TECHNISCHE UNIVERSITÄT BERLIN
Faculty IV
Institute for Information Systems and Quantitative Methods
Chair of Software and Business Engineering

Einsteinufer 17
10587 Berlin

https://www.sbe.tu-berlin.de/

Project Paper

Human In The Loop BPI with Reinforcement Learning

March 24, 2022

By:
Diana Baumann
Timo Großmann
Aaron Kurz
Bennet Santelmann
Omar Sherif
Jiefu Zhu
Module: Advanced Distributed Systems Prototyping

Contents

1 Motivation

Business processes are at the core of how organizations create value. Business process improvements (BPI) can increase the effectiveness and efficiency of the organizations executing the processes, improve customer satisfaction, and can even lead to favorable social or environmental outcomes [1][2][3]. They are, thus, a central part of business process management (BPM), which is defined as "the art and science of overseeing how work is performed in an organization to ensure consistent outcomes and to take advantage of improvement opportunities" [1].

While both radical, as well as evolutionary process change methods, are used in practice, evolutionary improvements offer the benefit that they are often smaller and easier to implement [4]. Virtually all traditional evolutionary BPM lifecycle approaches first analyze the current process for problems, design solutions for these problems, and replace the old process with the new, supposedly improved version. The new version is then monitored in the production system and, if problems arise, improved again [1]. This approach is not only cumbersome but also means that the two processes versions are not compared in a controlled manner since only one of them is deployed at a given time. It thus remains unclear whether performance changes are caused by process changes or changes in environmental factors.

To tackle these problems, Satyal proposes the AB-BPM methodology [4]. This methodology suggests the usage of DevOps principles, like continuous integration and dynamic AB Testing using reinforcement learning (RL), in an attempt to improve upon the aforementioned, traditional BPM lifecycle. In the proposal, the old and new process versions are deployed on a business process management system (BPMS) simultaneously, to allow for a fair comparison. This idea of AB-BPM is derived from the AB Testing methodology, where two versions of a website are shown to different users to compare their performance [5]. Using RL, process instantiation requests are routed dynamically to either one of the versions.

The RL algorithm used in AB-BPM belongs to the category of multi-armed bandit algorithms. Such an algorithm tries to make the optimal choice among a set of options as fast as possible, while also making sure that the choice is sensible and not rashly made [6]. In essence, the goal is a fair and fast comparison in a production environment, in contrast to the traditionally lengthy and costly manual analysis that can lead to do-overs

in case of unsatisfying results. Validating process improvements like this, according to [4], is in line with the well-known mantra 'fail fast, fail cheap'.

While this methodology has many benefits, there is one major potential downside: Giving an RL agent full control during these experiments is risky and makes the approach hard to use in real-world scenarios. Imagine a situation in which the RL agent does not take a certain aspect of process performance into account and the new variant falls short in this area. A situation could arise, in which many instances, and thereby real customers would be exposed to an inferior version.

That is why it could be beneficial to give the process expert in charge of the BPI effort more control in the RL-driven AB Testing of new process variants. We call this approach Human-in-the-Loop-AB-BPM methodology (HITL-AB-BPM). In this work, we present an implementation of a prototype realizing this methodology. The prototype allows for dynamic routing of process instantiation requests to the old or new process version and introduces several functionalities that enable a more controlled experiment by enhancing the assessment of variants by considering human expert feedback. We also extended the RL to include a more granular routing based on the customer category, by using a contextual multi-armed bandit (CB) algorithm. This allows the process expert to make assessments and manage risk based on the different customer categories interacting with the process.

In the following Section 2, we will give an overview of the relevant theoretical concepts of RL and AB Testing. Afterward, in Section 3, we will go into more detail about how the prototype works and what functionalities it offers to realize the aforementioned idea behind HITL-AB-BPM. We will then go on to outline the implementation and architecture of the prototype, in Section 4. Section 5 will present an evaluation of the prototype, presenting exemplary results that the usage of the tool can yield. Finally, we will discuss our contribution, limitations, and further research opportunities in Section 6.

2 Background

To better understand the presented implementation of the prototype based on the HITL-AB-BPM methodology, it is helpful to have a basic understanding of the topics of AB Testing and RL. Thus, in this chapter, we will give a brief introduction to the concepts

of AB Testing and RL, and more specifically, CB.

2.1 AB Testing

The basic idea behind AB Testing is to evaluate ideas quickly using controlled experiments. The key question of controlled experiments is, whether a specific newly introduced change will improve key metrics. Therefore, randomized experiments in a production environment are conducted to test two versions (A vs B). Such online testing can often reveal shortcomings such as bugs, inefficiencies, or poor performance. The new version is usually released to a small number of users, limiting potentially negative effects. If the newer version does not show signs of concern and predefined metrics look good, the portion is slowly increased. Such experiments are used to introduce and test micro-changes in web applications, user interfaces, recommender systems, or search ads. Companies, such as Amazon and Facebook use thousands of online controlled experiments to test micro changes. AB Testing is considered an indispensable tool, especially with agile development [5][7][8].

In the context of BPI, Satyal [4] proposed that AB Testing could be used to test a new and presumably 'improved' version of a business process in a fair manner, since both variants would be deployed and tested simultaneously. Client requests for that process would be split between the two variants. The results of the online experiment could be evaluated and key metrics computed and compared. If it turns out that the newer version of the business process would not be an improvement, the process could be further adjusted and tested again, before retiring the previous version.

