Optimizing Decision-Making in Business Process Simulation with Deep Reinforcement Learning

David Sequera d.sequera@uniandes.edu.co
Universidad de los Andes, CSW

***Abstract*—This project proposes a methodology for analyzing and optimizing business processes using deep reinforcement learning (DRL). As in most resource allocation approaches, business processes are modeled as Markov Decision Processes (MDPs) and executed in a stochastic simulation environment learned from event logs. A DRL agent interacts with this environment and learns a policy over activity and resource decisions that optimizes operational performance metrics, primarily cycle time. Unlike existing work that focuses mostly on resource allocation, the proposed approach jointly optimizes event routing and resource assignment and exploits rich state representations derived from process-level embeddings. The resulting policy is validated against heuristic and learning-based baselines and evaluated not only in terms of efficiency improvements but also in terms of similarity to realistic process behavior. The goal is to obtain an agent whose recommendations can support real-time decision-making in business process management.**

***Index Terms*—Reinforcement Learning, Business Process Simulation, Policy Optimization, BPMN, MDP, Agent-based Modeling.**

I. Introduction

The efficient execution of business processes is a key factor in achieving operational excellence across industries. Traditional optimization techniques often rely on historical data or simulated data from a manually created or discovered business model. This project proposes a dynamic, intelligent agent that learns optimal decision-making strategies by interacting with simulated business environments.

The central idea is to model business processes as Markov Decision Processes (MDPs), making them suitable for Reinforcement Learning (RL). By doing so, the agent can learn policies that optimize operational performance metrics such as cost, revenue, and time across different process traces, offering real-time guidance and long-term planning capabilities.

Given an event log of a stochastic business process, we seek to learn a decision policy that, at each decision point of an ongoing process, selects the next activity and its executing resource to minimize expected cycle time. The policy must be learned from a simulation environment that faithfully reproduces the control-flow and timing behavior of the real process.

This proposal is submitted in partial fulfillment of the requirements for the Master's degree in Engineering at Universidad de los Andes.
Advisor: Oscar Fernando González Rojas, Universidad de los Andes.

II. Theoretical Framework

A. Business Process Simulation (BPS)

Business Process Simulation is a practical way to "execute" a model of a business process before changing the real one. Where classical process analysis looks for closed-form results, simulation creates an *executable digital twin* that mimics arrivals, work steps, queues, and resource constraints to estimate cycle time, throughput, costs, and service-level compliance under uncertainty. In contrast to purely analytical queuing models, BPS embraces real-world complexities such as time-varying demand, calendars, priorities, rework, and human behavior—making it useful for what-if analysis and risk-aware decision-making [1, 2, 3, 4].

A process simulation model is represented as a flow, generally using BPMN. It is composed of activities (tasks), resources (agents, machines, teams), and gateways (transitions). This flow is a directed graph of activities (nodes) and gateways (edges), such as XOR/AND, that defines the probability of transition from one activity to another. A particular path is generally called a trace, and a certain execution of an activity is called an event (with associated activity, resource, start time, and end time). In order to simulate a real process, this model uses information such as calendars for resources or roles that define the set of activities a resource can execute, arrival time processes (e.g., Poisson or empirical time series), service-time distributions (lognormal, gamma, or fitted empirical), routing probabilities, priorities, SLAs, and resource capacities. Depending on the process, these components are used to accurately represent the process; the particular approximation of this article is detailed in Section VIII.

One of the most important ways is to model BPS is *discrete-event simulation* (DES) [1, 2, 3], which reduces the simulation problem to the basic unit of a business process—an event—so that the state of the process can be described as a set of events being executed at a certain moment in time. When simulating a specific process, there are two approaches to discovering its underlying behaviors: *Data-Driven Process Simulation* (DDPS) and *Deep Learning* (DL). DDPS defines the components of the process (such as the flow previously presented) and then discovers the values of these components from an event log, including the set of activities, resources, and branching probabilities. DDPS approaches make assumptions and generalizations about the problem, which help to model it but may leave important patterns behind. DL, on the other hand, learns patterns from the log that can be difficult to identify; these patterns allow more accurate representations at the cost of less interpretable models. The current approach attempts to leverage the strengths of both approximations. This approach is called Hybrid Process Simulation (HPS), where the main framework of DDPS is used and some components are replaced with DL models, [5] examines multiple simulation approaches and provides an overview of these HPS that uses both DDPS and DL. A common use of DL is for time prediction and DDPS for the flow, as stated in [6, 7].

