UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Zhaosi Qu

Back-end of Kairos:
A Prescriptive Process Monitoring Tool

Master's Thesis (30 ECTS)

Supervisor(s): Fredrik Milani, PhD
Mahmoud Shoush, MSc
Kateryna Kubrak, MA

Tartu 2023

Back-end of Kairos: A Prescriptive Process Monitoring Tool

Abstract:
Prescriptive process monitoring is an approach that aims to predict potential failures and provide recommendations to optimize business processes. It seeks to improve efficiency and productivity by aiding enterprises in making informed decisions during process execution. For example, it can be applied to optimize a company's supply chain management by predicting delays and suggesting actions based on historical data. The primary problem that this thesis address is the absence of a comprehensive tool capable of analyzing data from different sources and offering various types of prescriptive recommendations. Consequently, the objective of this study is to propose and implement a software solution that enables the integration of diverse algorithms and plugins in a seamless manner. The proposed approach includes back-end software that provides APIs to implement prescriptive process monitoring features. Users can upload event logs to the tool and receive various prescriptions for ongoing cases, encompassing predictions of the next activities, scoring the likelihood of adverse outcomes, providing treatment effects, and allocating resources based on treatment gains. Moreover, the modular design enhances adaptability and flexibility across various business domains. To evaluate the effectiveness of the proposed solution, a combination of requirements fulfillment evaluation and performance evaluation is conducted using datasets from the Business Process Intelligence Challenge (BPIC). As a result, this thesis contributes to the field by providing a prescriptive process monitoring tool that can provide multiple types of prescriptive recommendations.

Keywords:
Prescriptive process monitoring, Process mining, Process optimization

CERCS:
P170 Computer science, numerical analysis, systems, control

Kairose tagumine osa: retseptipõhine protsessi jälgimise tööriist

Lühikokkuvõte:
Preskriptiivne protsessimonitooring on uus lähenemine, mille eesmärk on ennustada võimalikke rikkeid ja anda soovitusi äriprotsesside optimeerimiseks. See püüab parandada tõhusust ja tootlikkust, aidates ettevõtetel teha informeeritud otsuseid protsesside täitmisel. Näiteks saab seda rakendada ettevõtte tarneahela haldamise optimeerimiseks, ennustades viivitusi ja soovitades tegevusi ajalooliste andmete põhjal. Selle väitekirja peamine probleem on kattuva tööriista puudumine erinevate allikate andmete analüüsimisel ning erinevat tüüpi preskriptiivsete soovituste pakkumisel. Seetõttu on selle uurimistöö eesmärk välja pakkuda ja rakendada tarkvaralahendus, mis võimaldaks mitmekesiste

algoritmide ja pistikprogrammide integreerimist sujuvalt. Pakutav lähenemisviis sisaldab tagasipoolset tarkvara, mis pakub API-sid preskriptiivse protsessimonitooringu funktsioonide rakendamiseks. Kasutajad saavad tööriista üles laadida sündmustepäeviku ning saada käimasolevate juhtumite kohta erinevaid retsepte, hõlmates järgmiste tegevuste ennustamist, ebasoodsate tulemuste tõenäosuse hindamist, ravi mõju pakkumist ja ressursside eraldamist ravikasumi alusel. Lisaks suurendab mooduldisain kohanduvust ja paindlikkust erinevates ärivaldkondades. Et pakutud lahenduse tõhusust hinnata, viiakse läbi nii nõuete täitmise hindamine kui ka jõudluse hindamine Business Process Intelligence Challenge (BPIC) andmekogude kasutamisega. Selle tulemusena annab see väitekiri panuse valdkonda preskriptiivse protsessimonitooringu tööriista pakkumisel mitmeid preskriptiivsete soovituste liike.

Võtmesõnad:
Retseptiivne protsessimonitooring, protsesside kaevandamine, protsesside optimeerimine

CERCS:
P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

Business process management (BPM) is a systematic approach to modeling, analyzing, and improving the efficiency of an organization's processes [DLRMAR13]. By aligning an organization's resources with its strategic goals, BPM enables continuous improvement, enhanced collaboration, and optimized performance across various departments and functions [DLRMAR13]. Process mining is a sub-field of BPM that aims to discover, monitor, and improve processes by analyzing historical data from event logs [Aal16]. It examines data such as ID, activity, and timestamp to discover business processes and can be used to pinpoint areas of improvement within processes. Process mining has numerous use cases, including compliance checking, bottleneck analysis, and optimization suggestions [VDA12].

Predictive process monitoring (PPM) is an extension of process mining that utilizes historical data to predict the outcome of ongoing cases or to forecast the next activity [DFGMM18]. The primary difference between predicting outcomes and predicting the next activity is the focus on the entire process outcome (e.g., successful loan application) rather than individual activities (e.g., credit check). These predictions can inform better decision-making and allow organizations to take corrective actions when necessary.

Prescriptive process monitoring (PrPM) is another advanced form of process mining that moves beyond predictions to provide practical recommendations for ongoing cases, and it is based on the idea that predictions are only meaningful if the system can provide actionable insights for better outcomes [KMND22]. Using live cases, PrPM can guide necessary interventions by suggesting various types of solutions, such as the best next activity to improve a specific Key Performance Indicator (KPI) [WDZM20], recommending an intervention that can improve a given business value [SD22], or triggering an alarm to prevent negative outcomes [TTdL$^+$18].

Although there have been advancements in PrPM, many existing methods still only focus on one type of recommendation [KMND22]. There is a gap in the field, as there is a lack of a comprehensive tool that can analyze data from different sources and offer an integrated prescriptive solution for process monitoring. The motivation for this thesis is to address this gap by developing a tool that can seamlessly work with diverse prescriptive methods and analyze multiple data sources.

To approach this problem, the thesis will follow the design science method to achieve the following objectives:

1. Design a solution for prescribing different types of recommendations for ongoing cases by allowing the integration of multiple prescriptive algorithms to be applied to the same event log.

2. Implement the solution in an application.

3. Evaluate the application's functionality using real-life datasets.

The main contribution of this thesis is an artifact that works as a prescriptive process monitoring tool and provides multiple types of prescriptive recommendations. Such an artifact is particularly valuable for analysts who work with process mining, as it expands their capacity to provide actionable recommendations and effectively intervene in ongoing cases.

The structure of this thesis is as follows: Section 2 presents the background of this field and existing similar works. Section 3 introduces the methods used to conduct this thesis. Section 4 provides the details of the developed artifact. Section 5 presents the evaluation process and the evaluation results. Finally, Section 6 summarizes the results, discusses the limitations of this work, and suggests future research directions.

2 Background and related work

This section introduces the relevant background of the research topic of this thesis and some similar works.

2.1 Background

Dumas et al. describe the concept of a business process as a sequence of interconnected activities, occurrences, and choices implemented by an organization to generate value for its clientele [DLRMAR13]. Furthermore, they characterize business process management (BPM) as a comprehensive framework that encompasses various principles, methodologies, and instruments to identify, examine, restructure, execute, and supervise business processes [DLRMAR13]. As more and more companies adopt BPM systems for management, a large amount of data related to business processes is generated in the BPM system, which can be utilized by process mining technology.

The idea of process mining is to discover, monitor and improve real processes by extracting knowledge from event logs readily available in today's information systems [AAM$^+$11]. An event log is a set of completed traces, where a trace is a non-empty sequence of events, all referring to the same case. Each event consists of a case identifier, activity name, timestamp, and other possible event attributes. Process mining enables analysis of these data in order to detect trends, identify bottlenecks, investigate frequencies, and provide insights [VDA12]. For example, in the case of a loan application process, process mining techniques can be applied to the collected event log data to generate a visual representation of the actual process flow, revealing patterns and providing actionable insights. This can help organizations identify process inefficiencies, cycle times, and compliance issues, enabling them to streamline their loan processing workflows, reduce processing time, and ultimately enhance customer satisfaction.

Based on the foundation of process mining, predictive process monitoring (PPM) emerged as an approach that forecasts the outcomes of ongoing cases. PPM utilizes event logs to analyze historical data and predict the likelihood of different outcomes for running cases, such as the completion time, resources required, or chances of success [TDRM19].

Unlike PPM which only provides predictions without offering interventions, the Prescriptive Process Monitoring (PrPM) is a collection of approaches for recommending or triggering interventions during the execution of a process to maximize its performance [SD22]. For example, some PrPM techniques can use business event logs to forecast negative outcomes that affect the performance of a process and use these predictions to determine if and when to initiate interventions to prevent or mitigate these negative consequences [FPTT$^+$22]. For example, in the loan application scenario mentioned earlier, the manager can use intervention recommendations provided by PrPM to reassign employees handling applications or adjust the application flow for special cases, thus improving the success rate of cases or shortening processing time.

In fact, there are various methods to achieve PrPM. Kubrak et al. [KMND22] summarized three types of methods for giving prescriptive recommendations, which are guiding methods, correlation-based methods, and causality-based methods.

Guiding methods give suggestions for ongoing cases that are similar to previous cases, based only on historical traces [KMND22]. For example, Arias et al. used historical information on the process execution and expertise information to prescribe recommendations for team formation, which uses the similarity of traces for resource allocation recommendation [AMGS16].

Correlation-based methods give recommendations to improve the process with respect to some key performance indicators, such as duration, cost, or the positive outcome of the case [KMND22]. For example, Fahrenkrog-Petersen et al. proposed a framework for triggering intervention actions based on the predictions provided by PPM by constructing a cost model and setting off alarms [FPTT$^+$22].

Causality-based methods rely on causality, and can be used to recommend interventions and their effectiveness [KMND22]. For example, Bozorgi et al. proposed an approach to generate case-level intervention recommendations with effectiveness by discovering causal rules from event logs [BTD$^+$20].

2.2 Related work

Some algorithms are dedicated to providing one kind of recommendation using business process data. For example, in guiding method algorithms, Abdulhameed et al. proposed a method that computes the co-working relationships of resources based on the frequency and processing time of doing a task in previous to suggest the high-ranking resource that would be best for co-working with others [AHAE18]. In correlation-based method algorithms, Fahrenkrog-Petersen et al. proposed a method that evaluates the probability of a negative outcome together with a cost model and the mitigation effectiveness to trigger an alarm. While in causality-based method algorithms, Shoush and Dumas proposed a method for allocating resources by building a causal estimation model [SD22]. However, these algorithms can only provide one type of recommendation. Although Kubrak et al. collected various PrPM algorithms, summarized them, and proposed a framework to characterize prescriptive process monitoring methods [KMND22], it does not implement a solution to combine them. We are looking for a solution to combine different types of prescriptive algorithms to provide more effective recommendations.

Regarding academic research-based similar software tools, there are also some tools that implement predictive or prescriptive process monitoring. For example, Nirdizati and ProLift. Nirdizati is an integrated open-source process monitoring platform that supports users in selecting and tuning various prediction models and enables the continuous prediction of different process performance measures at runtime [Ver18]. Nirdizati supports a scalable and modular architecture, provides hands-on configurable performance metrics, supports explanation, and provides indications on the preprocessing of the event log for the user [RDFGM22]. However, Nirdizati only supports predictive process monitoring and thus does not support prescribing intervention based on causality. ProLift is a causal rule discovery tool that can help process workers discover cases that may be improved by an intervention [BKD$^+$22]. Still, the tool only uses one modeling technique and can only provide one type of recommendation, i.e., whether predefined interventions have benefits in specific open cases.

In the industry, there are also some analytical tools. Some tools provide predictive process monitoring, such as Apromore, Rapidminer, and Alteryx. Apromore provides a predictive process monitoring add-on that uses a training module to analyze historical data, train predictive models, and uses a runtime monitoring module to provide prediction results for open cases[1]. Rapidminer is a data science platform that can provide insights into user behavior and predict possible results of cases using historical data[2]. Alteryx is an analytics cloud platform that offers predictive analytics services, enabling users to select algorithms and specify parameters to improve processes through training historical data[3]. However, these tools are based on predictive methods and cannot provide intervention recommendations based on causality. In addition, some tools provide prescriptive analytics functionality, such as FICO, IBM Prescriptive Analytics, and AIMMS. FICO uses prescriptive analytics to implement data-driven decision-making to improve the customer lifecycle [4]. IBM Prescriptive Analytics solutions use prescriptive optimization methods to improve operations, increase efficiency, and mitigate risks[5]. AIMMS focuses its purpose on using prescriptive analytics to improve supply chain decisions and provides map-based visual results [6]. However, these tools focus on providing a wider method set, including simulation, optimization, and decision analysis. And some of them are designed for a specific purpose. There is still a need for more specialized tools that focus explicitly on prescriptive process monitoring for different industries and contexts.

In summary, although there are some similar works and industry solutions, the gap that this thesis aims to fill has not been solved by these solutions.

[1]Apromore: `https://apromore.com/predictive-process-monitoring/`
[2]RapidMiner: `https://rapidminer.com/solutions/domain-experts/`
[3]Alteryx: `https://www.alteryx.com/glossary/predictive-analytics`
[4]FICO: `https://www.fico.com/en/fico-platform`
[5]IBM Prescriptive analytics: `https://www.ibm.com/prescriptive-analytics`
[6]AIMMS `https://www.aimms.com/`

3 Method

This study aims to design a feasible solution that allows a single system to provide different kinds of prescriptive recommendations for ongoing cases. Moreover, the solution needs to be implemented to explore the methods or technologies that can be used, assess the feasibility of the solution, and evaluate whether it is generalizable across different dataset inputs from the perspective of various application scenarios.

Therefore, the goals of this thesis are defined as follows:

1. Design a solution for prescribing different types of recommendations for ongoing cases by allowing the integration of multiple prescriptive algorithms to be applied to the same event log.

2. Develop an application to implement the solution.

3. Evaluate the functionality of the application using datasets.

The first goal is primarily concerned with the design of the logical aspects of the solution. The second focuses on the implementation level of the solution. The problem is related to how to implement the solution as a system, provide a way to integrate the system with the user's existing process mining system, and ensure the system's ability to support more types of prescriptions.

