriousness_2 ✕ workgroup ✕ Data Columns

seriousness_2	numerical
workgroup	categorical

Choose Threshold

Threshold: 50%

Discover Translucent Log

Figure 5.13: Petri net configuration page. The random forest model is selected, along with a numerical data column `seriousness_2` and a categorical data column `workgroup`. The probability threshold is set to 0.5.

A unique component for the prefix automaton configuration page is the graphical prefix automaton interface, allowing the user to merge states.

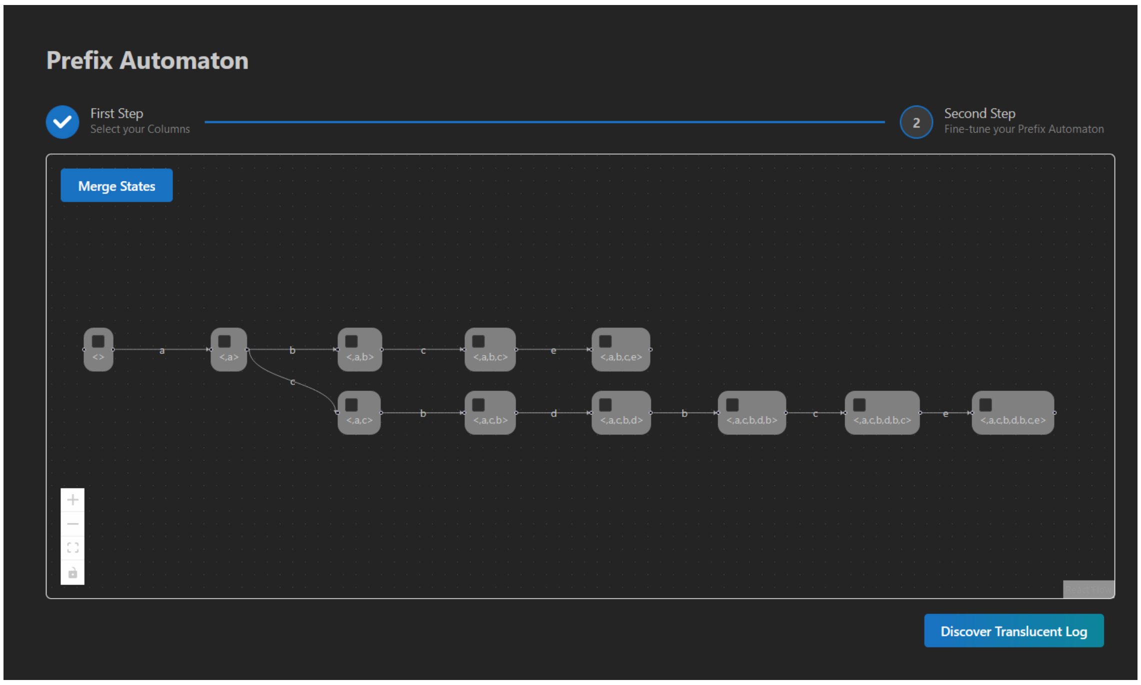


Figure 5.14: Prefix automaton configuration page.

Chapter 6

Evaluation

- Limitations: Using real-life translucent event logs is often implausible for our scenario due to the fact that most real-life logs do not contain the set of enabled activities. Therefore, direct evaluation by receiving a real-life translucent event log as input, stripping away the enabled activities column, inserting the log in our algorithm as input then comparing the result with the original translucent event log is not possible.
- Instead, we can use artificial process models. The evaluation process works like the following:
    1. We generate random process models, e.g. data Petri nets.
    2. We then play-out the model randomly and extract 1. a normal log and 2. a translucent event log.
    3. We then use the normal log as input to our TLG program and compare the result with the original translucent log.
- On top of that, we can also evaluate its versatility by directly comparing models generated using translucent event logs and the usual state-of-the-art process discovery algorithms. The evaluation process works like the following:
    1. Given a normal process log, we use state-of-the-art process discovery algorithms to generate process models.
    2. We use the log as input to our program and generate a translucent event log.
    3. We then generate a process model based on translucent-log based process discovery algorithms.
    4. We then compare the models based on their performance measures.
- In the model annotation setting, we can evaluate the performance of the model extension algorithm by comparing the stochastic precision of the annotated model with the original model. The evaluation process works like the following:
    1. We iterate over each trace and replay it on two models: The original model received as input and the annotated model.

2. For each transition, we calculate the stochastic precision by computing the product of the transition probabilities in each transition step.
3. We then add up the stochastic precision score for each trace and divide it by the total number of traces to get the average stochastic precision score.

Note that this is different from the translucent precision score defined in [12], since we need a precision measure comparable and applicable to both of the original log-model-pair and the translucent-log-annotated-model pair.

- As we are not presenting a single generation method, it will be necessary to compare and evaluate each method separately using the process described above.

6.1 Some another section

6.1.1 Transformer Models

The transformer model is implemented using the PyTorch library.

The model is trained on the PADS HPC Cluster using an NVIDIA GeForce RTX 2080 Ti GPU with a total memory of 11264 MiB equipped with CUDA 12.2. Using a learning rate of 0.00001 and a batch size of 16, the model is trained for 50 epochs.