B. Reinforcement Learning (RL)

Reinforcement Learning is one of the most important paradigms in Machine Learning (ML) due to its intrinsic environmental learning. ML can be divided into three main paradigms: supervised learning, unsupervised learning, and reinforcement learning (other approaches such as self-supervised learning exist, but these tend to fall under one of these three paradigms). Supervised learning requires a dataset with the type of input and the correct answer, whether for a classification or regression task. Unsupervised learning does not require a correct answer; in fact, there is no definition of a specific correct answer. The model learns underlying patterns of the data; these tasks are generally dimensional reduction or clustering. Reinforcement learning does not have a dataset; it relies on an environment that gives the model a signal indicating whether the action the model selected was good or not.

From the perspective of algorithms, traditional algorithmic design implied an understanding of the problem, the shape of the solution, and finally the logical steps to get from a given configuration of the problem to a solution. ML introduced a mathematical model that not only replaces the logical steps but can also be optimized from the understanding of the problem and the solution. RL addresses problems where there is only an understanding of the problem, and the solution is not as clear as in supervised ML problems; it only has a reward signal from an action taken. This introduces concepts such as the environment (the problem), a space of actions, and a space of states where an agent (the algorithm that makes decisions) can explore and exploit based on a learned policy. This is suitable for problems where there are a series of stochastic steps to reach a desired state (the solution). This is where the concept of Markov Decision Process (MDP) serves to model the problem. In practice, for each episode—or a defined period—the agent learns from a reward function that defines the desired behavior.

Alongside other major breakthroughs, the emergence of deep neural networks marked a turning point for the field. Earlier, training deep models was hindered by vanishing gradients and overfitting, limiting the practical depth and scale of neural architectures. Today, the availability of large datasets and the development of advanced models such as CNNs, RNNs, LSTMs, Transformers, GANs, and autoencoders have made deep networks far more trainable. This progress has significantly benefited reinforcement learning as well, enabling algorithms like Deep Q-Networks (DQNs) and driving rapid advances in the field now referred to as DRL.

As described in [8], DRL algorithms can be classified into these categories: Value-Based, Policy-Based, and Actor-Critic-Based. For the purpose of this section, we center our attention on two main algorithms: Q-learning and Policy Gradient (REINFORCE). From these algorithms, we build a foundation from which we can introduce state-of-the-art algorithms such as Actor-Critic approaches, notably PPO, A2C, and A3C.

Value-Based algorithms search for an optimal value (expected value) for a given state or state-action. One example is the Q-learning algorithm, where the expected value is either a state-value $V(s)$ or a state-action-value $Q(s, a)$ [9]. Q-learning, in its most common formulation, provides a policy that for a discrete space of actions and states determines a $Q(s, a)$ that indicates which action is best for a given state. This is achieved by updating the value based on rewards using the Bellman equation, which states that the value of a state equals the expected return of its successor states, and establishes a relationship between the states and their neighbors. The Bellman equation shares similarities with a moving average and helps to propagate the value of a reward through the trace of states involved in that outcome.

The Policy-Based approach REINFORCE is an algorithm whose policy $\pi(a|s)$ is parameterized directly. This means that the model does not know how good the other states or state-actions are, but knows the probable optimal action for a given state. It learns from trajectories of states, actions, and returns, and updates the policy parameters to increase the likelihood of actions that led to higher returns based on the final result. To achieve this update of the policy, it models the problem as a supervised learning task where all actions of a given trace are classified as positive or negative interactions.

Actor-Critic takes the best of both worlds by training an actor (policy-based) and a critic (value-based), where the critic only learns from the action-state taken by the actor, and the actor learns from the advantage—a concept that indicates whether the decision made by the actor is better than the expected value predicted by the critic. It is defined as the difference between the expected value and the actual reward, showing faster convergence and important improvements [10].

Both approaches—Value-Based and Policy-Based—are viable for optimizing a business process. However, because Actor–Critic architectures can handle high-dimensional, continuous state spaces efficiently and underpin many modern DRL algorithms, we adopt an Actor–Critic design for our agent.