In order to answer these goals, this thesis followed the design science research methodology. Since we aim to solve a problem (how to prescribe different types of intervention to ongoing cases) by creating and evaluating an artifact (an implementation of the solution), the design science research method is a suitable approach. According to Peffers et al., using design science as a research methodology is an effective approach for creating solutions that address real-world engineering issues [PTRC07].

As Figure 1 shows, this research will take the following steps:

1. Identify the problem domain and the requirements for the artifact;

2. Design, build, and evaluate the artifact based on the proposed solution;

3. Evaluate the artifact;

4. Extend the existing knowledge base by providing new artifacts and new experiences.

Section 3.1 will explain how requirements are collected. Section 3.2 will describe the factors that will be considered when selecting architecture, technology stack, and development management methods. Finally, Section 3.3 will explain how artifacts are evaluated and tested. Section 3.4 declares the use of AI-based tools in paper writing.

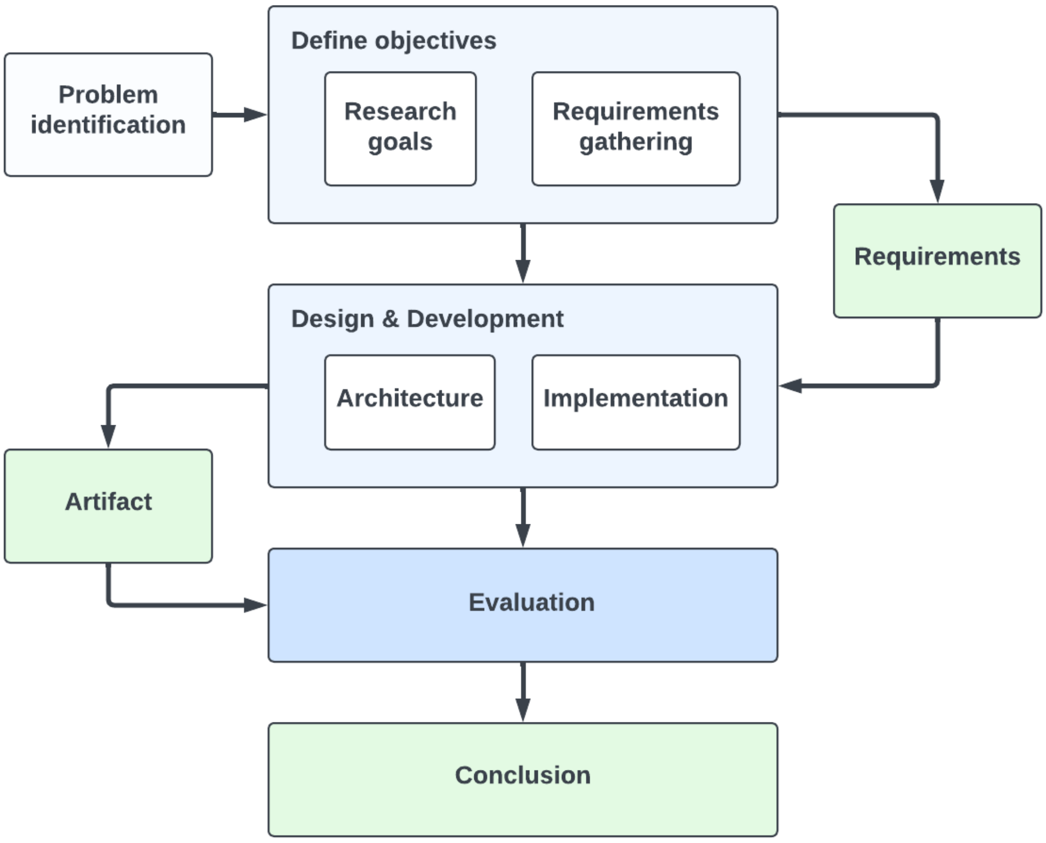


Figure 1. Design science method.

3.1 Requirements gathering

Requirements gathering and prioritizing are the first steps in designing the artifacts. In this step, we collected and prioritized the requirements based on the following sources.

3.1.1 Collect from related tools

The related artifacts mentioned in Section 2.2 were used as references, which are not based on specific application scenarios but provide some general approaches that can be used to solve different application scenarios. The characteristics of these artifacts are also used as a possible list for the requirements gathering of the artifacts.

3.1.2 Collect from related researches

We referred to the research of Kubrak et al. on the systematic literature review (SLR) [KMND22] and user interface concept for prescriptive process monitoring [KMND23] to determine which types of algorithms should be supported by the tool. We also identified the format and structure of training data, new data for analysis, and output data that should be supported by the tool in actual production environments. Additionally, during sprint meetings, our research team discussed and selected specific algorithms that should be implemented in the tool as the default example plugins.

3.1.3 Prioritization

In order to first verify the feasibility of the framework, solution, and artifact pipeline, we prioritized the requirements from the collection of requirements. This study prioritizes the requirements according to the MoSCoW method and classifies them into four categories.

- Must have: requirements that must be implemented.
- Should have: requirements that should be implemented.
- Could have: requirements that can be implemented.
- Won't have: requirements that will not be implemented.

In the feasibility verification phase, the study first implements the *Must have* requirements and creates a minimum viable product (MVP) to verify the designed solution.

3.2 Design and development

This section outlines the decision-making process for design choices, including the factors considered in selecting the architecture and the modeling techniques utilized. Furthermore, it delves into the rationale behind these selections.

3.2.1 Formulate Solution

This thesis prioritizes high-priority requirements and considers them as must-achieved goals when formulating the solution. The following aspects are considered in the design:

- The input and output data formats may be different for algorithms.
- The training time may vary significantly for different kinds of algorithms.
- Some algorithms require the labeling work of data, and some do not.

- Some algorithms may require human involvement to determine the parameters or intervention candidates.

Also, based on studies related to interventions [WRRM08, KMND22], several main types of interventions can be treated as desirable types of results:

- Control flow: suggesting the flow or sequence in which activities occur;
- Resources: suggesting specific resources for performing activities;
- Performance: suggesting to perform activities at different points in time.

Moreover, when thinking of the solution, it's notable that current intervention-based studies have not yet addressed the problem of automatically identifying available interventions. As they only predict the impact of taking each action on negative outcomes after the interventions have been identified, the selection of possible intervention candidates still requires the involvement of domain experts [KMND22].

3.2.2 Architecture

We have defined this tool as a back-end service utility that is capable of providing RESTful APIs. In doing so, we considered the following factors:

- The RESTful API provided by the back end, not only can be used by some front-end applications but also can be integrated with other business systems.
- The tool ultimately provides prescriptive recommendations as result data, rather than visual representations of the results. The setting and visualization design of the results is not within the research objectives of this thesis.

When choosing a back-end architecture, this study considers the following factors:

- Coupling between modules. The lower the coupling between different modules, the higher the maintainability of the system.
- Scalability. The system needs to be able to support an increasing number of algorithms and therefore needs to be highly scalable.
- Portability. The system needs to be able to run on different platforms, so it needs to be highly portable.

3.2.3 Modelling

The implementation used class diagrams and sequence diagrams for modeling, and the reasons for choosing these two modeling methods are as follows:

- Class diagrams are used to describe the relationship between classes in a system and are the basis of object-oriented design. In this study, we need to describe the relationship between classes to understand the system's logical structure, so class diagrams are chosen as the modeling method.
- Sequence diagrams are used to describe the interactions between objects in the system. In this study, we need to describe the interactions between objects in the system to show the system's workflow, so we choose sequence diagrams as the modeling method.

3.2.4 Development

This section describes the artifact's development process, including the project management methodology, the sequence of adding algorithms, the data sets used, and the development of supporting tools.

Project Management Project management uses some agile development methods. We divided the requirements into sprints based on a list of requirements, and each sprint had a plan that includes the sprint's goals and the sprint's timing. Each sprint had a two-week cycle. This process can be seen in Figure 2.

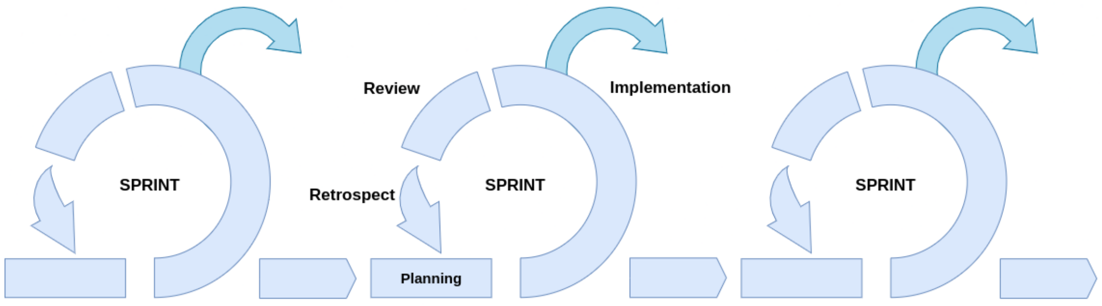


Figure 2. Sprint circles.

During the requirements phase, existing requirements might be modified, and new requirements might be added. These requirement modifications and additions were documented in the sprint plan so that they can be discussed during the sprint review. Functional tests are performed at the end of the design and development phase to ensure that the modules meet the requirements. At the end of the sprint, the supervisors conducted a sprint review with the author, including whether the goals of the sprint were met and whether the tasks of the sprint were completed. The results of the sprint review were used as a plan for the next sprint.

Adding Algorithms After implementing the MVP pipeline, logic modifications and additions are made on top of the pipeline, and the new algorithms added drive local design changes. Before adding an algorithm, the algorithm is analyzed, including its inputs and outputs. The results of this analysis are documented in the algorithm's documentation so that subsequent developers can refer to them.

The order in which the algorithms are added is based on the complexity and difficulty of the algorithm, and based on the study of Kubrak et al. [KMND22], we added the algorithms in this order:

1. Guiding algorithm;
2. Correlation-based algorithm;
3. Causality-based algorithm.

3.3 Evaluation

This section describes how to evaluate the system, including acceptance criteria, the data used in the evaluation, and how the evaluation is done.

3.3.1 List of acceptance criteria

The artifact proposed in this thesis can be accepted if it meets the following criteria:

- The *Must have* and *Should have* requirements fulfilled;
- The combination of the algorithms can provide results as a single file;
- The application can be successfully deployed with a demo instance.

3.3.2 Evaluation dataset

We use three datasets that come from BPIC[7]. In a natural production environment, data is generated in the form of streams, so we need to develop an event streamer to simulate the generation of data streams during the evaluation process. The main functions of this streamer include the following.

[7] https://www.tf-pm.org/competitions-awards/bpi-challenge

- Reading data from a specified dataset.
- Sending data to the application in the chronological order of events.
- Sending data to the application at a specified rate.
- Record the results of the application feedback.

3.3.3 Evaluation method

We performed a requirements fulfillment evaluation and a performance evaluation.

For requirements fulfillment evaluation, we used test cases and sample datasets, and predefined definitions to test the completeness of the features provided by the tool.

For technical evaluation, we measured the performance of the tool. First, we identified relevant metrics to determine one of the key performance indicators: for example, response time, throughput, or resource utilization. Then, we created a set of experiments to measure the identified metric using different datasets. Finally, we run the experiments, collect data for each metric, and show the results.

3.4 Usage of AI-based tools

This section clearly states the use of AI-based software tools during the writing process of this thesis. The following tools are used in the writing of this thesis:

- ChatGPT: Used to translate the English abstract of this thesis into Estonian and for grammar correction.
- Grammarly: Used for grammar correction and to improve readability.

4 Results

This section will introduce the requirements collection results of the developed artifacts, the architecture used during development, an introduction to the functions of each component, implementation details of each component, deployment methods for the artifacts, and an introduction to supporting documentation.

4.1 Requirements

This section will introduce the specific sources and results of requirement collection, as well as the priority we assign to these requirements.

4.1.1 Functional requirements

As we discussed in Section 3, the collected functional requirements came from references to similar tools, such as Nirdizati [RSF$^+$19] and ProLift [BKD$^+$22], and references to SLR paper [KMND22] and prescriptive process monitoring user interface concept research of Kubrak et al. [KMND23] Also, some requirements were derived from the discussions within the research team.

These requirements are presented in Table 1, each requirement has a unique identifier and corresponding description.

Table 1. Functional requirements

ID	**Description**
FR-1	Support for processing uploaded historical event log files in CSV, XES[8], and ZIP formats.
FR-2	Ability to provide column definitions for the uploaded log file.
FR-3	Creation of a project based on the event log file by providing outcome and treatment definitions.
FR-4	Ability to update basic information of the project, such as name and description.
FR-5	Ability to connect to multiple machine learning plugins to train models based on the created project.
FR-6	Support for setting plugins' parameters.

[8]About XES: `https://www.tf-pm.org/resources/xes-standard/about-xes`

Continuation of Table 1	
ID	**Description**
FR-7	Support for providing additional information for custom-developed plugins.
FR-8	Ability to disable or enable plugins after creating a project.
FR-9	Support for redefining column definitions or outcome and treatment definitions.
FR-10	Ability to re-upload a new log file while keeping parameter settings and additional information.
FR-11	Support for deleting a project.
FR-12	API for getting a list of all existing projects.
FR-13	Support for uploading new datasets for ongoing cases and obtaining results with different types of prescriptive recommendations.
FR-14	Ability to stream new event data to the system and obtain SSE-based data stream results with different types of prescriptive recommendations.
FR-15	Support for simulating events stream for testing conveniently.
FR-16	API for downloading preprocessed data, originally uploaded files, generated test files that only contain ongoing cases, and the dataset used for simulating streaming data.
FR-17	Support for loading, enabling, and disabling plugins during the deployment phase. This feature can be used for adding more custom plugins to provide other types of prescriptions.