2.2 Reinforcement Learning

RL is a type of machine learning technique where situations are mapped to actions. An agent must discover which actions return the highest reward by trial and error. The agent learns in an interactive environment over time. It receives a reward as feedback. The goal is to maximize the total cumulative reward.

In contrast to supervised learning, an RL agent is provided with the set of actions of which the agent chooses one. The action is performed in the environment, and the agent receives feedback on how good or bad the chosen action was in form of the reward. After that, the agent is transitioned into the next state. The agent learns, over many iterations,

the best possible action and aims to maximize the reward [6]. Figure 1 illustrates the action-reward feedback loop.

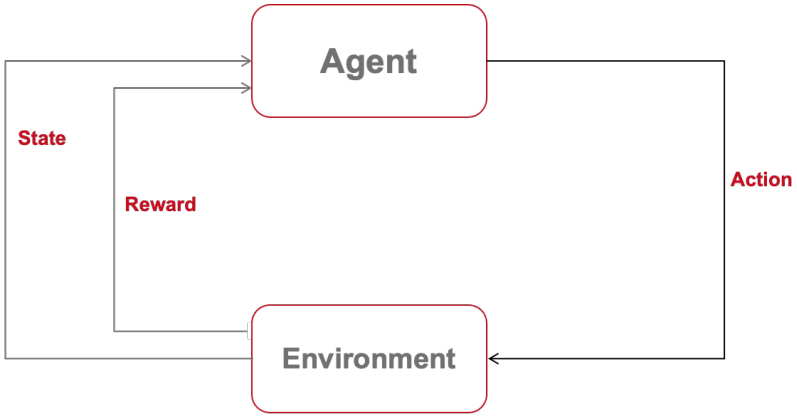


Figure 1: Action-Reward Feedback Loop [6]

One of the biggest challenges in RL is the trade-off between exploitation and exploration. Exploiting means selecting already tried and proven actions to produce high rewards. Exploration is set to discover actions that might yield higher rewards in the future. The agent should progressively favor those actions that yield the highest rewards.

2.2.1 Contextual Multi-armed Bandits

CBs can be understood as an extension of multi-armed bandits. In the underlying learning problem of multi-armed bandits, the agent has to choose from a set of actions. After the agent's decision, it receives a numeric reward depending on the action the agent selected. The goal is, as in general RL, to maximize the reward over time. Each action or arm has a mean reward. This reward is hidden from the agent, otherwise, the problem would be trivial to solve. Even though the reward values are not known, it is possible to estimate them over time. Based on these estimates, at any given time, an action with the highest estimated reward value exists.

The trade-off between exploitation and exploration applies to multi-armed bandits as well, where exploiting current knowledge means selecting the action/s with the highest estimated reward value/s. Exploration aims to not select the actions that yield the highest reward values at a specific timestamp but to maximize the overall reward in the long term by exploring the action space.

Furthermore, a CB algorithm chooses in a sequence of independent trials, actions based on a given context from a set of known actions [9]. The goal is to learn a policy

that maps situations with actions that are best in those situations to maximize the reward over time. Such a problem can be understood as an associative search in which the agent searches for the best actions given an associated situation in which they yield the highest rewards [6].

In the context of AB Testing, the action space consists of two different variants of the business process. Over time the highest yielding reward value for each action is estimated based on a given context.

3 Functionality of the Prototype

To manage risks and make the concept of live testing process improvement hypotheses more viable for real-world usage, we have added a set of functionalities that try to strike a balance between optimizing the exploration-exploitation-split using RL and allowing for human control by a process expert. These functionalities can be seen as one of the core contributions of this work, since they are the mechanisms we propose as part of the HITL-AB-BPM methodology, to iterate upon some of the open issues of the AB-BPM methodology. Note that the user of our product is a process expert who is in charge of a business process improvement effort. The terms user, process expert, or human expert are used interchangeably.

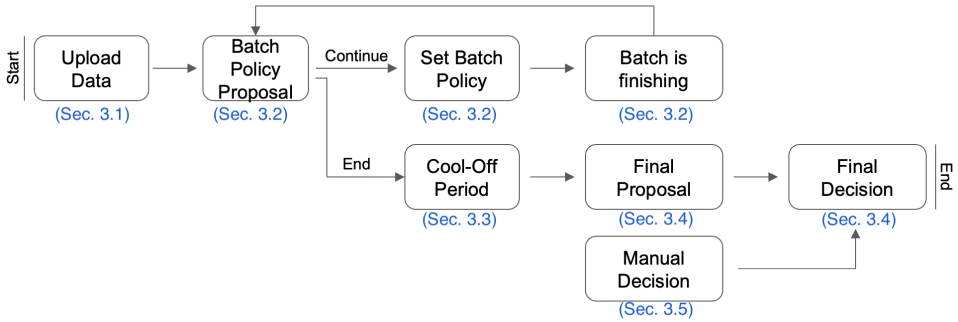


Figure 2: Conceptual Flow of the App

Figure 2 illustrates a broad overview of the flow of the application. At first, the user will upload relevant process data to start the experiment. Afterward, there are several sub-experiments, which are, thanks to custom batch policies in full control of the user. At the same time, the user is guided by the recommendations of the RL agent, through so-called batch policy proposals. Once the user decides to end the experiment, the cool-off period starts, during which long-running instances are evaluated. Finally, the user can

make a final decision on which process variant is preferential, with the help of the data gathered during the experiment. There is also the option to cut the experiment short at any time, with a manual choice by the user. In the following, we will introduce these topics in more detail. To make the context of these detailed explanations clear, we have added bold text that corresponds to the steps seen in Figure 2 throughout this section. Furthermore, we have added section stamps to the figure, referencing where in this paper the functionalities are described.