III. State of the Art

A. Business Process Simulation Environments

When designing an RL agent, the environment plays an essential role. If the environment does not represent reality accurately, the agent will not identify the patterns that are intrinsic to the problem you want to solve. For this reason, it is essential to ensure the environment accurately captures the reality of the process.

The environment is essentially the simulator; however, there is a subtle difference between their responsibilities. The function of a simulator is to simulate the execution of the process. Conversely, the environment is responsible for providing the agent with the next state and reward based on the action the agent took. In that sense, the environment serves as a wrapper around the simulator to provide other capabilities, such as the correct representation of the state and the calculation of the reward function.

To introduce the process simulator—the most important part of the environment—we can describe a business process with a process model. A process model is a flow of activities with a certain number of routes or possible paths, called traces. A trace can also be seen as a configuration (permutation) of a process. As described in Section II-A, this flow is represented by a directed graph with nodes (events) and edges (transitions).

Every event has an activity, start time, end time, and resources. After an event is executed, the next step is selected from the possible transitions. The set of transitions to possible next events from a given event is modeled as a probability distribution where each transition has an associated probability, also known as the branching probability. This probability is calculated as a conditional probability of the next state given the current state, computed from the frequencies observed in the event log. This is a simplified approximation that cannot capture the full range of factors that influenced the next activity. As a matter of fact, process simulators have addressed this problem with the use of DL.

For the purpose of this article, we are especially interested in the branching probabilities and also in waiting and processing times. The simple approximation of the probability distribution does not take into account factors of the trace that could lead to another possible distribution, such as past activities and resources used. To address this issue, alternatives such as [7] use Decision Trees to capture this intrinsic information for next-activity prediction. In this case, the authors use both simple probability distributions and Decision Trees to predict the next activity.

In terms of time prediction for tasks, Long Short-Term Memory Recurrent Neural Networks (LSTMs) are also used. Implementations such as [6] generate a trace and then predict the processing and waiting time of each task. On the other hand, implementations such as [7] consider the prediction as a whole and predict both the next activity and the times of each activity at each step, to capture instances where time also influences the next activity.

These environments are essential to the model because they reflect an accurate reality that the reinforcement learning model aims to capture.

B. Resource Allocation Optimization Approaches

Although there is a trend in the use of RL with BPS, it is mainly centered on resource allocation, not on the possible events (activity instances) that a certain trace could take in order to achieve better performance. These approaches vary in perspective of optimization. Most optimize activities directly, linking resources to an activity and waiting for the trace to have an event with that activity. Other approaches are for specific use cases, such as batching and domain-specific interventions. In this section, we present the most common approximations.

Approaches such as [8] use hybrid simulators for the generation of traces. This trace is used in an environment where the entire state of the process is passed to the agent, which determines the best resource for a given activity. As further work, they proposed varying the simulations and attempting to optimize the traces instead of the activities. Other resource optimizations such as [11] seek to optimize simulated traces with genetic algorithms, using a sequence of activities where a resource is assigned and then simulated to obtain the cost of that configuration.

Another perspective [12] attempts to optimize the batches of activities, taking into account several factors such as waiting time, processing time, cost, and resource utilization. In a similar approximation, [13] attempts to map the similarity of tasks with the premise that similar tasks optimize time because they share subtasks that could reduce the time when executed by the same resource. They then introduce an entropy measure that compares how well a certain worklist reduces this entropy, thereby reducing cycle time. This approach optimizes the entire process by searching for a minimum in the entropy with the use of reinforcement learning.

C. Prescriptive Process Monitoring

From the perspective of Prescriptive Process Monitoring (PrPM), articles tend to focus more on the analysis of the next activity. For example, [14] proposes a model that searches for the most optimal time to trigger an already-defined domain-specific intervention using reinforcement learning. This approach takes into account as an optimization goal the outcome of the process; it gives the example of whether a client takes a loan or not. Other perspectives such as [15] propose an interesting approach to learning the best possible path for an MDP constructed as a pair of activity and cluster. The clusters are created with a representation of the prefixes by activity frequencies and positions, then grouped by k-means. This approach uses policy iteration—an RL algorithm—to update the state-values of the MDP with a Monte Carlo simulation.