4.1.2 Non-functional requriements

When proposing non-functional requirements, the following aspects were mainly considered:

- With respect to usability, it is crucial that the system processes transaction logs, trains models, and predicts results within an acceptable time frame to ensure efficiency and utility.
- Given that the artifact developed in this research constitutes backend software and offers Application Programming Interfaces (APIs) for integration with frontend ap-

plications, it is essential to provide comprehensive documentation or user manuals for frontend developers to facilitate seamless integration.

- Considering the primary value of the artifact, the time and development costs associated with developing new plugins should be relatively minimal. Ideally, plugin developers should focus on algorithm implementation without needing extensive knowledge of the overall system logic.

Consequently, we put forth an array of non-functional requirements as depicted in Table 2. In this context, we employ the BPI Challenge 2012 dataset [9] processing time as a benchmark. The specified time constraints are established based on executing programs on a computing system equipped with an AMD Ryzen™ 9 5900HX Mobile Processor [10] and 32GB of RAM.

Table 2. Non-functional requirements

ID	**Description**
NFR-1	When calling the API locally, the time used for a training phrase on the BPIC-2012 dataset should be under 60 seconds.
NFR-2	When calling the API locally, the duration required for predicting outcomes on an ongoing cases dataset, derived from the BPIC-2012 dataset, should not exceed 60 seconds.
NFR-3	The plugin system should allow for easy modification, replacement, or addition of prescribing and predicting plugins.
NFR-4	Comprehensive documentation should be provided for core application's usage and plugin development details.
NFR-5	the service should ultimately be deployed to a publicly accessible server for others to test and integrate with the visualization layer.

4.1.3 Requirements prioritization

In order to achieve rapid iteration and validate design concepts first, it is necessary to prioritize requirements during the development process. Here, we organize functional requirements according to the MoSCoW model mentioned in Section 3, as shown in Table 3.

[9] BPIC 2012: `https://data.4tu.nl/articles/BPI_Challenge_2012/12689204`

[10] AMD 5900HX processor: `https://www.amd.com/en/products/apu/amd-ryzen-9-5900hx`

Table 3. Requirements prioritization

Priority	Requirements
Must have	FR-1, FR-2, FR-3, FR-5, FR-12, FR-14, FR-15, FR-17
Should have	FR-4, FR-6, FR-7, FR-8, FR-11, FR-13
Could have	FR-9, FR-10, FR-16

4.2 Architecture design

The tool implemented by the thesis is used as one of the backend services of Kairos [11]. The overall architecture of Kairos is shown in Figure 3. The dashed gray area represents the visualization layer application of Kairos, which is designed and developed by another thesis's author[12]. The architecture of the artifact developed in this thesis is shown below that area. As we can see, the back end consists of several main components: core application, dataset processor, event streamer, plugins, database, and message broker.

Microkernel architecture. The backend design adopts the microkernel architecture pattern. This is because modular design should be used to facilitate the addition of more algorithms. The optional architectures are:

- Microservice architecture: breaking down a large application into multiple independent, loosely coupled small services that can be developed and deployed separately to improve system flexibility, scalability, and maintainability.
- Event-driven architecture: based on event-triggered and processing mechanisms, decoupling various system components for responsive interaction to improve system scalability and flexibility.
- Microkernel (plugin) architecture: core functions provided by a small kernel while other functions are extended through pluggable add-ons to achieve a flexible and easy-to-maintain system structure.

Considering that the data granularity of microservices requires each service to have its own database, it is unnecessary for this application and increases the development and maintenance burden; therefore, it will not be adopted. In addition, considering generality requirements we want to provide HTTP-based APIs for any possible external consumer programs including the visualization layer so that external programs interact with this application via API calls directly processed by business logic without

[11]Kairos: `https://kairos.cloud.ut.ee`

[12]Karios visualization layer: `https://github.com/visualpm`

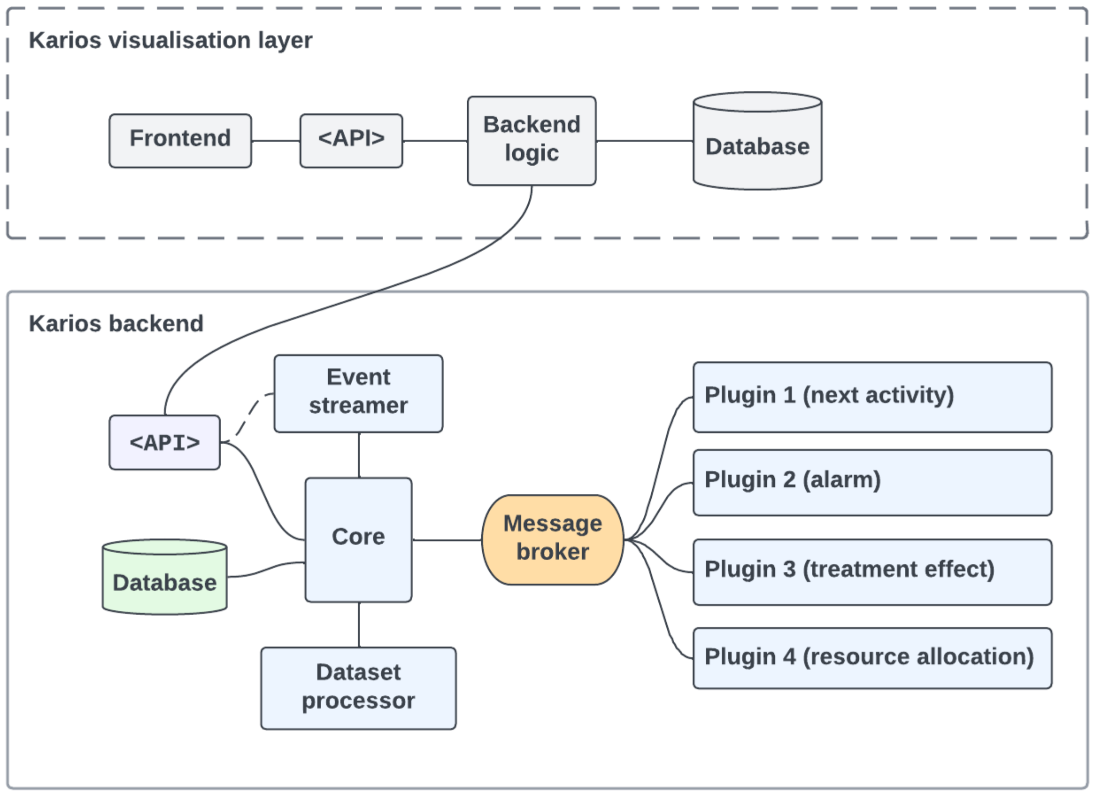


Figure 3. Overall architecture of Kairos.

abstracting events which would otherwise be cumbersome given current needs; hence full adoption of event-driven architecture is not necessary. Finally, considering the need to add more plug-and-play machine learning algorithms for providing more prescription types we adopt microkernel architecture for adding functionality in isolation but at the same time borrowing ideas from both microservices and event-driven approaches by independently encapsulating different components achieving functional decoupling as well as asynchronous processing capabilities enabling distributed deployment.

Core application. The core application is responsible for coordinating the workflow of the entire system. The core application mainly collaborates with the dataset processor to preprocess event logs, provide APIs to the visualization layer of Kairos, store necessary data in the database, communicate with plugins through the message broker, provide data required by plugins, and process the results returned by plugins. Some components of the core application also serve as a common code library for plugins, which can be used by plugins to automatically handle some common tasks, reduce the difficulty of plugin development, and allow plugin developers to focus solely on implementing the plugin algorithm.

Dataset processor. The dataset processor is an independent worker that is responsible for processing event log files and converting them into data formats that can be used by plugins. It utilizes a multi-process approach to effectively use multi-core CPUs and improve processing efficiency.

Event streamer. Event streamer is used to simulate the process of an external business management system sending new events to the application. After event logs have been processed, analyzed, and trained, the system can accept push notifications for new events and provide prescription recommendations in a timely manner for ongoing cases. Event streamer is designed specifically for this purpose: if the application has not yet been integrated into the business management system but users want to test its functionality, they can enable simulation mode. In this case, the event streamer will use test data and call the application's API to push new events. The system will generate results based on these new events and be consumed by external programs.

Plugin system. Plugins are independent modules that are responsible for providing a specific type of prescription for ongoing cases. Four distinct example plugins have been developed to extend the application's capabilities. The first plugin is responsible for predicting the next activity in the ongoing case, while the second plugin assesses the likelihood of negative outcomes. The third plugin provides a score that indicates the need for intervention. The fourth plugin, in turn, allocates resources based on the gains of the ongoing case.

4.3 Implementation

This section introduces the specific details and technical choices for artifact development.

4.3.1 Technology stack

This section will introduce the technology stack used in implementing the artifact, as well as the reasons for choosing them.

The programming language is Python 3. The choice of language is mainly based on the following reasons:

- Support for machine learning algorithms. As a general-purpose programming language, Python has a rich set of third-party libraries, including those related to machine learning, such as scikit-learn, numpy, and so on. These libraries can help us implement machine learning algorithms quickly.

- Concise and interpretative. The Python language has a concise syntax and is easy to understand, which helps improve readability.
- The authors have experience using the language.

The database is PostgreSQL, and the choice of database is mainly based on the following reasons.

- Relational database, which has a data structure that can support process mining systems well.
- It supports multi-platform deployment, and can be deployed on Linux, Windows, macOS, etc., for evaluation and testing.
- Support JSON data type, which can well support the data structure of process mining system.
- The ORM (Object-relational mapping) library SQLAlchemy for Python supports the integration of this database.
- The authors have experience with this database.

The back-end API framework uses FastAPI, and the framework was chosen mainly for the following reasons.

- Support for asynchronous programming, which can improve the system's performance.
- Swagger documentation can be provided to facilitate front-end development and integration with other systems.
- Fast responsiveness to achieve better performance. According to the web framework benchmark provided by TechEmpower[13], FastAPI is one of the fastest frameworks for the Python language.

4.3.2 Core application

We will introduce its specific functions and technical details from several aspects, including the workflow of the core program, object modeling, and interaction with external programs.

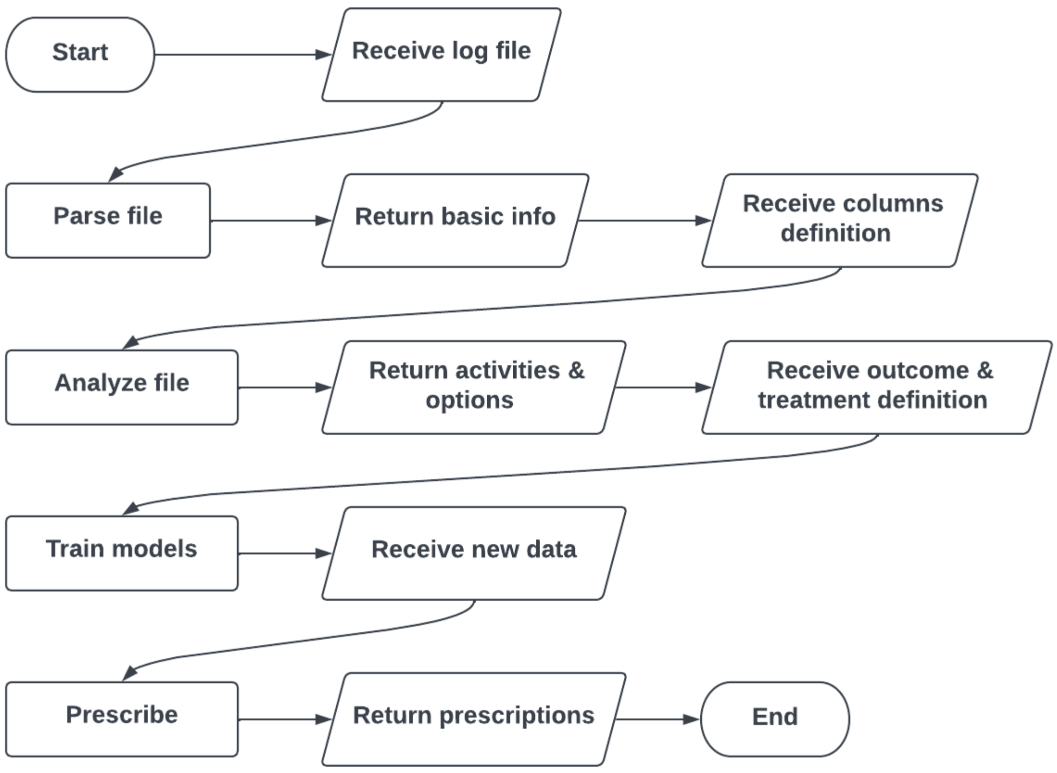


Figure 4. Simplified flowchart of core application.

Flowchart. The simplified flowchart is shown in Figure 4. Please refer to Figure 23 in the appendix for the complete flowchart.

According to the flowchart, the first stage involves uploading the event log file, which should contain only completed cases, and defining the columns and project parameters. This stage also includes parsing the log file and checking for its validity. If the log file is valid, basic information is returned to the API consumer, while an invalid file result at the end of the process.

The second stage of the process is preprocessing and training. After defining the columns and project parameters, the system preprocesses the data according to the definitions provided. The outcome and treatment definitions are also defined in this stage, which the system uses to train the models. The system validates the project definition and checks if it is valid or not. If the definition is valid, the models are trained; otherwise, the process ends.

The third stage of the process involves sending new data to the system. Users can

[13]https://www.techempower.com/benchmarks/

either upload a new dataset or stream new event data to obtain prescriptions for ongoing cases. If users choose to upload a new dataset, the system validates the dataset and prepares it for processing. If the dataset is valid, the system provides prescriptions for valid cases; otherwise, the process ends. Alternatively, if users choose to continuously push new event data to the system, the system validates the data and saves it to the database. If the data is valid, the system provides prescriptions for valid cases in a streaming manner; otherwise, the process ends.

Finally, in the fourth stage, the system returns the prescriptions to the user. The user can then use these prescriptions to improve their processes.