3.1 Upload Data

Before the experiment can start, the user has to **upload data** about the process which is the subject of the improvement efforts. This upload of data contains BPMN files for versions A and B of the process, the specification whether A or B is the old version (also referred to as default version), the name of the process, and historical data about the old process version. The execution data about the old process is used for the evaluation of new instances that are part of the experiment. For more info on the reward function and the RL algorithm refer to Section 4.1.

The process variants are deployed to and executed on the process execution engine of the BPMS Camunda[1]. A BPMS is a process-aware information system that uses an explicit description of a process model in the form of a BPMN model. A main part of a BPMS is the execution engine, which allows for the creation of executable process instances, the distribution of work to resources, and the management of related data. Altogether a BPMS is a platform to execute processes that tries to coordinate the processes of an organization in a way that will lead to units of work being done at the right time by the right resource [1]. Throughout the experiment, the prototype will then poll the REST API of the Camunda engine for the performance metrics necessary for the evaluation of instances. In the case of this prototype, we only consider the process performance indicator instance duration. More information about how the prototype uses the Camunda engine can be found in Section 4.

[1] `https://camunda.com/products/camunda-platform/bpmn-engine/`, last accessed 2022-03-19

3.2 Batching

To make the experiment more controllable and allow the human to react, we have split up the experiment into multiple batches. The beginning of a batch is marked by the user setting a batch policy and ends after a certain number of instantiation requests have been routed. This number, called batch size, is also set by the user along with the batch policy. A batch policy mainly contains the routing decision the instance router will follow during a certain batch. E.g. if we have the two customer categories public and government and two process versions, A and B, the routing probabilities might specify that 80% of instantiation requests for government customers should be routed to version A and only 20% to version B and the other way around for public customers. One potential reason for such a decision could be the minimization of exposure risk to a not yet established process version for the more important government customer categories.

At the beginning of the experiment, after the upload of the necessary process data, the user has to **review** the first **batch policy proposal**. A batch policy proposal is the suggestion of the RL agent on how the next instances should be routed to most efficiently explore and then exploit the knowledge about which process version is preferential. The first batch policy proposal that is presented to the user is a naive one, suggesting the same probability of routing for each version in every customer category context since there is no data yet with which the RL agent could have been trained. The user then has the option to modify that batch policy proposal, they can set the batch size and adjust the routing probabilities. After they then **set** the first **batch policy**, the next instances are then routed according to this policy, up **until the batch size is reached**. Every process instance that is started as part of a batch is a so-called experimental instance since their performance is then analyzed and taken into account by the RL agent and in other analytical capabilities of the tool. Instantiation requests that are coming in between batch policies (when the user has to check the new batch policy proposal), or before the first batch policy, are not part of the experiment and are routed to the default version.

After one batch policy is done, meaning all instances of that batch policy have been started, the process engine is polled for information about the already finished instances and the RL agent is trained with that info. The agent then creates a new batch policy proposal for the user to review. The user interface (UI) of such a proposal can be seen in Figure 3.

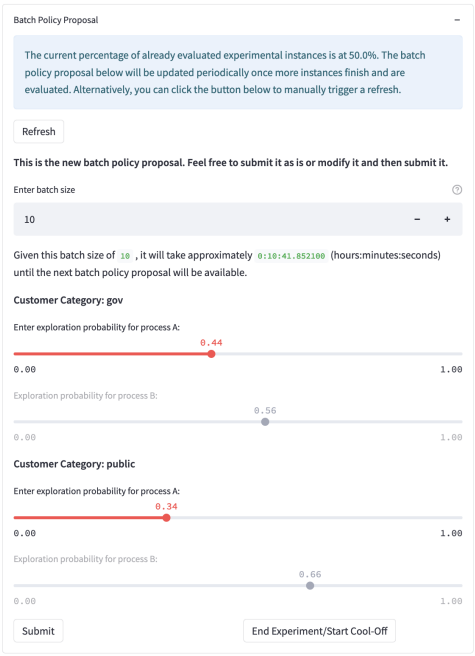


Figure 3: Batch Policy Proposal Presentation to User

The user then has some options. Firstly, they have the temporal choice of either acting immediately or waiting. If they act immediately, maybe not all previous experimental instances have finished, due to the long-running nature of many business processes [4]. This means that the learning and therefore also the batch policy proposal could be incomplete and therefore misguiding. Another option is to wait. The software will periodically re-poll the process engine for more finished instances and update the latest batch policy accordingly. Alternatively, the user can also click a button to manually re-trigger the re-polling of the process engine and the learning on the new data. To make this conflict between a fast experiment and a decision based on complete data more obvious, the user will see a message above the batch policy proposal which alerts them of these possibilities, along with the current evaluation percentage. This percentage shows how many of the experimental instances that have been started have already been evaluated and taken into account for the batch policy proposal.

No matter whether they decide to wait for more data or act immediately, the user has essentially two choices when they review the latest batch policy proposal:

- **continue** with another experimental batch:
  - accept the proposal and set it as batch policy,
  - modify the proposal and set batch policy;

- **end** the experiment and start cool-off.

When they decide to continue the experiment, they have the option to either accept the proposal presented by the RL agent or modify it before setting the new batch policy. This means that another batch of incoming process instantiation requests is then routed according to this batch policy and taken into account for the RL agent and the analysis. This loop of reviewing the batch policy proposal, setting the batch policy, and running and learning from the new batch can be repeated as long as the user wishes. If they decide, however, to end the experiment, the cool-off period starts.