D. Additional Business Process Optimization Approaches

Apart from the general approaches for resource allocation and PrPM, [16] presents an overview of the techniques used to optimize a process. Schumann et al. present a known medical process where the objective is to assist as many patients as possible with certain restrictions. This article evaluates multiple approaches such as simulated annealing, evolutionary algorithms, tabu search, and deep reinforcement learning. It concludes that these techniques achieved better results than the reference baselines, although reinforcement learning showed a particularly promising area of investigation.

Apart from these specific resource allocation approaches, from another perspective using state-of-the-art techniques in NLP, [17] evaluates the performance of LLMs in Process Mining tasks, such as describing a process, extracting insights from a process, other approaches like [18] also evaluates the generation of a process model from a natural language description. This is particularly interesting because it translates a series of modeling problems into the world of embedding representations, where we hypothesize that a vector could represent the state of a process. If so, we can manipulate that representation to summarize, obtain insightful information, or even predict the best event-resource pair for a certain case to optimize the entire process. This concept is not new for the field (resource allocation) in the sense that LSTMs already capture the state of a process by processing the sequence of executed activities up to a certain time. The innovation is the relationship between natural language models trained for a certainly different task and the effective capture of the state with LLMs.

E. Characterization Table Definitions

- **Techniques:** The algorithms used to optimize the resource allocation.
- **Optimization Objective:** The metric or metrics that are being optimized.
- **Optimization Scope:** This could optimize the trace itself or all the instances of the process that are being executed (it is most common to optimize the entire process).
- **Testing:** The way the article compares and evaluates the performance with respect to the proposed optimization method.
- **Action Space:** The attributes that the agent is capable of controlling; these could be resources or resources and activities.
- **Type of Simulation:** Whether the article mentions the use of what-if scenarios in their implementation. What-if scenarios can be thought of as certain restrictions to the original flow of the process.
- **Process Memory:** Takes into account information about the prefix of the case instance (token) that is being processed.
- **Application Field:** Is the application field of the approximation wheter if it is for Prescriptive Process Monitoring or Resource Allocation.

IV. Motivation and Novelty

The proposed work is novel in three ways:

1) **Optimization of activity routing and resource allocation.** Prior DRL-based approaches largely optimize resource allocation given fixed control-flow, whereas we allow the agent to choose both the next event (activity instance) and its executing resource.
2) **Embedding-based state representation.** Instead of hand-crafted state vectors, we represent the process state using embeddings derived from structured JSON summaries and machine-learning predictions of next activities and processing times.

Although RL has demonstrated remarkable success in domains such as robotics and games, to the best of our knowledge, its application in business process simulation is not fully explored. RL is mainly used for resource allocation [19, 8, 13, 12], other approximations are tailored to a single use case [16] and in the field of PrPM approximations explore domain specific interventions [14] or optimize the best path within the constraints of the environment [15].

The definition of a clear framework from which processes can be analyzed as MDPs is key for real-time analysis. It allows the generalization of what-if scenarios and the evaluation of an ongoing process based on previous behaviors. The main contributions of this article are the definition of a simulation environment from a business process log, and an architecture of an agent that optimizes both resources and events (activity instances). This introduces a new set of tools to analyze and optimize processes, supporting business decision-making with a mathematical model that predicts the best scenario without the need to make simulations but with the already ongoing process. This methodology represents intersection between the simulation and prescriptive process monitoring.

V. Research Questions

The project seeks to address the following **Research Questions**:

> ***RQ1.*** *Can a deep reinforcement learning agent, operating on a stochastic business process simulation, learn a policy for the next event (activity instance) and resource selection that reduces average cycle time compared to baseline decision policies?*
> ***RQ2.*** *How does the behavior induced by the learned DRL policy differ from realistic process executions in terms of control-flow and temporal characteristics?*
> ***RQ3.*** *What is the incremental benefit of DRL-based decisions compared to purely heuristic and purely machine-learning–driven baselines?*

VI. Research Challenges

Two main challenges are anticipated:

A. Environment Design

Simulating a realistic business process requires careful modeling of stochastic elements—delays, parallel tasks, probabilistic transitions, and other factors—so that it represents the original log of the process well enough. Therefore, the environment should be dynamic so that the agent can explore the largest number of possibilities, but realistic enough so that the policy learned by the agent is capable of describing reality with a certain level of precision. This environment could be traditionally simulated or could use generative AI to generate a diverse variety of simulation scenarios.