Class diagram. The system is structured using several core classes, as depicted in Figure 5.

The EventLog class represents the event log file and contains details such as the file name, saved name, and associated Definition object. The Definition class contains definitions necessary for system training, such as column definitions, case attributes, outcome definitions, and treatment definitions, among others.

The Project class represents a project that uses the system for prescriptive process monitoring. It contains details such as the project name, description, and associated EventLog object, as well as a list of associated Plugin objects.

The Plugin class represents a plugin in the system and contains details such as the plugin name, description, and status.

The Event and Case classes are used to represent events and cases in the stream data. By utilizing these classes for storing stream data, the system can provide timely and prescribing recommendations for ongoing cases, allowing users to make informed decisions about their processes.

Overall, the system design is highly structured and organized, with each class containing details essential for efficient prescriptive process monitoring.

Sequence diagram. Figure 6 shows the interaction between the front-end and back-end during the project creation phase. Upon successfully training the project, new data can be incorporated into the project to retrieve prescriptions. This can be done via two channels, viz. uploading a new dataset to the project or streaming new events data via API calls to the project. In addition to this, the simulation feature embedded in the tool can be leveraged to test this functionality effortlessly without the need for any coding.

Figure 7 assumes that the user will first use a new test dataset to obtain all prescriptions in one call. Upon successfully uploading a new dataset, the API response will include a `result_key`. In the event that a new test dataset was already uploaded during project creation, a `result_key` would have been provided in the response as well. The `result_key` can be utilized to retrieve the dataset result.

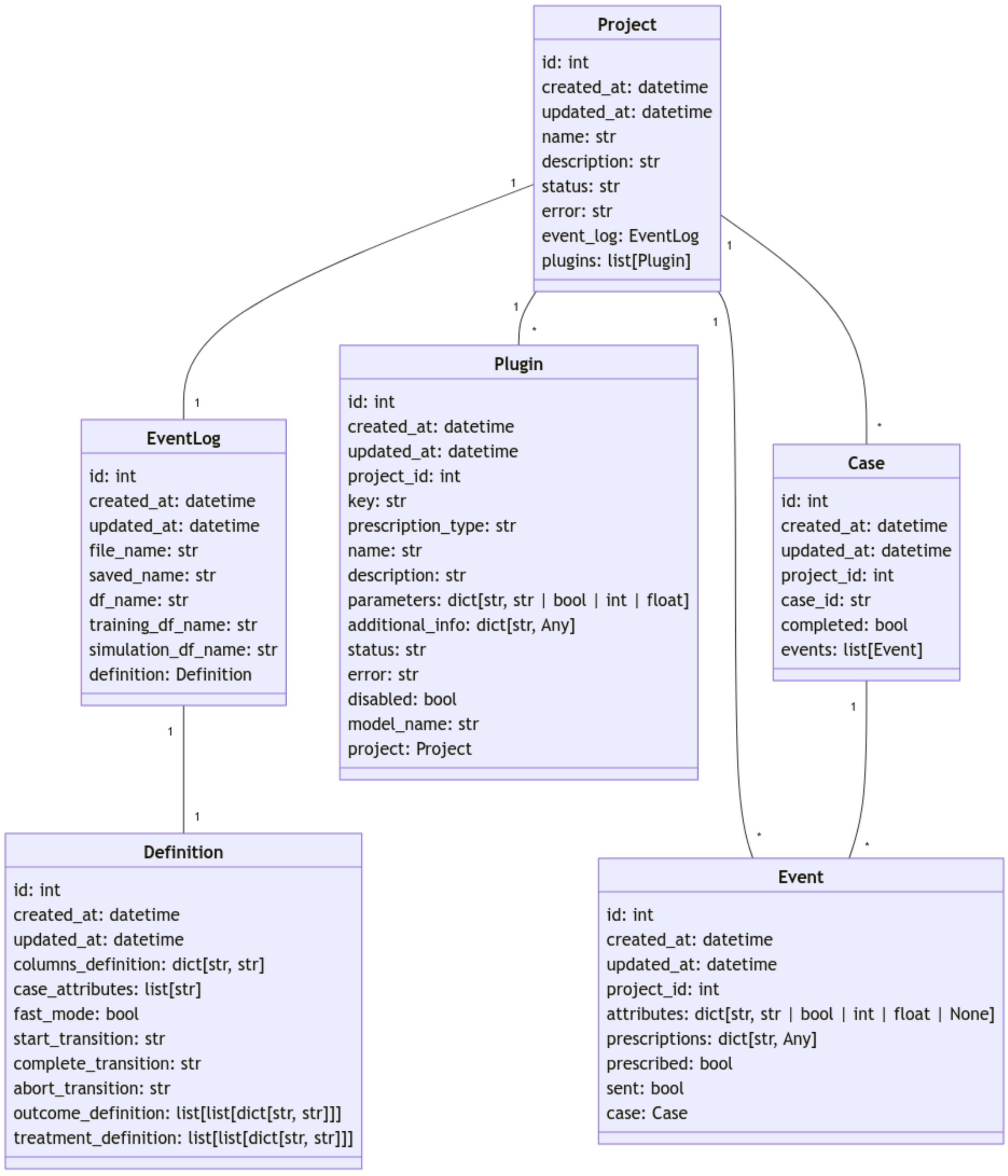


Figure 5. Class diagram of the core application.

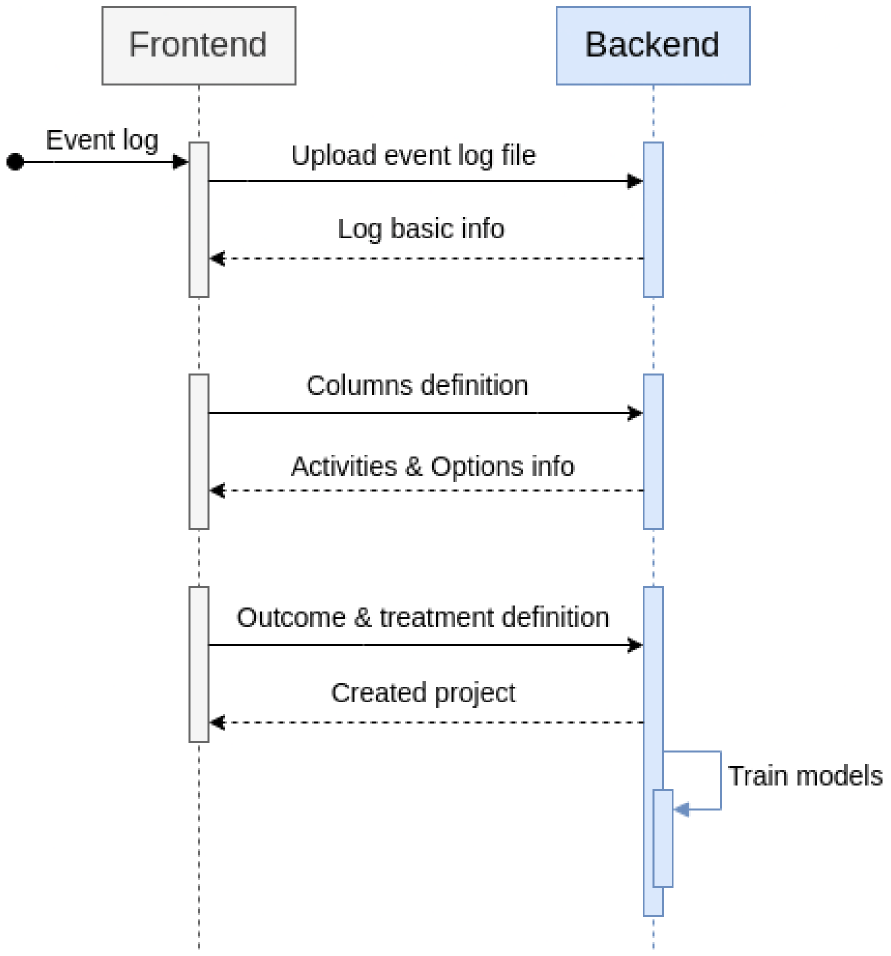


Figure 6. Interactions between frontend and backend when creating a project.

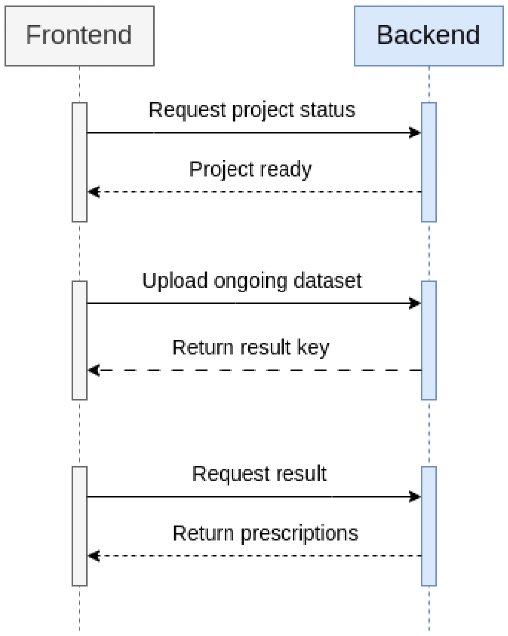


Figure 7. Interactions between frontend and backend when uploading new ongoing cases dataset.

Subsequently, Figure 8 assumes that the user will use the streaming API to post new events and receive new prescriptions in a streaming manner. If the client subscribes to the SSE endpoints, A message will be sent to the client every time a new prescription is available.

4.3.3 Dataset processor

This section will introduce the workflow and necessary technical details of the dataset processor.

Technical choices. Due to our use of Python for development, and the global interpreter lock in Python limiting its ability to utilize multi-core processing, although we can use the `multiprocessing` library to start new processes within the program, potential issues with data processing and recognition between processes may result in additional costs for development and maintenance. Therefore, we have separated the component responsible for preprocessing data from the core application. This has three benefits:

1. Running in an independent environment allows us to fully utilize multi-processes and multi-core processing programs without worrying about performance issues

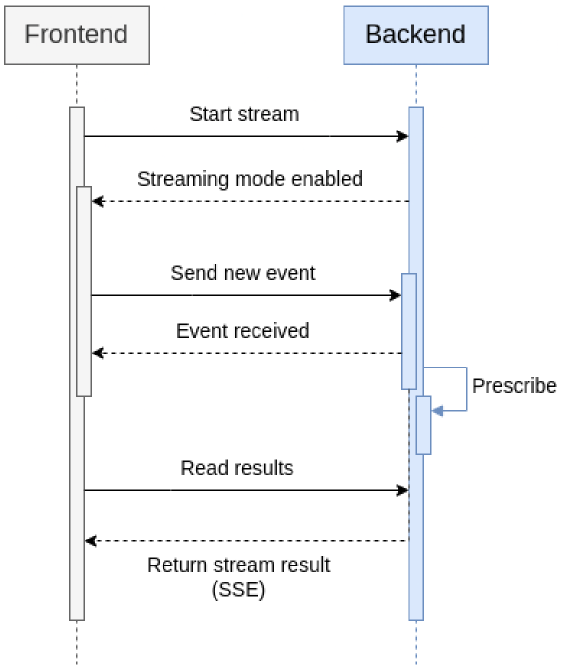


Figure 8. Interactions between frontend and backend when streaming new data.

caused by occupying uvicorn's worker compared to integrating it into the core application.

2. It improves fault tolerance as pre-processing services run separately and are not affected by other services.

3. It provides possibilities for distributed deployment where multiple instances can be run for load balancing.

Given that we will be using RabbitMQ[14] as the message broker to facilitate communication between plugins and core application, the communication between dataset processor and core application will also be done through RabbitMQ. This is illustrated in Figure 9. The Core application sends a `PROCESS_REQUEST` request to the dataset processor which includes an event log file, definitions for each column, and conditions used for labeling outcomes and treatments. Once the dataset has been preprocessed, the dataset processor returns a `PROCESS_RESULT` data which includes processed files along with processing time statistics.

[14]RabbitMQ: `https://www.rabbitmq.com`

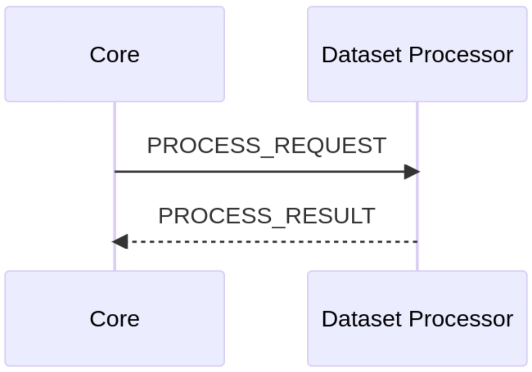


Figure 9. Communication between core application and dataset processor.

Workflow. When processing files, the dataset processor performs operations in the following steps:

1. Automatically identify timestamp text within files and convert it into a `datetime` object.
2. Calculate the duration of each case from the first event to the last event ended as a case attribution column.
3. If XES file is being processed with transition information included, automatically filter events based on timestamps within the XES file.
4. Convert values belonging to numeric columns from strings into integers or floating point numbers according to user-provided column definitions.
5. Convert values belonging to boolean columns from strings into boolean values according to user-provided column definitions.
6. Analyze each case based on user-provided column definitions along with outcome and treatment definition/conditions in order to generate labels for outcome, treatment, and treatment resource.
7. Finally rename columns according to predefined conventions so that plugins can select necessary data for training purposes.

Event timestamp recognization When processing an XES file as an event log, the processor assumes that it conforms to the standard lifecycle transition model[15]. As the core application requires identification of a timestamp for each event, the dataset processor will utilize transition and timestamp information to extract it.

Initially, the tool checks if all rows in the event log lack a `start` transition. If any row does not contain this information, the processor uses the timestamp of the row that has a `complete` transition to identify the timestamp for each event. Alternatively, if there is no `complete` transition present in the event log, the processor uses the `start` transition to identify the timestamp for each event. If `start` transition and `complete` transition are absent in the event log, the processor preserves the timestamp of the row as is, and thus all rows remain unmerged.