3.3 Cool-Off Period

The **cool-off period** is another mechanism that is introduced to deal with long-running instances. At the time the user decides to end the experiment, there might still be experimental instances that have not finished yet and are therefore not part of the analysis and recommendation of the RL agent. It would be advantageous for the final decision to have information about all experimental instances, so we give the user two options: they can either wait for all the instances to finish and make a final decision or cut the experiment short and make a manual decision before all instances are finished (more on the manual decision in Section 3.5). To give the user more clarity about this consideration, we also display the information about the current evaluation progress in the UI of the cool-off period (as seen in Figure 4). The interface area with information about the cool-off period as well as the final decision is only available after the user started the cool-off period.

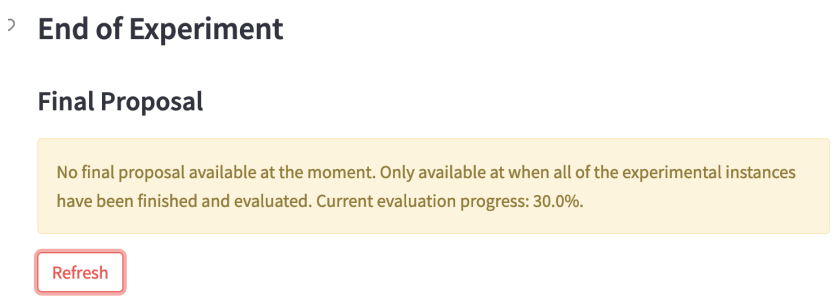


Figure 4: Information about Open Instances in Cool-off

During the cool-off period, incoming instantiation requests are routed to the default

version and the newly started instances are not part of the experiment and evaluation.

3.4 Final Decision

After all the experimental instances have finished, the user is asked to make a **final decision**. This decision entails setting a winning version for each customer category. After this choice has been submitted, all the incoming instantiation requests will be routed per this decision. There are two main resources that the user can consult to help make a decision: the data insights and the final proposal by the RL agent. The data insights are a collection of tools allowing the user to view the data collected in varying degrees of granularity, more information can be found in Section 3.6. The **final proposal** is the routing decision the RL agent would make if it would have to route an additional instance. An important thing to note here is, that this proposal by the agent does not necessarily only contain the way the agent would exploit based on the collected data, but might also be based on the need for more exploration. So especially when there have not been many experimental instances, it might be better to make a decision based on the data insights instead of the final proposal. In the future, one might want to change this behavior to purely extract the exploitation preferences from the agent or add information regarding the uncertainty. A current screenshot from the final decision area of the prototype, including the final proposal, can be seen in Figure 5.

This concludes the explanation of the elements that are part of the process flow and that depend on the state of the experiment. The functionality that is described in the following Sections (3.5, 3.6) is available during the whole time of the experiment. An overview of the experiment states and which functionality is available in each can be seen in Table 1.

Additionally, we can differentiate between these reasons for why the experiment might be done: 'Experiment ended normally' and 'Manual choice by human expert'. The first one means that a final decision has been made in response to a final proposal, and the latter means that the experiment was ended prematurely with a manual decision.

3.5 Manual Decision

We have added the functionality to conclude the experiment at any time and make a **manual decision** for either version of the process. After this manual decision has been

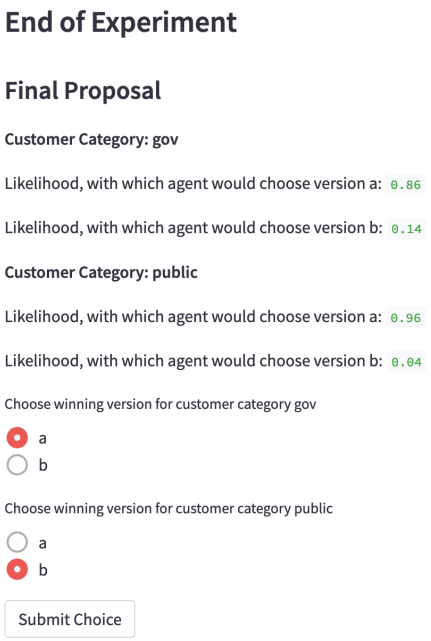


Figure 5: Final Proposal UI

Experiment State	Available/Relevant Functionality
Running, before the first batch policy has been set	View Batch Policy Proposal, Set Batch Policy, Manual Decision
Running, in batch	Data Insights, Manual Decision
Running, outside batch	Data Insights, View Batch Policy Proposal, Set Batch Policy, Manual Decision, End Experiment
In Cool-Off	Data Insights, Manual Decision
Cool-Off over, waiting for final decision	Data Insights, Final Decision, Manual Decision
Done	Data Insights

Table 1: Possible Experiment States and Available and/or Relevant Functionalities per State

made, the experiment is over and all instantiation requests are routed to the version that has been selected. One can currently not differentiate between customer categories here.

The reason for including this functionality is to allow for something like an 'emergency exit'. Imagine, for instance, the legal team of a company alerts the operations office that something is wrong and the process expert notices that it is due to either one of the process versions. Another reason for using this functionality might be an obvious choice early on, with the process expert quickly making out one version to be far better than the other or some problem with one version. In all of these cases, it would make sense for

the user to quickly disable one version and send all future requests to the better process version.