B. Agent Interaction

Although in traditional RL environments the nature of the environment allows a clear distinction between the action of the agent and the results generated by the environment, for business process optimization, the distinction between action and result must be defined. Additionally, as in areas such as NLP where encoders can be trained on multiple tasks, an agent could learn from a list of defined tasks where the essence of the policy is shared between them.

Business Process Articles	**Techniques**	**Opt. Scope**	**Opt. Metric**	**Testing**	**Action Space**	**Simulation**	**Memory**	**Application field**	**Purpose**
Optimizing Resource Allocation Policies in Real-World Business Processes Using Hybrid Process Simulation and Deep RL [8]	DRL	Whole Process	Cycle Time	Heuristics	resources	What-if	With Memory	Resource Allocation	Optimize resource allocation via Activity–Resource pairs
Learning policies for resource allocation in business processes [19]	DRL, Bayesian Optimization	Whole Process	Cycle Time	Heuristics	resources, instance	AS IS	Without Memory	Resource Allocation	Optimize resource allocation via Activity–Resource pairs
Resource allocation using task similarity distance [13]	RL	Whole Process	Cycle Time	Heuristics & Original Log	resources	AS IS	Without Memory	Resource Allocation	Reduce worklist entropy to improve cycle time
Discovering optimal resource allocations for What-if scenarios using data-driven simulation [11]	Genetic Algorithms	Whole Process	Multiobjective (Cost, Flow Time, Waiting, Workload)	What-if Scenarios	resources, instance	What-if	Without Memory	Resource Allocation	Search optimal configurations using GA over simulated cases
Optimization of Activity Batching Policies in Business Processes [12]	DRL	Whole Process	Cycle Time & Cost	Heuristics	activity batch	AS IS	Without Memory	Resource Allocation	Optimize batching of common activities
Prescriptive Process Monitoring Under Resource Constraints: A Reinforcement Learning Approach [14]	DRL	Whole Process	Outcome of the Case	DRL	binary decision over interventions	AS IS	Without Memory	Prescriptive Process Monitoring	Improve outcomes via selective intervention policies
Recommending the Optimal Policy by Learning to Act from Temporal Data [15]	RL	Whole Process	Generic KPI	DRL	trace cluster, action	AS IS	Without Memory	Prescriptive Process Monitoring	Optimize KPI via MDP policy iteration
This Approach	DRL	Whole Process	Cycle Time	Heuristics & Original Log	resources, activities	AS IS & What-if	Simulator Memory	Resource Allocation	Search an optimized trace for a chosen objective

TABLE I: Characterization Table

VII. Contributions and Impact

This project is expected to deliver both methodological contributions and practical benefits for business process management:

- Development of a functional prototype of a reinforcement learning agent capable of learning optimal business process policies.
- A framework for modeling BPMN simulations as Markov Decision Processes (MDPs).
- Empirical validation through case studies or synthetic process models.

With these practical contributions:

- Real-time decision support, enabling managers to predict the best next actions in ongoing processes.
- Automation of resource allocation decisions and activity sequencing.
- Improved understanding of the long-term impact of operational changes, supporting both tactical and strategic process optimization.

The agent's recommendations could serve as an advisory system for decision-makers or be directly integrated into business process automation tools.

VIII. Approach

This project is mainly characterized by the use of an agent in a simulated environment that can search for an optimal policy of a action space of activities and resources. It is divided into two stages the transformation or generation of the environment from an event log and the training of the agent which is defined by an objective function composed by operational metrics (in this case the cycle time). This section is structured in two parts first the general flow of data from an event log to a optimal simulated generation from this process, and second a dive deep into the most important components of the pipeline.