When the fast mode is enabled, which is the default setting, the processor uses the `complete` transition and `ate_abort` transition to identify the timestamp for each event if both `start` transition and `complete` transition are present in the event log.

However, when the fast mode is disabled by the user, the processor uses the `start` transition to identify the start timestamp for each event and uses the `complete` transition to identify the end timestamp for each event if both `start` transition and `complete` transition are present. In this mode, two timestamps are obtained for each event. It is important to note that non-fast mode may take a long time to process depending on the size of the event log. If the event log is too large, the system may not allow users to disable fast mode. In such a case, it is recommended to pre-process the event log to obtain the timestamp for each event and convert the file to CSV before uploading, or simply use the fast mode.

In certain cases, the XES file may not adhere to the lifecycle standard or may not fulfill the unique demands of some business domains. In these scenarios, the tool grants the flexibility to specify the `start`, `complete`, and `ate_abort` transitions in alignment with the prerequisites.

Generation of outcome and treatment labels. The dataset processor accepts nested conditions for identifying multiple outcomes and treatments to label them. For example, Figure 10 shows how the processor determines the category of an outcome based on its nested conditions when a user specifies the definition of a positive outcome. Similarly, the developed tool also allows users to specify negative outcomes' definitions, and treatment categorization works similarly. Additionally, if a user specifies a resource column, then the processor identifies the first resource that triggers treatment in a case and adds it as case attributes to the dataset.

[15]Standard lifecycle transition model: `https://www.tf-pm.org/resources/xes-standard/about-xes/standard-extensions/lifecycle/standard`

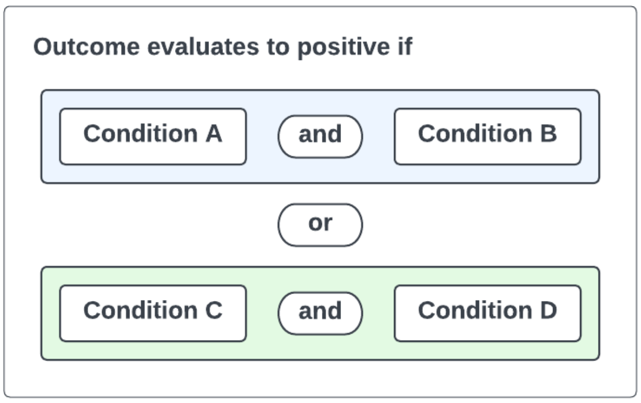


Figure 10. Dataset processor determines the label of outcome based on conditions.

Usage of multiple CPU cores. When processing the dataset, the dataset processor will split it into several parts according to the number of CPU cores and use a process pool for parallel processing to speed up the processing speed.

4.3.4 API service

This section introduces the details of the API services provided by the core application.

RESTful API. This tool mainly interacts with external programs through RESTful API. REST stands for representational state transfer[16], and RESTful API refers to web services that follow the REST style.

The following considerations were taken into account when choosing HTTP-based web APIs:

- More flexible compatibility: most API management tools and programming language ecosystems have very mature implementations for calling RESTful APIs, so using a RESTful API has better compatibility.
- Simple integration, which is conducive to collaboration between visualization layer authors.
- The front end can directly call the API in the browser to display some data.

The CRUD-based functionality provided by this tool's API fully follows the REST style. Table 4 shows some examples of API endpoints for the `project` resource, and

[16]What is a REST API? `https://www.redhat.com/en/topics/api/what-is-a-rest-api`

the complete table can be found in Appendix Table 24. We can see that in the case of uploading the entire new dataset and obtaining all prescription suggestions, the program will provide an interface for uploading data and obtaining results to external parties through RESTful API. In the case of obtaining prescription suggestions through streaming data, the client will also use RESTful API to send new events to the server. This is because new events may come from various systems rather than the visualization layer. Therefore, HTTP-based APIs that are generally applicable to various network and system environments are our preferred method for receiving events.

Table 4. API endpoints partial example

Method	**Endpoint**	**Description**
POST	`/project`	Create a new project based on uploaded event log
GET	`/project/{project_id}`	Get an existing project by ID
PUT	`/project/{project_id}`	Update a project's name and description
DELETE	`/project/{project_id}`	Delete an existing project
POST	`/project/{project_id}` `/result`	Upload a new ongoing dataset to a trained project to start prescribing
GET	`/project/{project_id}` `/result/{result_key}`	Get the prescription result from a uploaded dataset
POST	`/project/{project_id}` `/stream/event`	Push a new event to a trained project under streaming mode

SSE API. Considering the scenario where the prescription result is obtained through streaming data, there is a need for some time to analyze the case of new events by the program and its plugins. As a result, the server does not immediately return results to the client upon receiving new events. Therefore, it is necessary to establish a way for clients to continuously receive prescription result stream data. There are four common solutions: polling RESTful API on the client side to obtain new data, using message broker systems such as Kafka to transmit data, establishing client-server connections through WebSocket for bidirectional transmission of data, and subscribing clients with Server-Sent Events (SSE) API and sending them streaming data afterward. We chose SSE API mainly because of its following advantages:

- Compared with the polling method, the HTTP overhead of SSE API is low. In low-event frequency situations, unnecessary polling calls will not be made.
- Compared with using the Kafka method, SSE API follows HTTP web standards and is consistent with other APIs provided by this tool for external applications which reduces integration complexity.
- Compared with the WebSocket method, the SSE API protocol is lighter weight and supports automatic reconnection by default.

As shown in Figure 11, after establishing a connection between the visualization layer (client) and the back end (server), this connection will be kept alive continuously while the server sends messages unidirectionally to pass available prescriptions.

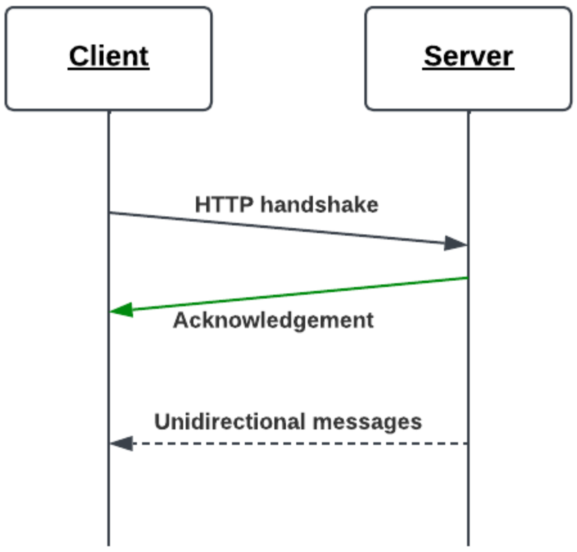


Figure 11. Client and server using SSE.

4.3.5 Event streamer

Considering that the interval for sending data by the system during streaming data simulation is an integer number of seconds, ranging from 1 second to several seconds, the I/O of hard disk and memory is not frequent and the system load is relatively low when executing tasks. In this case, threads can be used directly in the core application instead of processes, and there is no need to run this component in a separate environment. This component uses the `requests` package to directly call API services running locally. For user-uploaded training datasets, 80% of the dataset is used for model training and validation while the remaining 20% is reserved for simulating stream data.

It should be noted that the term "percentage" is being used in reference to the percentage of cases, as opposed to the percentage of events, within this particular context. Hence, there is no reason to be concerned about the inclusion of incomplete cases in the partitioned datasets.

As shown in Figure 12, when sending data to the system, the event streamer sends them according to the occurrence time order of all events in the simulation dataset, rather than grouping them by case. Thus, this component can simulate multiple ongoing cases. When all events in a case have been sent, the streamer attaches an indicator to the data of the last event, representing the end of a case. This flag is also passed on to the consumer of prescription results.

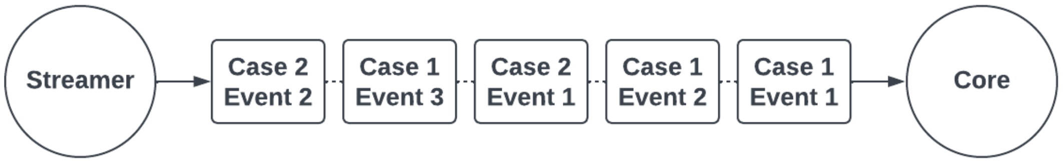


Figure 12. Event streamer sends events continuously to the core application.

4.3.6 Plugin system

This section will introduce the plugin system and the details of the default plugins equipped in the system.

Communication between core and plugins. Each plugin has its own independent runtime environment and runs in a separate Docker container. This is based on the following considerations:

- Different plugins may have different dependencies based on their algorithm implementation. Some dependencies have specific requirements for Python version and package dependencies. Running various plugins in the same environment can lead to dependency conflicts.
- Separating environments and processes will better utilize the CPU's multi-core capabilities.

- Not only are the plugins and core programs independent of each other, but also independent from one another, which improves overall system fault tolerance and availability.

The plugin communicates with the core program through RabbitMQ. There are several types of communication: online reporting, communication related to training models, communication related to obtaining prescription results, and error reporting. Let us now explain each type of communication.

Online reporting: The purpose of the plugin reporting its online status to the core program is to allow the core program to select plugins and determine analysis completion based on the plugin's online status when creating projects, sending data for analysis, and waiting for analysis results. This also improves system fault tolerance. If some plugins go offline due to certain reasons, the core system will not wait endlessly for result returns but automatically handle workflow processing according to their offline status. As shown in Figure 13, the core program sends an `ONLINE_INQUIRY` request to all registered plugins every 5 minutes and updates plugin state records based on `ONLINE_REPORT` response bodies. Additionally, when the core application starts up or a plugin service restarts after an interruption, it will automatically send an online report to update its status.

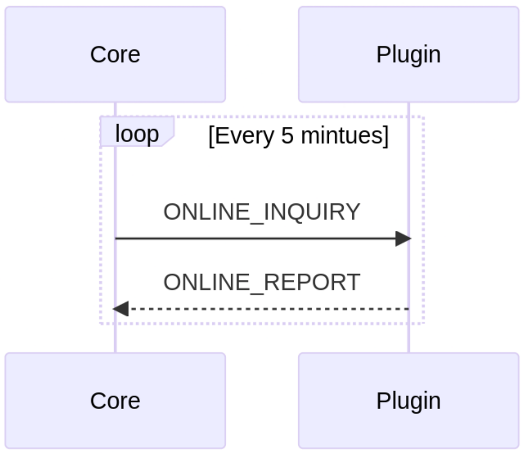


Figure 13. Communication between core and plugin: online reporting.

Communication related to training models: As shown in Figure 14, when a new project is created by an API consumer, the core firstly starts preprocessing the dataset in the background. Once preprocessing is done, the core sends a request asking if this dataset can be shared with the specific plugin. The plugin responds with whether it can work on that dataset or not. Then, the training task is added to the queue. During training, the plugin actively reports back progress information. After training ends, the plugin returns the path where the model file is stored locally so that it can be saved into the database.

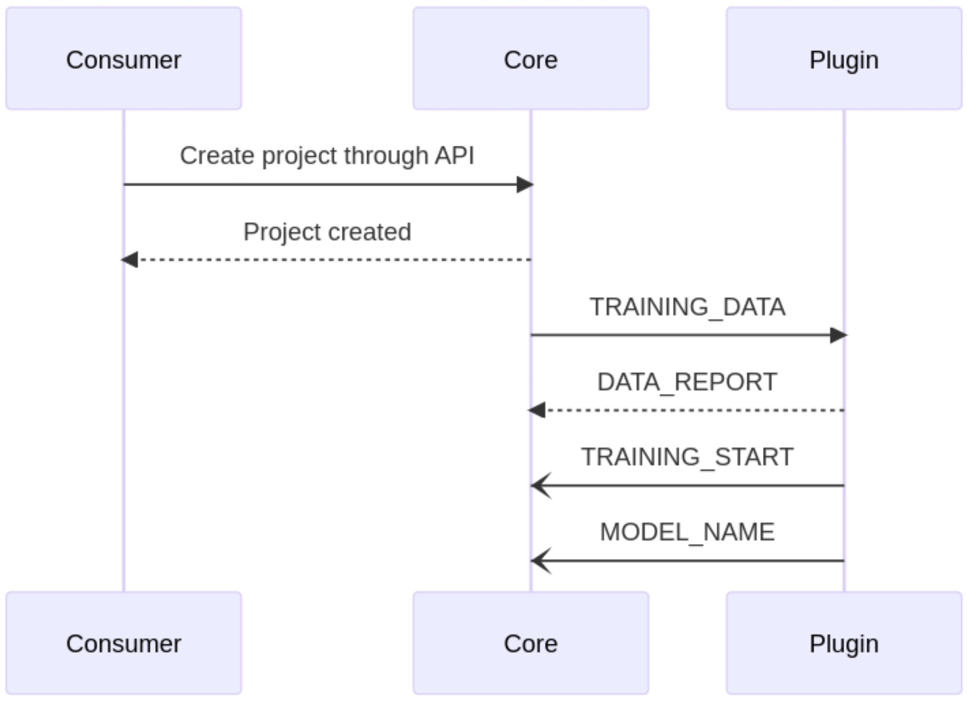


Figure 14. Communication between core and plugin: during training phrase.

Communication related to obtaining prescription results: According to the user's choice, there are two ways to finish this workflow. One way is to prescribe on an ongoing dataset, and the other is to prescribe by receiving streaming data. If the user has uploaded a test dataset that only contains ongoing cases, the workflow is shown in Figure 15. The core will send the uploaded dataset to the plugin, and the plugin will process the data and send the results back to the core. On the other hand, if the project is ready, the API consumer can activate the project to start the prescribing workflow (Figure 16). The activation means that in some cases, the plugin may need to load the model file into memory, especially after the system restarts. This can also help the system save resources. Upon receiving the new event data from the API consumer, the core will forward the data to the plugin, which will process the data and send the results back to the core. This routine will continue until the streaming mode is disabled, and then the plugin will release the resources.