3.6 Data Insights

Since the user of the HITL-AB-BPM tool is an expert in the area of business processes and BPI, the opportunity to view information about the document and draw a conclusion based on that data is probably a core requirement. Since not all users have the same demand and capacity for details, our prototype offers the choice of different levels of granularity, aggregation, and abstraction, with which the collected data can be viewed. These views are explained in the following.

3.6.1 Experiment Metadata

On the metadata tab, the user has the opportunity to see relevant metadata about the current experiment at a glance. This information includes, for example, the process name, the relevant customer categories, and the experiment state. A screenshot of this can be seen in Figure 6.

Figure 6: Metadata UI

3.6.2 Aggregate Data

For an overview of the collected data, users can refer to the aggregate data tab, which will present them with a table containing some metrics about each version, as well as a graph with the instantiation decisions over time. Figure 7 shows a screenshot of this tab.

Figure 7: Aggregate Data UI

3.6.3 Detailed Data

In case the user wants to have an even more detailed look into the data, the detailed data section will allow them to see information about every experimental instance. They can choose from which batch they want to see the data and will then be presented with a table containing information for all instances. In case they want to analyze the data in a more customized manner, the user also has the option to download this data as a CSV file. The main parts of this screen can be seen in the screenshot in Figure 8.

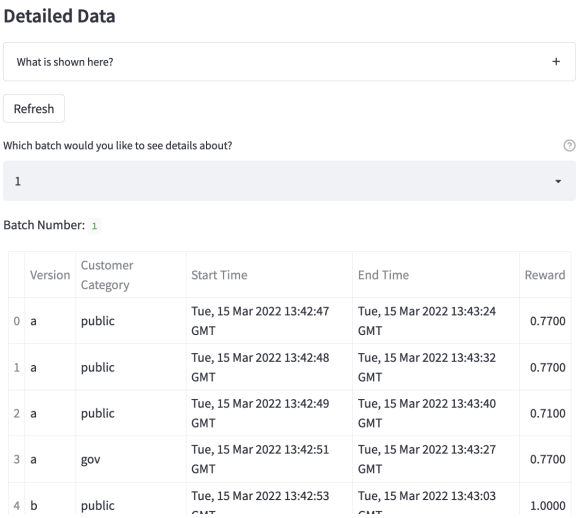


Figure 8: Detailed Data UI

4 Architecture and Implementation

To realize the prototype, several different components and technologies are required. To combine all these different components, we use Docker and Docker Compose to manage the infrastructure. Docker and Docker Compose[2] are used to ease reproducing the development environment on different local setups. For communication between components, we use a Docker bridge network. Figure 9 illustrates the prototypes' architecture. The prototype consists of three main components. The backend is the management component that serves the process instantiation request of customers. The front end is the single point of interaction with the user. It communicates with the backend via RESTful API endpoints. And lastly, the process engine executes process instances and provides necessary data to train the RL agent.

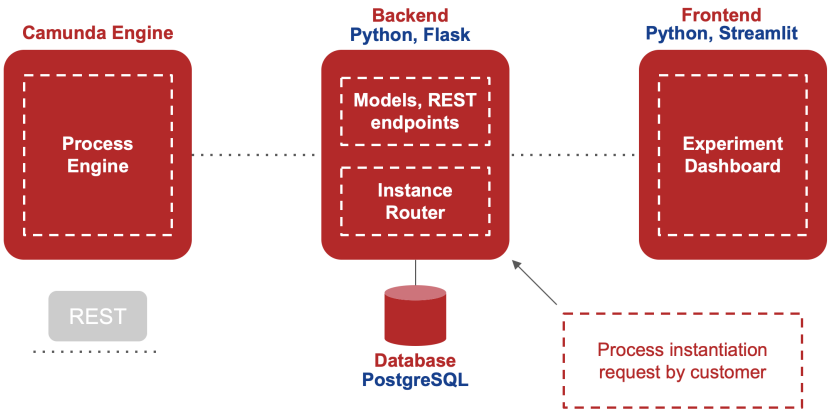


Figure 9: System Architecture, including Main Technologies Used

The backend is built using the micro-framework Flask[3] because it fits the scale of the project and allows for rapid development. It accesses the database to save all relevant information. The backend handles the logic of

1. communicating with the process engine through the REST API offered by Camunda to retrieve relevant information about processes and also initiate and stop processes,
2. managing the interaction with the RL agent,
3. housing the routing algorithm used to map requests from clients/customers to process versions,

[2] `https://www.docker.com/`, last accessed 2022-03-19
[3] `https://flask.palletsprojects.com/en/2.0.x/`, last accessed 2022-03-19

4. receiving information from the front end and using it accordingly to adjust parameters in the routing algorithm.

The front end was built via Streamlit[4] as an interface for the human in the loop. Streamlit is a front-end framework based completely on Python which allows for rapid development of prototype front-end projects without the overhead of regular front-end frameworks.

We use the Camunda engine as the process execution engine, process instances can be executed on it and relevant information about processes retrieved. Normally, this execution happens by real resources finishing real tasks. Alternatively, this execution can also be simulated, using the Camunda BPM simulator[5]. For development, testing, and evaluation of our prototype we used the simulator. More details on that can be found in Section 5. The backend can be seen as an intermediary between the clients and the Camunda engine. When a client requests a process instance through some action (e.g. placing an order in an online shop), that request would first go to the backend of our prototype. The backend would then make a routing decision and start an instance in line with that decision in the Camunda engine. Finally, the backend returns the Camunda instance identifier to the client.

The database is used to save relevant information for the configuration of the algorithm. For an overview of the data we stored in the database, please see Figure 10. PostgreSQL[6] is used to provide the database.