A. Definitions

Let a business process be represented using the following formal components:

- **Event Log** (L): An event log is a finite set of traces:
$$L = \{\sigma_1, \sigma_2, \dots, \sigma_n\}.$$
- **Activity** (A): An activity is an element of the finite set of possible operations in the process:
$$A = \{a_1, a_2, \dots, a_m\}.$$
- **Event** (e): An event is an activity instance represented as the tuple
$$e = (c, a, t_s, t_e, r),$$
where $c \in C$ is the case identifier, $a \in A$ is the executed activity, $t_s, t_e \in T$ are the start and end timestamps ($t_s \leq t_e$), and $r \in R$ is the resource executing the event. The set of all events is denoted
$$E = \{e_1, e_2, \dots, e_k\}.$$
- **Trace** (σ): A trace is a temporally ordered sequence of events belonging to the same case:
$$\sigma = \langle e_1, e_2, \dots, e_\ell \rangle,$$
where all events in the sequence share the same case identifier c.
- **Case** ($c \in C$): A case is an instance of a process execution, associated with a trace σ and a collection of performance metrics:
$$c = (\sigma, \text{metrics}),$$
where metrics may include processing times, waiting times, resource usage, and cycle time.
- **Resource** ($r \in R$): A resource is an entity with the capability to execute a subset of activities:
$$r : A_r \subseteq A,$$
where A_r is the set of activities that resource r can perform.

Other terms that are broadly used in the field are prefix and suffix that refers to a trace until or from a certain point respectively, it is also important that to clarify that the term Trace and the term Case are usually interchangeable and depends specially on the part that the author wants to focus either on the actual permutation of activities or in the metrics related to that specific instance like time.

From the perspective of reinforcement learning these definitions clarify concepts that are essential to create this architecture.

- Environment: is in charge of the evaluation of the actions that an agent takes with a given state and define the next state, and reward based on the action. it is generally defined as Markov Decision Process (MDP) in the field of RL
- Agent: is the actor that for a given state takes a certain action based on a policy that defines the best possible action that optimize a certain objective function.

B. Data Preparation

As a part of the pipeline the model needs to discover a process simulation model, this model in broader terms is composed of three parts. First the process flow, that could be seen as a set of possible traces or paths discovered from the log of events. Second a pool of available resources and a calendar of available time intervals for each resource or role (depending on the implementation). Finally it is composed by a branching probability that is generally modeled as the conditional probability of a the next activity given the actual activity. Although in this case, we will be using machine learning alternatives that take into account the information from the prefix of the trace, as explained in [7], where the authors proposed a decision mining variation of their model using decision trees, with the expectation of capturing underling patters that the simple probability cannot.

The flow is converted in a Petri Net which could be though as a directed graph that defines a model where tokens (cases) can move from one node to another until the trace is completed. This models allows us to synthesize the process structure. after this step a ML model is trained with the task of predicting a branching probability, for a given ongoing case what is the most probable next activity, finally we allocate resources based on disposability, although in this part the agent will be responsible of selecting from the pool of activities the most optimal resource for that specific instance.

C. Environment

Once the process simulation model is created we wrapped the simulator so that we can define a formal environment, this environment has three key functions, the first one is to receive and interpret the actions that comes from the agent, the second one is to evaluate the performance of this actions, in this particular case cycle time and the third one is to generate the next state for the agent to predict the best action to take. We will be using the simulator described in [7] also used in [8] with a some modification, specifically in the activity decision part. We replace the environment's internal sampling of next activities with decisions proposed by the DRL agent, while still using the ML-based time predictions of [7]. Additionally we don't care about white box interpretation and we expect neural networks to predict better this AS-IS probability so instead of decision trees as proposed in [7] we are going to use LSTMs.

We selected this model due to the improvement in the prediction of times and also for the inclusion of a more robust model to represent the distribution of possible next activities in a certain moment of the trace. Once the simulator estimates the cost in time of the action that the agent took, the environment then calculates a reward function that uses the previously calculated values and then provides the agent with the respective reward indicating the agent if the action that was taken resulted in an improvement or not.

D. State

The state is described in this part is quite different that usual approaches like [15, 19, 8] . It uses NLP techniques to generate the state of the actor, as demonstrated in [20] NLP embedding representations goes beyond the field of natural language processing and transcends to complex fields like image generation. Also taking into account works like [18] LLMs are capable of understand business process. With this background into account we proposed a new pipeline for the generation of the state. State in the initial state will be extracted from the actual state of the simulation as an ongoing process (a set of cases) that will be modeled as a strictured format with calculated metrics, in this case a JSON file with the information of the number of cases, resources utilization, resources available, list of the last n events of the whole process. Then this information is processed by a encoder that converts this sequence into a embedding representation of the state, this state is then passed to the agent allowing it to capture much more information than with another feature engineered approach. Additionally we will concatenate the probability distribution predicted by the environment in order to facilitate the decision and stability of the agent knowing which option is the most probable nonetheless probably not the optimal.