Error reporting: In the event that the plugin encounters an error, the error message

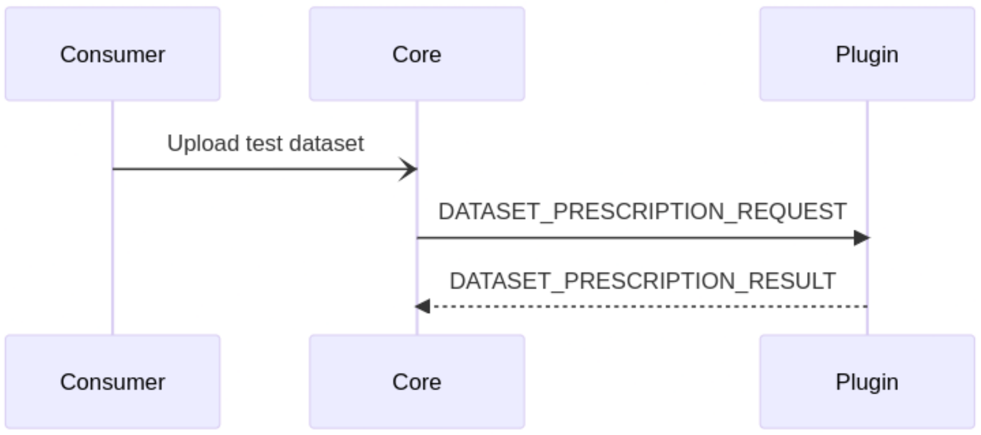


Figure 15. Communication between core and plugin: prescribing by dataset.

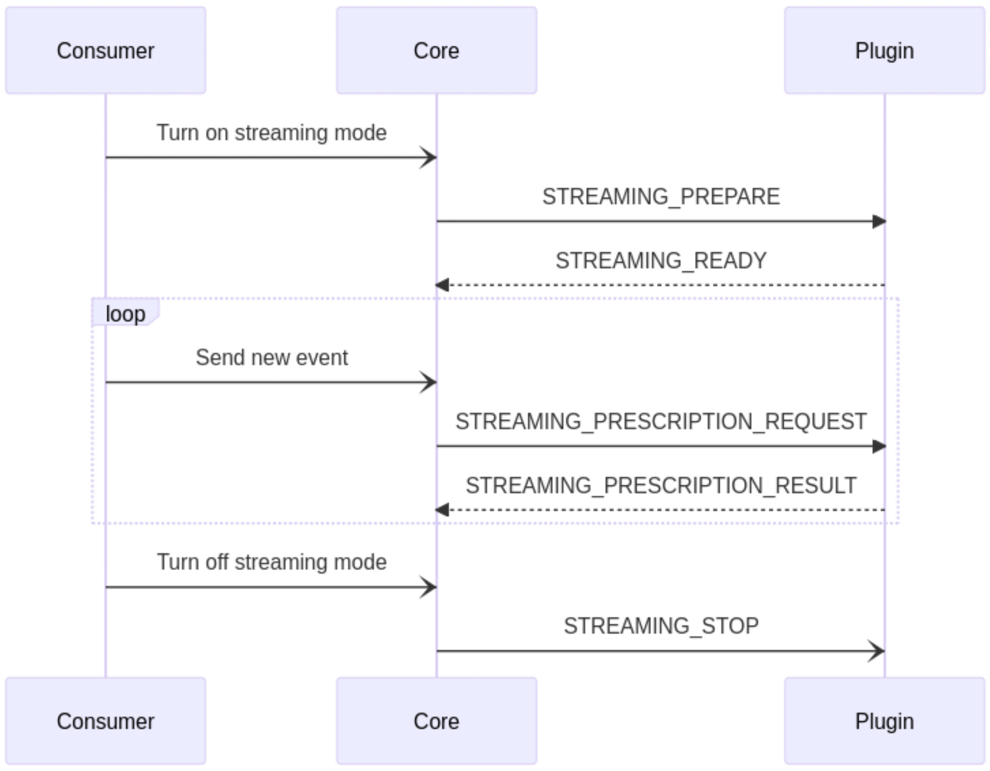


Figure 16. Communication between core and plugin: prescribing by the stream.

will be promptly relayed to the core (Figure 17). Subsequently, the core will update the status of the plugin to reflect the `ERROR` status, with the error message attached to the plugin. This implies that the plugin would be rendered unusable unless certain project definitions are modified by the user. In the event that all plugins have been marked with the `ERROR` status, the core will designate the entire project as being in an error state, necessitating intervention by the user.

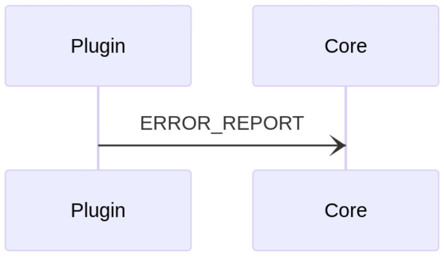


Figure 17. Communication between core and plugin: error reporting.

Default plugins. To ensure the compatibility of different algorithms in terms of preprocessing data, providing information, and aggregating results, we developed four plugins with different algorithms and functions to test and verify design concepts and workflows for the plugin system.

As described in Section 3, the algorithms we used are all based on machine learning, discussed and selected within the research team, and we referred to relevant papers for each algorithm's specific implementation. The selection of algorithms took into account the following points:

- The selected algorithms should cover at least the categories of guiding, correlation-based, and causality-based.
- Since our focus is on testing the integration of different types of algorithms in a toolset rather than researching the algorithms themselves, the implementation of each category's algorithm should not be too complex.
- Ideally, the algorithm has already provided code implementation in its corresponding paper or has an open-source package that can be called upon.

Each plugin necessitates and utilizes three fundamental data types from the event log file for training: Case ID, activity, and timestamp. When encoding data based on activity, this tool uses three widely recognized encoding methods suitable for analyzing event logs with machine learning algorithms, as mentioned by Leontjeva et al.: boolean encoding, frequency-based encoding, and simple index encoding [LCDF$^+$15]. Users can customize each plugin's parameters to select one of these three encoding methods according to their requirements.

Next, we will introduce the basic functions and principles of each plugin as well as the additional data required for training and prediction. We will also briefly describe how adding each plugin contributes to improving functionality.

- Plugin-1: This plugin uses guiding methods to predict possible events in ongoing cases based on historical data as feedback suggestions for users. Le et al. introduced a method using k-nearest neighbors (kNN) to train models based on sequences composed of certain lengths of events from all traces [LGN17]. The outcome is defined as the next activity (if any) in each sequence so that predicting the next activities can be achieved by implementing this method into our plugin. Adding this plug-in verifies the feasibility of providing training data sets, and datasets waiting for predictions while obtaining results through core programs.

- Plugin-2: This plug-in belongs to the correlation-based methods category, which provides probabilities about negative outcomes when an ongoing case reaches its current state. Teinemaa et al. mentioned constructing a probabilistic classifier using the random forest algorithm to obtain the likelihood of undesired outcomes based on partial traces [TTdL$^+$18]. This plugin implements this method and automatically labels each case according to user-defined outcome definitions before providing them for classifier training. The probability obtained through this plugin can be used by the visualization layer to trigger alarms for cases that require intervention based on a threshold set by users. Adding this plug-in helps test and verify the workflow of identifying outcome categories defined by users, and passing them to plugins for correlation-based algorithms.

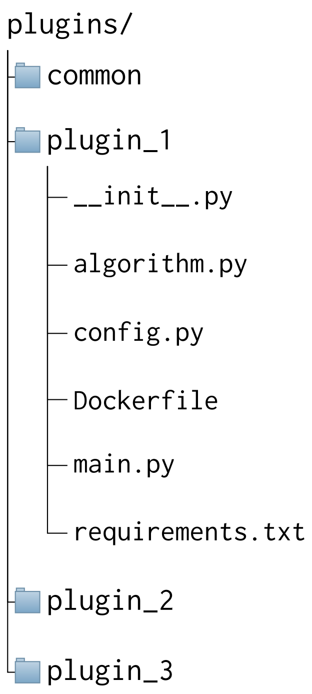
- Plugin-3: This plug-in belongs to the causality-based methods category, which provides conditional average treatment effect (CATE) scores for interventions in ongoing cases. Minami developed CasualLift [17] using machine learning methods proposed by Athey et al. [AI15] and pylift [18] as references. This plugin calls the CausalLift package and uses a two-model approach with historical data and new cases' partial trace data to calculate the CATE score when intervening in a case. The visualization layer can provide intervention suggestions based on the

[17]CasualLift: `https://github.com/Minyus/causallift`
[18]pylift: `https://github.com/rsyi/pylift`

CATE score provided by this plugin depending on user settings at the front end. Adding this plug-in helps test and verify the workflow of identifying outcome and treatment categories defined by users, providing data, and defining treatments passed onto plugins for causality-based algorithms.

- Plugin-4: This plug-in belongs to the causality-based methods category, which considers resource constraints while allocating resources automatically among cases requiring interventions. Shoush et al. proposed a prescriptive monitoring technique that triggers interventions under resource constraints [SD22]. This plugin refers to their method where it allocates resources among cases requiring intervention whose CATE score exceeds a certain threshold under stream mode considering resource constraints. Since this algorithm requires additional information such as an available resources list during runtime along with the time required for executing an intervention, we adjusted the plugin system to support core programs in providing additional data beyond what users have provided during training. Adding this plug-in helps improve the plugin system by enabling it to pass on extra information to meet a wider range of algorithm requirements.

Adding a new plugin. The process of adding a new plugin to the plugin system is straightforward and highly customizable. The system is designed to be extensible, allowing developers to easily integrate new plugins to enhance its functionality. The root directory of the entire tool code contains a `plugins` directory for placing common library code and plugin code for the plugin system. The structure of this directory is shown in Figure 18. For a plugin, only six files need to be created: `__init__.py`, `algorithm.py`, `config.py`, `Dockerfile`, `main.py`, and `requirements.txt`. Except for `/algorithm.py`, the other files are relatively short and provided with common templates, so developers can copy and modify them according to actual situations in just a few minutes. As shown in Figure 19, only one class about a specific algorithm needs to be added in `/algorithm.py` file, which implements four functions. The message broker communication with the core application, model saving/loading, memory resource release, etc., are all handled by the common library of plugins so that developers do not need to pay attention to these contents but focus on implementing algorithms specifically. At the same time, some built-in methods provided by parent class `Algorithm` allow `FooBarAlgorithm` to directly obtain training dataset, user-defined parameters, and additional information required by algorithms as well as models. This allows developers to obtain and save the required data without having to pay attention to the implementation logic of these methods.



```
plugins/
├ common
├ plugin_1
│ ├ __init__.py
│ ├ algorithm.py
│ ├ config.py
│ ├ Dockerfile
│ ├ main.py
│ └ requirements.txt
├ plugin_2
└ plugin_3
```

Figure 18. File structure of plugins folder.

```python
class FooBarAlgorithm(Algorithm):
    def __init__(self, algo_data: Dict[str, Any]):
        super().__init__(algo_data)

    def preprocess(self) -> str:
        # Pre-process the data
        pass

    def train(self) -> str:
        # Train the model
        pass

    def predict(self, prefix: List[dict]) -> dict:
        # Predict the result by using the given prefix
        pass

    def predict_df(self, df: DataFrame) -> dict:
        # Predict the result using a DataFrame
        pass
```

Figure 19. Class definition for an algorithm.

4.4 Deployment

According to the requirements of NFR-5, the tool developed in this article needs to be deployed on a publicly accessible server. To ensure portability, we chose Docker to deploy the service. Docker is a software technology that allows developers to create, deploy, and run applications in containers, which are self-contained environments isolated from the host system and other containers. We used Docker Compose tool to define the required services and ultimately deployed 8 containers on the server [19], including one PostgreSQL database container, one RabbitMQ message broker container, one core application container, one dataset processor container, and four plugin containers.

4.5 Documentation

According to the requirements of NFR-4, corresponding documentation should be provided when delivering this tool, so that front-end developers can integrate it. The documentation [20] is presented in the form of a static website, generated using the Hugo [21] static website generation framework and continuously published using GitHub Actions [22]. The documentation is divided into four main parts that introduce the usage methods for all APIs and necessary internal logic and principles:

1. Getting started: This part introduces the basic methods of using this tool, detailed steps for the complete process, and provides a Postman [23] collection for testing purposes, which includes usage examples for all API endpoints.

2. Workflow: This part provides a detailed introduction to uploading event logs, training projects, sending new event data, and receiving prescription results APIs for a complete basic usage flow. It also covers important considerations and calling methods.

3. Advanced usage: This part introduces some advanced operations including continuous sending of new events to the system, redefining event logs and projects, enabling/disabling plugins, updating plugin parameters and additional information.

4. Development: This part describes the basic development details of the tool and provides a detailed description on how to add new plugins to facilitate plugin development by developers.

[19]API service: `https://prcore.chaos.run`
[20]Documentation: `https://prcore-docs.chaos.run`
[21]Hugo: `https://gohugo.io`
[22]GitHub Actions: `https://github.com/features/actions`
[23]Postman: `https://www.postman.com`

In addition, the document is equipped with a chat assistant specifically designed to answer document-related questions [24]. This document assistant is presented in the form of a web page and implements knowledge-based question-answering using ChatGPT [25] and embeddings [26] technology, developed using Streamlit [27] and Langchain [28].

[24]Documentation assistant: https://prcore-assistant.chaos.run
[25]ChatGPT: https://openai.com/blog/chatgpt
[26]Embeddings: https://platform.openai.com/docs/guides/embeddings
[27]Streamlit: https://streamlit.io
[28]Langchain: https://python.langchain.com/en/latest/index.html

5 Evaluation

This section will demonstrate the process and results of testing for the developed artifacts.

5.1 Evaluation setup

We conducted two types of evaluations: requirements fulfillment evaluation and performance evaluation.