4.1 Reinforcement Learning Algorithm

A core functionality of the prototype is the batch policy proposal. After each batch, the proposals' values are based on the learning of the RL agent. In contrast to usual RL problems, the user has a high level of influence. Even though the user does not directly influence the learning of the RL agent, due to the acceptance or adjustment of the batch policy, the routing of the experimental instances can heavily be influenced by the human expert which in turn influences the learning of the agent. Therefore, the agent has to be adjusted to the requirements of the underlying challenges such as the interference of the human expert and the batch policy.

In this prototype, a CB algorithm is used. Normally, a CB chooses an action under a

[4] `https://streamlit.io/`, last accessed 2022-03-19
[5] `https://github.com/camunda-consulting/camunda-bpm-simulator`, last accessed 2022-03-19
[6] `https://postgresql.org`, last accessed 2022-03-19

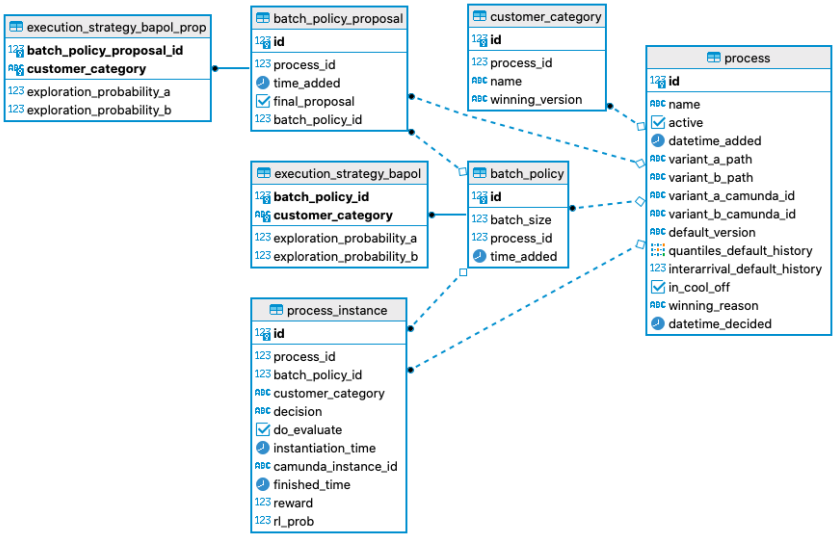


Figure 10: Database: ER Diagram

given context. Then only the reward of the chosen action/arm is revealed to the agent. Additional information (observations) is in contrast to full RL problems not allowed. The reward is calculated based on the duration of the process instance. Contrary to most RL algorithms, the goal of this model is not to maximize the accumulative reward. Instead, the reward has to be understood as a loss/cost function where the CB's goal is to minimize the loss/cost. In Section 4.1.1 the specialty of the used algorithm is explained in detail.

Incoming process requests are routed to the process engine according to the latest batch policy. The experimental instances are then executed. For each instance, a new database entry is created. An entry includes which process version is executed, a start and end timestamp, and the customer category. As mentioned above, depending on the duration of an individual process instance, terminated process instances are periodically pulled from the process engine and the end timestamp is set accordingly.

For each terminated instance that can be evaluated, the duration is calculated based on the two timestamps. The duration, the version, and customer category are passed to calculate the reward and train the model. Detailed information on the reward function can be found in Section 4.1.2. The reward is added to the process instances' database entry to offer data insights to support the human expert's decision-making. Additionally, the probabilities for each possible combination of action and context are extracted and a new batch policy proposal is set or an existing batch policy proposal is updated.

4.1.1 Vowpal Wabbit

The CB algorithm used in this prototype is part of Vowpal Wabbit (VW). VW is an open-source fast online-learning machine learning library[7]. This library was chosen because it offers a highly customizable, robust Python interface and good documentation. As mentioned before, in contrast to convention the algorithm minimizes loss/cost. For easier convertion therefore, we limited the reward space to the interval between zero and one. This does not negatively affect the performance but has to be mentioned. An advantage of the limited reward space is that it allows us to transform the reward we calculate normally to cost (cost = 1 - reward). The model learns on cost. To make the data insights more intuitive, we display the reward in the front end, not the cost.

The VW library offers a range of usable CB algorithms. For this prototype, a Random Network Distillation Explorer is used. The algorithm constructs an auxiliary prediction problem whose target value is zero. For a CB problem, this is equivalent to a randomized approximation to the LinUCB bound. For each (context, action) pair additional features are constructed due to the rich information the CB algorithm is provided, namely the tuple (action, cost, probability).

4.1.2 Reward Function

The reward function is a central component of every RL algorithm. In simple terms, a reward function provides a numeric score for each action taken, given a specific context. In this prototype process instances are only evaluated based on time (duration). It poses a challenge to find a suitable function that maps numeric scores to different durations to allow for comparisons between multiple instances. Therefore, based on the idea of [4], a step-wise reward function is implemented. In Figure 11 the adapted reward function can be seen.

The goal of a good reward function should be to penalize poor performance. That is why long-running process instances can pose a problem. These outliers have to be evaluated and considered without influencing the accumulative reward too much [4]. We divide the reward space into K=20 quantiles and define an upper cutoff (0.8) and a lower cutoff (0.2). These values can be configured in the configuration file in the source code. The quantile steps are based on the historical data from the default process version.