We will use a transformer-based encoder trained jointly with the DRL agent on the structured JSON representation of the state. The encoder parameters are updated through the DRL loss, allowing the state representation to adapt to the decision task.

E. Agent

The agent is a DRL agent, Actor-Critic which tries to estimate both, the stat-value and the best possible action for a given state, this is described as a probability distribution of a discrete set of activities, in this case we have 2 probability distributions, one for the set of possible activities and one for the set of possible resources, after the prediction there is a mask as in most DRL implementation where the unfeasible actions and resources for a given state are discarded. For the Critic part the agent takes the actual state and predicts the expected future reward, this is useful because it allows the agent to calculate how good is the decision that the actor took based on what is the expected value for that state. Similar to the Generative Adversarial Networks (GANs) this is a cooperative network where the better the Critic is the better the Actor becomes and vice-versa.

The agent goal is to converge in an optimal policy $\pi(\alpha|s_n)$ or also $\pi((a,r)|s_n)$ where α is a pair of (a,r) activity and resource, for a given case. This policy predicts the best action α for a given state ideally for any given state, this is called generalization. This policy can from any given state predict the best possible suffix for a certain objective, this predictions could be greedy (deterministic) or stochastic with methods like top-k or top-p used widely in NLP.

Finally the agent with this policy can analyze traces of ongoing process and predict the best possible suffix given the actual state. This could lead to an optimal suggestions not only for simulations but for logs until a certain point, showing the optimal What-if scenario, with or without variations.

F. Mask

Since the agent will try to take greedy actions, we will be using a top-p and top-k approximation not for sampling but for limiting the actions that the agent can take with the purpose of maintaining the desired patterns of the process. The action space will be limited or masked only for the available actions for a certain state, this actions will be described by the probability distribution given by the environment. We will define a certain top-k elements and/or top-p probability to get a certain flexibility especially for activities, in the case of resources it will use also a mask calculated with the use of the actual state.

This approximation is a common practice in DRL as in implementations like [19] where Middelhuis et al. use masks to prevent the agent from doing unfeasible actions.

G. Reward Function

The reward function will be calculated with respect to the cycle time, as the optimization goal is to reduce the cycle time we will use a similar reward function to the ones proposed in [19, 8] .

$$r(\sigma) = \frac{100}{ct(\sigma)}$$

Additionally we are searching for a certain threshold to simulate the business logic of a process where the goal is to process all case with in a certain period of time. The agent will get this reward at the end of the trace. This will incentivize a behavior where the reward most be below that value.

$$R(\sigma) = \begin{cases} +r(\sigma), & ct(\sigma) < T \\ -r(\sigma), & ct(\sigma) \geq T \end{cases}$$

This type of reward is feasible because RL algorithms where based on Dynamic Programming concepts like backtracking where the last action has an impact in all the other previous actions. This reward will propagate through out the other previous actions influencing the agent, so that it will prioritize actions that lead to traces that have a cycle time below the threshold.

IX. Evaluation

The goal of the evaluation is twofold: (i) to assess whether the proposed DRL-based decision-making agent can reduce cycle time compared to heuristic and learning-based baselines, and (ii) to analyze how much the DRL-generated behavior deviates from realistic process executions in terms of control-flow and temporal characteristics.

A. Research Hypotheses

We formulate the following hypotheses:

H1 (Effectiveness): The DRL agent yields lower average cycle time than baseline decision policies across multiple processes.

H2 (Realism): The event logs generated under the DRL policy remain similar to realistic executions, as measured by control-flow and temporal distance metrics, i.e., the DRL policy does not "game" the simulator in ways that destroy process realism.

H3 (Incremental benefit): Adding DRL-based resource allocation on top of ML-based activity decision models yields incremental improvements over purely ML-driven or purely heuristic baselines.