Requirements fulfillment evaluation. We conducted evaluation tests for both functional and non-functional requirements. During these tests, BPIC 2012 dataset was utilized.

For functional requirements, we designed test cases for each requirement based on the functional requirements table. As shown in Table 5, the test case table includes requirement ID, preconditions, and acceptance criteria. The tables for other requirements are displayed in the appendix from Table 8 to Table 23. If all the acceptance criteria are met, then the requirement is deemed to be satisfied. Except for the test cases of FR-17, requests from Postman collection [29] were used to obtain results and compare them with expected behaviors during testing. For FR-17, manual testing was conducted using the docker-compose tool by modifying parameters in `.env` file in order to disable or enable plugins.

Table 5. Test case for FR-1.

Requirement	FR-1
Preconditions	The dataset is prepared in CSV, XES, and ZIP formats.
Acceptance criteria	1. A new event log object is created and returned in the response. 2. The event log ID is returned. 3. The response body must contain the header of all columns, the inferred definition of each column, and the data from the first five rows in the dataset.

For non-functional requirements, we tested NFR-1 and NFR-2 using an automated Python script that called the API to trigger project training and prescribing, while automatically recording time-consuming tasks. The system conditions used for testing were 8 cores, 16 threads CPU (3.3 GHz), and 32GB RAM. To evaluate NFR-3, we used the time taken by the author to add Plugin-4 as a reference.

[29] `https://prcore-docs.chaos.run/getting-started/test-with-postman/`

Performance evaluation. To further evaluate the performance limitations of this artifact, we conducted performance testing. In the performance test, we used task processing time under specified computing resource conditions as the key performance indicator. First, we tested the time consumed for preprocessing and training on different datasets separately. Also, during the training process, all four default plugins are enabled. We used BPIC 2011, BPIC 2012, and BPIC 2017 datasets for testing. Please refer to Table 6 for basic information and the size of these datasets.

Table 6. Datasets information.

	BPIC 2011	**BPIC 2012**	**BPIC 2017**
Number of traces	1143	13087	31509
Number of events	150291	262200	1202267
Events per trace	131.488	20.035	38.156
Min events per trace	1	3	10
Max events per trace	1814	175	180
Number of attributes	128	7	19

There are two modes of data input supported by this tool during the prescribing stage: one is to continuously push new event data to the tool, with only one event included each time. The tool will analyze the case to which the new event belongs and return prescriptions. The other mode is to upload a dataset containing only ongoing cases to the tool. The tool will analyze all cases included in the dataset and return prescriptions for all cases at once. Since the former mode analyzes only a single case each time with small amounts of data, and requires little time for prescribing, it is not meaningful to evaluate its performance in terms of time consumption. Therefore, we chose to use the latter mode of data input for evaluating prescribing performance. Thus, in addition to the time used in training, we generated a test set containing only ongoing cases from projects trained on three BPIC datasets using built-in functionality of this artifact and tested how long it took to prescribing these data sets respectively. The ongoing cases datasets used in these tests account for 20% of the original dataset in terms of the number of cases.

5.2 Evaluation results

In this section, we presented the results of the evaluation and conducted some analysis and discussion on the outcomes.

Requirements fulfillment results. We conducted the testing following the steps mentioned in Section 5.1. As shown in Table 7, after testing, all functional requirements collected during the requirements collecting phase have been evaluated as fulfilled.

Table 7. Fulfillment of functional requirements.

Requirements	**Fulfillment**
Must have	All 8 requirements fulfilled: FR-1, FR-2, FR-3, FR-5, FR-12, FR-14, FR-15, FR-17
Should have	All 6 requirements fulfilled: FR-4, FR-6, FR-7, FR-8, FR-11, FR-13
Could have	All 3 requirements fulfilled: FR-9, FR-10, FR-16

Moreover, we have evaluated or tested the non-functional requirements:

- NFR-1: It took a total of 14.3 seconds to preprocess and train the BPIC-2012 dataset, which meets the requirement.
- NFR-2: We uploaded new simulated data generated from BPIC-2012 based on 20% of already trained projects through BPIC-2012 and waited for processing results. Finally, it took a total of 20.3 seconds for prescribing, which meets the requirement.
- NFR-3: The author spent a total of 90 minutes adding Plugin-4 [30], including testing and debugging of the new plugin. This time can be used as a reference for assessing the difficulty of adding a new plugin.
- NFR-4: As discussed in Section 4.5, we completed the document addition, and therefore, it meets this requirement.
- NFR-5: As described in Section 4.4, we successfully deployed this service; hence, it satisfies this requirement.

Therefore, except for NFR-3, which cannot be accurately quantified, all other non-functional requirements have been met.

Performance results. Regarding the training time for different datasets, the results are shown in Figure 20. BPIC-2012 had the shortest processing time which may be due to fewer attributes included in each event.

[30]From commit a736209 to commit 8dd4457, the duration is about 90 minutes.

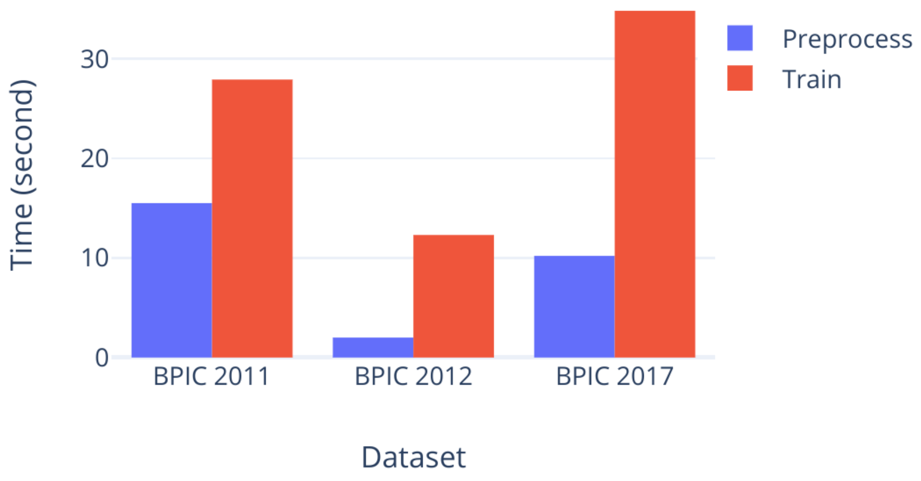


Figure 20. Preprocessing and training time used on BPIC datasets.

Regarding the prescribing time for different ongoing cases datasets, the results are shown in Figure 21. It can be seen that BPIC-2011 took the longest but still within 60 seconds.

Furthermore, to identify reasons for performance bottlenecks, we enabled and disabled different plugins one by one and performed the prescribing tests again using only the dataset with the highest processing time (BPIC 2011) mentioned above. We then recorded the prescribing times of different plugins against the same test dataset. It is worth noting that the function of Plugin-4 is only applicable when operating under stream data mode, so it was not included in this round of tests. As shown in Figure 22, Plugin-3 consumed three times more time compared to Plugin-1. After analyzing the time consumption of different key nodes in this plugin, we found that the performance limitation was caused by an external package called CausalLift, which was referenced by this algorithm and had nothing to do with the core program or plugin system itself. Additionally, the impact of CausalLift on performance is minimal under stream data mode because there is a certain time interval for external programs to push new events of the same project into the system, and it is not expected to have high frequency under normal usage scenarios. Moreover, our system processes requests from different projects in parallel, so CausalLift has less impact on performance issues under stream data mode.

According to the acceptance criteria mentioned in Section 3.3, all criteria have been met.

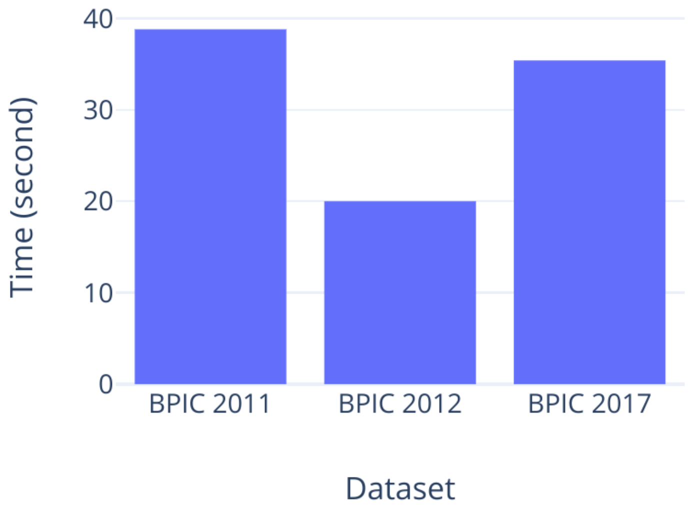


Figure 21. Prescribing time used on different BPIC datasets.

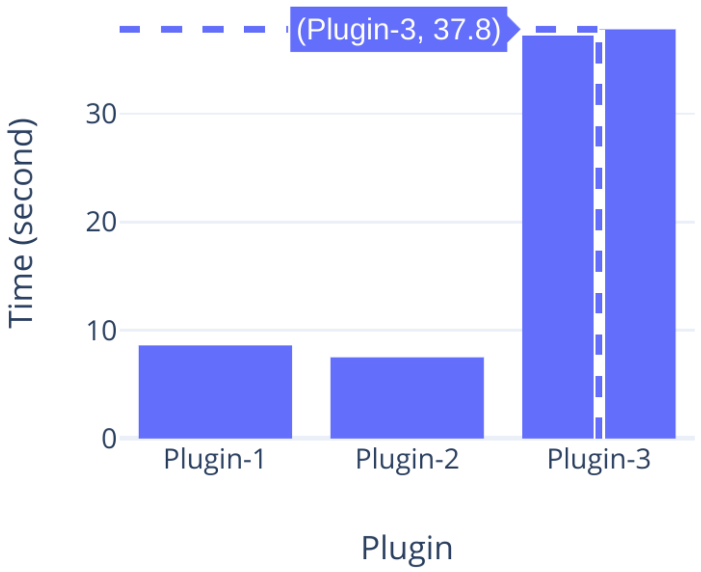


Figure 22. Prescribing time used on BPIC 2011 dataset by different plugins

6 Conclusion

The thesis demonstrates the development and usage of Kairos' backend, a versatile and customizable tool that can train models using business event log files to provide predictive and prescriptive recommendations for ongoing cases in various domains. By utilizing the tool on Kairos, the user is capable to upload their files, define columns, create projects, and set parameters to effectively generate results for their business event logs. Furthermore, the developed artifact offers key features such as plugin integration, enabling users to enhance the functionality with their custom plugins or modify existing plugins to suit their needs. The provision for redefining definitions, re-uploading files, and the support to carry out various project-related tasks, such as project deletion and retrieval, display the flexibility and adaptability of the tool. For the getting prescriptions phase, the tool allows for the submission of new data for live prescriptions on ongoing cases, providing two methods for uploading new events data and obtaining results.

However, there are still some limitations to this tool. These limitations include the risk of incomplete requirements collection, the risk that the adopted architecture may not be compatible with the processing of certain special event logs, and we have only tested and evaluated it using BPIC datasets without fully exploring the problems that real data from other fields may bring. Although it passed the functional tests using the BPIC dataset, the system did not evaluate the prediction accuracy of utilizing real data processed results with outcome and treatment definitions that conform to real domain situations under the testing of domain experts. However, the algorithm or package used by the plugin has been peer-reviewed, so its accuracy can be guaranteed to a certain extent. It should also be noted that the core value of the entire artifact is as a unified interface and aggregator, while plugins themselves are only for verifying design concepts and can be added, replaced, or deleted as needed.

In future work, more plugins can be added to expand functionality. By adding different types of plugins, adjustments can be made to the core program and plugin system to make them more universal and compatible, meeting the operational requirements of various algorithms. At the same time, existing plugin performance can also be optimized by replacing packages such as CasualLift. Also, cloud-based file and model storage can also be supported to enable distributed deployment, making the system more scalable. In addition, support for more types of APIs in data uploading and result retrieval can be provided. For example, supporting Kafka usage will meet the needs of more different systems.

Overall, the development as part of this thesis showcases a flexible tool with an easy-to-use API design and integration capabilities, enabling users to access valuable predictive analytics and prescriptive recommendations to optimize cases' performance. Future research and development efforts can further enhance the tool's functionality and capabilities by adding more plugins, ensuring its continued relevance and applicability across a wide range of industries and domains.

References

[Aal16] Wil van der Aalst. Process mining: The missing link. In *Process Mining*, pages 25–52. Springer, 2016.

[AAM$^+$11] Wil van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, et al. Process mining manifesto. In *International conference on business process management*, pages 169–194. Springer, 2011.

[AHAE18] Nada Mohammed Abdulhameed, Iman MA Helal, Ahmed Awad, and Ehab Ezat. A resource recommendation approach based on co-working history. *International Journal of Advanced Computer Science and Applications*, 9(7), 2018.

[AI15] Susan Athey and Guido W Imbens. Machine learning methods for estimating heterogeneous causal effects. *stat*, 1050(5):1–26, 2015.

[AMGS16] Michael Arias, Jorge Munoz-Gama, and Marcos Sepúlveda. A multi-criteria approach for team recommendation. In *International Conference on Business Process Management*, pages 384–396. Springer, 2016.

[BKD$^+$22] Zahra Dasht Bozorgi, Aleksei Kopõlov, Marlon Dumas, Marcello La Rosa, and Artem Polyvyanyy. Prolift: Automated discovery of causal treatment rules from event logs. In *2022 International Conference on Process Mining (ICPM)*, 2022.

[BTD$^+$20] Zahra Dasht Bozorgi, Irene Teinemaa, Marlon Dumas, Marcello La Rosa, and Artem Polyvyanyy. Process mining meets causal machine learning: Discovering causal rules from event logs. In *2020 2nd International Conference on Process Mining (ICPM)*, pages 129–136. IEEE, 2020.

[DFGMM18] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, and Fredrik Milani. Predictive process monitoring methods: Which one suits me best? In *Business Process Management: 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9–14, 2018, Proceedings 16*, pages 462–479. Springer, 2018.