Chapter 7

Discussion

Chapter 8

Conclusion

Bibliography

[1] Wil M. P. van der Aalst. *Data Science in Action*, pages 3–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. ISBN 978-3-662-49851-4. doi: 10.1007/978-3-662-49851-4_1.

[2] Josep Carmona, Boudewijn van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking: Relating Processes and Models*. Springer, Germany, November 2018. ISBN 978-3-319-99413-0. doi: 10.1007/978-3-319-99414-7.

[3] Anne Rozinat and Wil M. P. van der Aalst. Decision mining in ProM. In *Business Process Management*, pages 420–425, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-38903-3.

[4] Anne Rozinat and Wil M. P. van der Aalst. *Decision mining in business processes*. BETA publicatie : working papers. Technische Universiteit Eindhoven, 2006. ISBN 90-386-0685-0.

[5] Massimiliano de Leoni and Wil M. P. van der Aalst. Data-aware process mining: discovering decisions in processes using alignments. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, page 1454–1461, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450316569. doi: 10.1145/2480362.2480633.

[6] Felix Mannhardt, Sander J. J. Leemans, Christopher T. Schwanen, and Massimiliano de Leoni. Modelling data-aware stochastic processes - discovery and conformance checking. In Luis Gomes and Robert Lorenz, editors, *Application and Theory of Petri Nets and Concurrency*, pages 77–98, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-33620-1.

[7] Wil M. P. van der Aalst. Markings in perpetual free-choice nets are fully characterized by their enabled transitions. In *Application and Theory of Petri Nets and Concurrency: 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, June 24-29, 2018, Proceedings*, page 315–336, Berlin, Heidelberg, 2018. Springer-Verlag. ISBN 978-3-319-91267-7. doi: 10.1007/978-3-319-91268-4_16.

[8] Wil M. P. van der Aalst, Victor Khomenko, Jetty Kleijn, Wojciech Penczek, and Olivier H. Roux. Lucent process models and translucent event logs. *Fundamenta Informaticae*, 169(1–2):151–177, jan 2019. ISSN 0169-2968. doi: 10.3233/FI-2019-1842.

[9] Harry H. Beyel and Wil M. P. van der Aalst. Creating translucent event logs to improve process discovery. In Marco Montali, Arik Senderovich, and Matthias Weidlich,

editors, *Process Mining Workshops*, pages 435–447, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-27815-0.

[10] Harry H. Beyel and Wil M. P. van der Aalst. Improving process discovery using translucent activity relationships. In Andrea Marrella, Manuel Resinas, Mieke Jans, and Michael Rosemann, editors, *Business Process Management*, pages 146–163, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-70396-6.

[11] Sander J. J. Leemans. *Robust process mining with guarantees.* PhD thesis, Eindhoven University of Technology, 2017.

[12] Harry H. Beyel and Wil M. P. van der Aalst. Translucent precision: Exploiting enabling information to evaluate the quality of process models. In João Araújo, Jose Luis de la Vara, Maribel Yasmina Santos, and Saïd Assar, editors, *Research Challenges in Information Science*, pages 29–37, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-59468-7.

[13] Li Lin, Lijie Wen, and Jianmin Wang. *MM-Pred: A Deep Predictive Model for Multi-attribute Event Sequence*, pages 118–126. 2019. doi: 10.1137/1.9781611975673.14.

[14] Björn R. Gunnarsson, Seppe vanden Broucke, and Jochen De Weerdt. A direct data aware lstm neural network architecture for complete remaining trace and runtime prediction. *IEEE Transactions on Services Computing*, 16(04):2330–2342, jul 2023. ISSN 1939-1374. doi: 10.1109/TSC.2023.3245726.

[15] Zaharah A. Bukhsh, Aaqib Saeed, and Remco M. Dijkman. Processtransformer: Predictive business process monitoring with transformer network, 2021.

[16] Chiara Di Francescomarino and Chiara Ghidini. *Predictive Process Monitoring*, pages 320–346. Springer International Publishing, Cham, 2022. ISBN 978-3-031-08848-3. doi: 10.1007/978-3-031-08848-3_10.

[17] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324.

[18] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995. doi: 10.1109/ICDAR.1995.598994.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[20] Anne Rozinat and Wil M. P. van der Aalst. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In Christoph J. Bussler and Armin Haller, editors, *Business Process Management Workshops*, pages 163–176, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-32596-3.

[21] Alessandro Berti, Sebastiaan van Zelst, and Daniel Schuster. Pm4py: A process mining library for python. *Software Impacts*, 17:100556, 2023. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2023.100556.

[22] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with numpy. *Nature*, 585(7825):357–362, Sep 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2.

[23] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[25] Eric Baer. *What React Is and Why It Matters*. O'Reilly Media, Inc., 2018. ISBN 9781491996737.

Appendices

Appendix A

Some Appendix

Try to avoid appendices.

Acknowledgments

At first, I would like to express my gratitude to the awesome supervisor that gave me this template.