[7] `https://vowpalwabbit.org/`, last accessed 2022-03-19

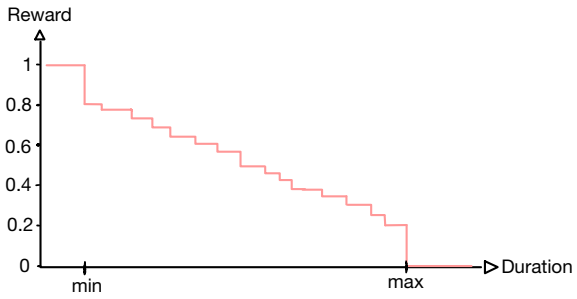


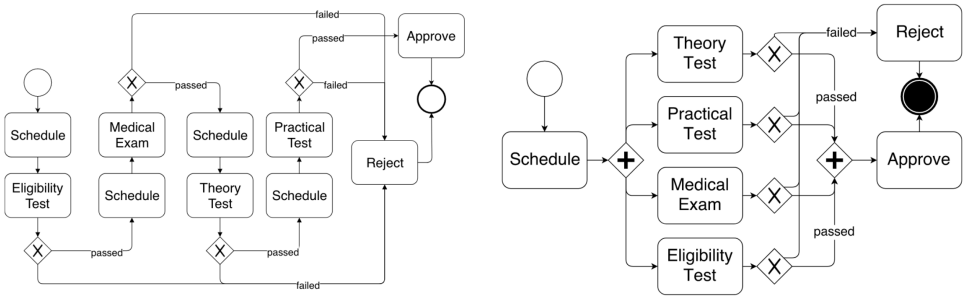
Figure 11: Step-wise Reward Function

Process instances faster than the fastest instance of the default version history receive a reward of 1, and instances slower than the slowest instance of the default version history receive a reward of 0. Instances in between the boundaries receive a reward based on which quantile they fall in.

5 Evaluation

To evaluate the prototype, we have decided to use the same exemplary process that has also been used as part of the evaluation of AB-BPM by Satyal [7]. The two process versions used have been synthetically created, based on data from the relevant business domain. This domain is helicopter pilot licensing, and the BPMN diagrams of the processes can be seen in Figure 12. Version A of the process works through the required steps of attaining a license sequentially. If none of these steps fail, then the license is approved, but if one of them fails the process ends with rejection. In Version B, these steps are parallelized, meaning that the person trying to get a license can, for example, already take their medical exam while waiting for the results of the eligibility test, and so on. This should, in theory, reduce the time that it takes for the process to conclude. And since for now, our prototype only takes the cycle time, or duration, of the process instances into account, Version B should be determined as preferential. More specific information about duration and success rate for the activities of the process can be seen in Table 2.

For our evaluation, we had to simulate both the process execution as well as the incoming customer process instantiation requests. For the process execution, we used the Camunda BPM simulator. One day in Table 2 has been down-scaled to one second in the simulation. Since our tool differentiates the incoming process instantiation requests for each customer category and [7] does not, we had to decide on customer categories.



(a) Version A

(b) Version B

Figure 12: Helicopter Licensing Process Versions; Figure from [7]

Activity	Min. processing time	Max. processing time	Success rate
Schedule	1 day	1 day	N/A
Eligibility Test	1 day	3 days	99.7%
Medical Exam	1 day	3 days	99.7%
Theory Test	2 weeks	5 weeks	72.1%
Practical Test	1 week	2 weeks	33.2%
Approve	Immediate	Immediate	N/A
Reject	Immediate	Immediate	N/A

Table 2: Helicopter License Durations and Success Rates; Table Content from [7]

Therefore, a process instantiation can either be coming from a public customer (public) or a government customer (gov). Those incoming instantiation requests are simulated by a simple Python script (client simulator), and the number of requests coming from either category is about equal. Note that the front end also includes a client simulator, but only if the development mode of the front end is activated. For this evaluation, the client simulator sent an instantiation request every five seconds. Our evaluation contains two batches, each of size 10. For the initial batch, we routed more instances to the new process version, B, for the public customer (80%) and fewer of the requests from the gov customers will be routed to this new version (20%). The argument here could be, that the government customers are more stable and important and should therefore not be exposed to the new, unverified version so much. Because we defined version A as the old, default version, we also had to upload historic execution data for it in our prototype. To get this historical data, we simulated version A one hundred times beforehand.

After the first batch has finished, and we have waited for all prior experimental instances to be finished, the RL agent presented us with a batch policy proposal. This proposal represented the fact that version B is faster, and therefore preferential, in com-

parison to version A. For customer category public, the proposed further routing would send 91% of instances to version B, and for customer category gov, 79% of instantiation requests are suggested to be sent to version B. If we look at the performance data, this makes sense: the average duration of version A was, with 72 seconds, a lot higher than the average duration of B, which was only 19 seconds. The reason why the decision of the RL has not been more clearly in favor of B might be because only a small number of experimental instances have been routed thus far, making the need for more exploration a valid concern. For the second batch of size 10, we then submitted the proposed batch policy as is.

After the second batch, we ended the experiment and waited for all experimental instances to finish. The final proposal then was also in line with the expectation: The probability, with which the RL agent would choose B was 96% for customer category public and 95% for customer category gov. The aggregate data was also matching the final proposal's suggestions: Version A took 67 seconds to finish on average, while instances of version B took, on average, only 21 seconds to finish.

We, therefore, conclude that our prototype can determine preferential process versions while allowing for full control by the process expert. One limit in our evaluation is, that the process behavior and performance were the same for both customer categories. This is because the Camunda BPM simulator does not allow for varying simulation properties based on process instantiation variables. Therefore further evaluation of the prototype regarding the behavior and usefulness of the customer category differentiation is warranted.