[DLRMAR13] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A Reijers. *Fundamentals of business process management*. Springer, 2013.

[FPTT$^+$22] Stephan A Fahrenkrog-Petersen, Niek Tax, Irene Teinemaa, Marlon Dumas, Massimiliano de Leoni, Fabrizio Maria Maggi, and Matthias

Weidlich. Fire now, fire later: alarm-based systems for prescriptive process monitoring. *Knowledge and Information Systems*, 64(2):559–587, 2022.

[KMND22] Kateryna Kubrak, Fredrik Milani, Alexander Nolte, and Marlon Dumas. Prescriptive process monitoring: Quo vadis? *PeerJ Computer Science*, 8:e1097, 2022.

[KMND23] K. Kubrak, F. Milani, A. Nolte, and M. Dumas. Design and evaluation of a user interface concept for prescriptive process monitoring. In *International Conference on Advanced Information Systems Engineering*, 2023. (to appear).

[LCDF$^+$15] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. Complex symbolic sequence encodings for predictive monitoring of business processes. In *Business Process Management: 13th International Conference, BPM 2015, Innsbruck, Austria, August 31–September 3, 2015, Proceedings 13*, pages 297–313. Springer, 2015.

[LGN17] Mai Le, Bogdan Gabrys, and Detlef Nauck. A hybrid model for business process event and outcome prediction. *Expert Systems*, 34(5):e12079, 2017.

[PTRC07] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[RDFGM22] Williams Rizzi, Chiara Di Francescomarino, Chiara Ghidini, and Fabrizio Maria Maggi. Nirdizati: an advanced predictive process monitoring toolkit. *arXiv preprint arXiv:2210.09688*, 2022.

[RSF$^+$19] Williams Rizzi, Luca Simonetto, Chiara Di Francescomarino, Chiara Ghidini, Tõnis Kasekamp, and Fabrizio Maria Maggi. Nirdizati 2.0: New features and redesigned backend. In *Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019 co-located with 17th International Conference on Business Process Management, BPM 2019, Vienna, Austria, September 1-6, 2019*, pages 154–158, 2019.

[SD22] Mahmoud Shoush and Marlon Dumas. Prescriptive process monitoring under resource constraints: a causal inference approach. In *Process Mining Workshops: ICPM 2021 International Workshops, Eindhoven, The*

Netherlands, October 31–November 4, 2021, Revised Selected Papers, pages 180–193. Springer, 2022.

[TDRM19] Irene Teinemaa, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. Outcome-oriented predictive process monitoring: Review and benchmark. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(2):1–57, 2019.

[TTdL$^+$18] Irene Teinemaa, Niek Tax, Massimiliano de Leoni, Marlon Dumas, and Fabrizio Maria Maggi. Alarm-based prescriptive process monitoring. In *Business Process Management Forum: BPM Forum 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings 16*, pages 91–107. Springer, 2018.

[VDA12] Wil Van Der Aalst. Process mining. *Communications of the ACM*, 55(8):76–83, 2012.

[Ver18] Ilya Verenich. *Explainable predictive monitoring of temporal measures of business processes*. PhD thesis, Queensland University of Technology, 2018.

[WDZM20] Sven Weinzierl, Sebastian Dunzer, Sandra Zilker, and Martin Matzner. Prescriptive business process monitoring for recommending next best actions. In *International conference on business process management*, pages 193–209. Springer, 2020.

[WRRM08] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change patterns and change support features–enhancing flexibility in process-aware information systems. *Data & knowledge engineering*, 66(3):438–466, 2008.

Appendix

I. Test reports

Table 8. Test case for FR-2.

Requirement	FR-2
Preconditions	The event log has been uploaded and the created object was successfully returned.
Acceptance criteria	1. Return a list of all activities that appear in the log, including a count of how many times each activity appears. 2. If the `RESOURCE` column is defined, return a list of all resources and include a count of how many times each resource appears. 3. Return a list of column names that can be used to define outcome and treatment.

Table 9. Test case for FR-3.

Requirement	FR-3
Preconditions	The event log has been uploaded and the column definition has been successfully configured.
Acceptance criteria	1. A project object has been successfully created and returned in the response body. 2. The response body contains information such as the status of the project, associated event log, list of plugins, etc. 3. If test event log is uploaded along with the upload of regular logs, then result_key must not be `null`.

Table 10. Test case for FR-4.

Requirement	FR-4
Preconditions	The project has been created.
Acceptance criteria	1. The API belonging to this function can be called at any time, regardless of the project's status. 2. The name and description of the project have been successfully updated as requested.

Table 11. Test case for FR-5.

Requirement	FR-5
Preconditions	The project has been created and the event log has completed pre-processing. Column definitions for event logs, as well as outcome and treatment definitions, have been provided.
Acceptance criteria	1. Can access the project and view the plugins that are automatically added to it. 2. Can check the training status of each plugin. 3. All plugins will eventually reach a state of training completion.

Table 12. Test case for FR-6.

Requirement	FR-6
Preconditions	The plugin object has been created and its status is TRAINED.
Acceptance criteria	1. Can modify the parameters of the plugin. 2. If retraining is required after modifying the parameters, the plugin will automatically start the training process and complete it successfully.

Table 13. Test case for FR-7.

Requirement	FR-7
Preconditions	The plugin object has been created and its status is TRAINED.
Acceptance criteria	1. Can modify the additional information of the plugin. 2. If retraining is required after modifying the additional information, the plugin will automatically start training and complete it successfully.

Table 14. Test case for FR-8.

Requirement	FR-8
Preconditions	The plugin object has been created and its status is TRAINED.
Acceptance criteria	1. It is possible to successfully enable or disable plugins under a certain project. 2. Disabled plugins will not participate in the prescribing activities of that project.

Table 15. Test case for FR-9.

Requirement	FR-9
Preconditions	The project has been created and all plugins have been trained. The new definitions for outcome and treatment are valid for the event log.
Acceptance criteria	1. The relevant definition can be successfully modified, and the updated project object is returned. 2. The project automatically starts retraining and successfully completes training.

Table 16. Test case for FR-10.

Requirement	FR-10
Preconditions	The project has been created and all plugins have been trained. The new event log file to be uploaded contains all the columns from the old file.
Acceptance criteria	1. The Event log object has been updated and the updated data has been successfully returned. 2. A list of all activities appearing in the log is returned, including a count of how many times each activity appears. 3. If the `RESOURCE` column is defined, return a list of all resources and include a count of how many times each resource appears. 4. Return a list of column names that can define outcome and treatment.

Table 17. Test case for FR-11.

Requirement	FR-11
Preconditions	The project has been created.
Acceptance criteria	The project has been successfully deleted, and the associated event log has also been deleted at the same time.

Table 18. Test case for FR-12.

Requirement	FR-12
Preconditions	Several projects have already been created.
Acceptance criteria	Return a list of projects in pagination style, which allows viewing other pages of the projects list based on path parameters.

Table 19. Test case for FR-13.

Requirement	FR-13
Preconditions	The project has been created and its status is TRAINED.
Acceptance criteria	1. After uploading a new dataset, the `result_key` was returned. 2. The prescription results for all cases in the dataset can be obtained through the `result_key`.

Table 20. Test case for FR-14.

Requirement	FR-14
Preconditions	The project has been created and its status is TRAINED.
Acceptance criteria	1. New event data can be successfully pushed to the system. 2. When the number of events in a case exceeds a certain amount, prescription results messages can be received through SSE endpoints. 3. Prescription results for multiple plugins can be included in the result message.

Table 21. Test case for FR-15.

Requirement	FR-15
Preconditions	The project has been created and its status is TRAINED.
Acceptance criteria	1. Can enable simulation mode by calling the API and receive a response indicating success. 2. Can receive messages containing simulated data prescription results at SSE endpoints.

Table 22. Test case for FR-16.

Requirement	FR-16
Preconditions	The project has been created and its status is at least PROCESSED.
Acceptance criteria	1. Preprocessed datasets can be downloaded through the API. 2. Test datasets containing only ongoing cases can be generated and downloaded. 3. Original log files can be downloaded. 4. Datasets for conducting data stream simulations can be downloaded.

Table 23. Test case for FR-17.

Requirement	FR-17
Preconditions	None
Acceptance criteria	1. Can use environment variables to decide whether to globally load certain plugins when the core program starts. 2. Globally enabled plugins can be queried through APIs. 3. Globally disabled plugins are not available in the list of plugins provided by the API.

II. Figures and tables

Table 24. API endpoints

Method	**Endpoint**	**Description**
POST	`/event_log`	Upload an event log file
GET	`/event_log/{event_log_id}`	Get an existing event log object
PUT	`/event_log/{event_log_id}`	Define columns of the event log by ID
PUT	`/event_log/{event_log_id}` `/upload`	Re-upload the event log file, but keep all existing definitions
GET	`/event_log/all`	Get all uploaded event logs
GET	`/event_log/{event_log_id}` `/definition`	Get the definition of the event log by ID
GET	`/plugin/all`	Get all created plugin objects
GET	`/plugin/available`	Get all available online plugin containers
GET	`/plugin/{plugin_id}`	Get plugin information by ID
PUT	`/plugin/{plugin_id}`	Set parameters and additional information for the specified plugin object
PUT	`/plugin/{plugin_id}` `/{trigger_type}`	Disable or enable plugin object by ID
POST	`/project`	Create a new project based on uploaded event log
GET	`/project/all`	Get all existing projects
GET	`/project/{project_id}`	Get an existing project by ID
PUT	`/project/{project_id}`	Update a project's name and description
DELETE	`/project/{project_id}`	Delete an existing project
PUT	`/project/{project_id}` `/definition`	Update the project definition by project ID

Continuation of Table 24		
Method	**Endpoint**	**Description**

POST	`/project/{project_id}`
	`/result`
		Upload an new ongoing dataset to a trained project to start prescribing

GET	`/project/{project_id}`
	`/result/{result_key}`
		Get the prescription result from a uploaded dataset

PUT	`/project/{project_id}`
	`/stream/start`
	`/{streaming_type}`
		Start the project's stream mode by ID and streaming type

PUT	`/project/{project_id}`
	`/stream/start`
		Start the project's stream mode using default streaming type

PUT	`/project/{project_id}`
	`/stream/stop`
		Disable the stream mode

PUT	`/project/{project_id}`
	`/stream/clear`
		Disable the stream mode and clear all streamed data

POST	`/project/{project_id}`
	`/stream/event`
		Push a new event to a trained project under streaming mode

GET	`/project/{project_id}`
	`dataset/{dataset_type}`
		Download project datasets by dataset type

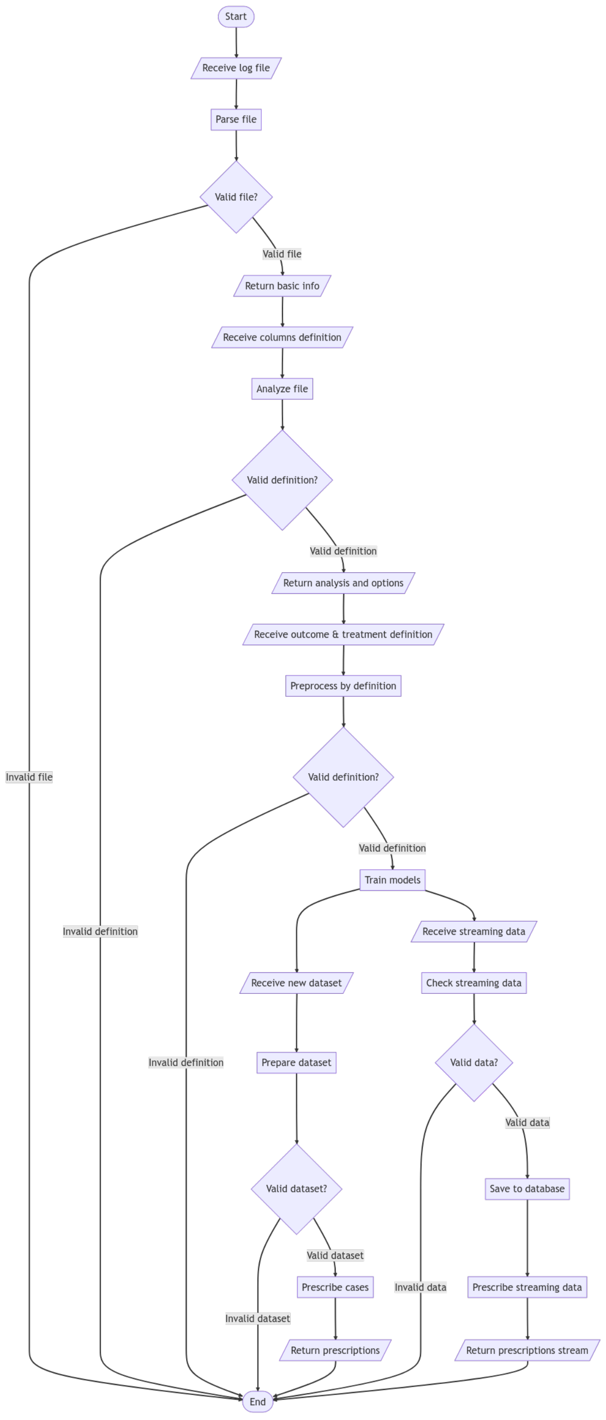


Figure 23. Flowchart of the core application.

III. Resource links

Here we list the resource links for the artifact and related documents developed in this thesis, including web pages and code repositories.

Backend code repository: `https://github.com/prcore/prcore`

Documentation: `https://prcore-docs.chaos.run`

Public demo API: `https://prcore.chaos.run`

API Swagger UI: `https://prcore.chaos.run/docs`

Documentation AI-assistant: `https://prcore-assistant.chaos.run`

Documentation AI-assistant code repository: `https://github.com/prcore/assistant`

IV. Licence

Non-exclusive licence to reproduce thesis and make thesis public