6 Discussion

The goal of this project was to build a software prototype that enables business process improvement using AB Testing and a contextual multi-armed bandit algorithm while incorporating the feedback of a human expert to enhance the assessment. For this purpose, we have introduced several functionalities which enable the process expert to conduct RL-driven AB tests with a high level of control. These features include, among others, batching of the experiment, routing proposals from the RL agent, and data insights. Altogether, these capabilities allow for rapid and controllable evaluation of business process improvement hypotheses in a production environment, combining the efficiency of RL

as well as the experience and risk-management capabilities of the human expert. The resulting prototype is available open source[8], to allow for further work and research on this matter.

6.1 Limitations

Yet, some limitations need to be addressed, as they offer starting points for future research opportunities. All of these limitations occur because this prototype is the result of a project in a university environment and can be rectified by further work efforts. Mainly, there are three types of limitations that we discuss in the following: (1) absence of real-life data, (2) more sophisticated functionalities, and (3) formal requirements engineering.

To begin with, the available data for testing the prototype is theoretical. Hence, the prototype needs to be tested with data collected in actually existing and operating companies. A comparison could then be made between the actual outcomes of improvement efforts with the recommendations the HITL-AB-BPM tool would have given.

Furthermore, the design and functionalities of the prototype can be improved further. For example, the human process expert should have the option to allow the agent to follow their updated decisions during the whole batch, making it possible to exploit the learned knowledge faster and avoid routing to an inferior process version. On top of that, the reward function of the current state only takes the variable "time" into account. However, real processes are often too complex to be represented adequately by only one variable. Thus, the reward function should be able to incorporate more process performance indicators, such as financial aspects, for example. Furthermore, the RL agent should be able to take ongoing instances into account for the evaluation. Next, the possibility of using other front-end frameworks should be explored, as Streamlit opposes some impediments. For example, several performance issues occurred, showing a lack of scalability and robustness, as well as missing the option to create multi-page applications. Using more evolved frameworks might be therefore a prosperous avenue to explore. These ideas can be integrated in the future. Another option to consider is a custom implementation of the contextual bandit. This could bring the prototype closer to current research ([4]) and enable easier future adaptation.

[8]Prototype available on Github: `https://github.com/aaronkurz/hitl-ab-bpm`, this paper is referring to release v0.1.0

Additionally, the features and RL algorithm properties introduced in this project have been mainly based on theoretical ideas about the needs of potential users. A more formal requirements engineering is therefore needed to determine key capabilities and properties that such a tool and the underlying algorithm and methodology should have. This is particularly important given the fairly low success rate of software projects: One-fifth of all software projects fail, while a large percentage of the rest concludes without reaching its main goals [10]. Poorly captured and maintained requirements have been identified as a major contributing factor for the failure of software projects [11]. A formal requirements engineering is hence needed to increase the probability of successful continuation of the research efforts in this field. Furthermore, since this extension of the AB-BPM methodology aims at increased viability for real-world usage in the industry, direct contact with potential users and experts on the matter early on is vital. We need to first focus on 'building the right thing' before we can go on to 'building the thing right'.

6.2 Future Research

While the prototype works within a before-mentioned environment, future research has to focus on improving the functionalities of the prototype. Hints and ideas regarding these improvements have been given in Section 6.1. At a later stage, testing the usage and effect of the HITL-AB-BPM methodology and tool in a randomized controlled trial wold be needed to verify the supposed usefulness.

Acknowledgements

This paper and the prototype presented are the results of a student project in the course "(Advanced) Distributed Systems Prototyping" at Technische Universität Berlin (TUB). The project was led by a cooperation between the chair of *Software and Business Engineering (SBE)* at TUB and *SAP Signavio*. We want to thank *SAP Signavio* and *SBE* for their continuous support, expertise, and input.

References

[1] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[2] G.J. Botha, P.S. Kruger, and M. De Vries. Enhancing customer experience through business process improvement : an application of the Enhanced Customer Experience Framework (ECEF). *South African Journal of Industrial Engineering*, May 2012.

[3] Shahrzad Roohy Gohar and Marta Indulska. Environmental Sustainability through Green Business Process Management. *Australasian Journal of Information Systems*, 24, November 2020.

[4] Suhrid Satyal. *Business Process Improvement with Performance-Based Sequential Experiments*. PhD, University of New South Wales, Sydney, July 2019.

[5] Ron Kohavi and Roger Longbotham. Online Controlled Experiments and A/B Testing. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning and Data Mining*, pages 922–929. Springer US, Boston, MA, 2017.

[6] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. MIT Press, November 2018. Google-Books-ID: uWV0DwAAQBAJ.

[7] Suhrid Satyal, Ingo Weber, Hye-young Paik, Claudio Di Ciccio, and Jan Mendling. Business process improvement with the AB-BPM methodology. *Information Systems*, 84:283–298, September 2019.

[8] Andrés Muñoz Medina, Sergei Vassilvitskii, and Dong Yin. Online learning for non-stationary a/b tests. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 317–326, New York, NY, USA, 2018. Association for Computing Machinery.

[9] Tyler Lu, David Pal, and Martin Pal. Contextual multi-armed bandits. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 485–492, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[10] Chaos Manifesto. Technical report, The Standish Group International, West Yarmouth, USA, 2018.

[11] Azham Hussain, Emmanuel O C Mkpojiogu, and Fazillah Mohmad Kamal. The Role of Requirements in the Success or Failure of Software Projects. 6:6, 2016.