B. Baseline Policies

We compare the proposed DRL agent against five decision policies. Each policy is defined as a combination of: (i) an *activity selection rule* and (ii) a *resource allocation rule* (subject to feasibility constraints). All policies operate on the same simulation environment and underlying process model.

a) RA-RR: Random Activity + Random Resource:

- **Activity selection:** At each decision point, select the next activity uniformly at random from the set of feasible activities (or proportionally to the empirical branching probabilities estimated from the log, depending on the implementation choice).
- **Resource allocation:** Assign a resource uniformly at random from the set of feasible and available resources for that activity.
- **Purpose:** Naive baseline, establishes a lower bound on performance.

b) GP-RR: Greedy Probability + Random Resource:

- **Activity selection:** Select the next activity with the highest empirical branching probability (i.e., basic conditional probability estimated from the log).
- **Resource allocation:** Random feasible resource as in RA-RR.
- **Purpose:** Tests the effect of a simple greedy control-flow policy that ignores resource intelligence.

c) DM-RR: Decision Model + Random Resource:

- **Activity selection:** Use the decision mining / deep learning next-activity model from the simulation environment (e.g., LSTM or other ML model trained on prefixes) to obtain a probability distribution over next activities; select the most probable activity (argmax). This model is designed to mimic the real process and is not necessarily optimal with respect to cycle time.
- **Resource allocation:** Random feasible resource.
- **Purpose:** Represents a "best effort" control-flow predictor that reproduces the original process behavior, but without optimized resource allocation.

d) DM-DRL: Decision Model + DRL Resource Agent:

- **Activity selection:** ML decision model as in DM-RR (argmax or sampling according to probabilities).
- **Resource allocation:** DRL agent restricted to resource choices only (i.e., the action space is the set of feasible resources for the selected activity).
- **Purpose:** Isolates the contribution of DRL for resource optimization on top of a realistic activity-selection policy.

e) DRL-AR: Full DRL Agent (Activity + Resource):

- **Activity selection:** DRL agent jointly selects the next activity and the resource, i.e., an action is an (a, r) pair, with masking for infeasible pairs.
- **Resource allocation:** Implied in the joint action.
- **Purpose:** Evaluates the full proposed architecture, where the agent learns a joint policy $\pi(a, r \mid s)$ to optimize the cycle time objective.

All policies are evaluated in the same environment, with identical process models, resource calendars, arrival processes, and time-prediction components. This allows us to attribute performance differences to the decision policy rather than to differences in process configuration.

C. Evaluation Metrics

We evaluate the policies along two main dimensions: (i) efficiency (cycle time) and (ii) behavioral similarity between logs.

1) Efficiency Metrics: For each process and each policy, we run multiple independent simulations and log all completed cases. From these logs we compute:

- **Average cycle time** $\overline{ct}$: mean completion time per case.

- **Cycle time distribution:** Empirical distribution (e.g., histogram or kernel density) to assess not only the mean but also dispersion and tail behavior.
- **Standard deviation and confidence intervals:** For each policy, we report mean and standard deviation of $\overline{ct}$ over repeated simulation runs, and optionally 95% confidence intervals.

Formally, for a simulation run generating a log L with cases $\sigma \in L$,

$$ct(\sigma) = \text{end\_time}(\sigma) - \text{start\_time}(\sigma), \tag{1}$$

$$\overline{ct}(L) = \frac{1}{|L|} \sum_{\sigma \in L} ct(\sigma). \tag{2}$$

We repeat the simulation N times per policy (e.g., $N = 10$) and compute:

$$\mu_{\text{policy}} = \frac{1}{N} \sum_{i=1}^{N} \overline{ct}(L_i) \tag{3}$$

$$\sigma_{\text{policy}} = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} \left(\overline{ct}(L_i) - \mu_{\text{policy}}\right)^2}. \tag{4}$$

2) Behavioral Similarity Metrics: To ensure that improvements in cycle time do not come at the cost of unrealistic behavior, we measure the similarity between: (i) a *reference log* (e.g., log generated by the DM-RR policy or by the original environment trained to match the real process), and (ii) the log generated under the DRL policy (DRL-AR), and optionally other baselines.

We analyze both control-flow and temporal dimensions:

a) Control-flow log distance.: We use a control-flow distance metric between event logs, such as a footprint-based or directly-follows graph (DFG)-based distance. Intuitively, this metric captures differences in which activity sequences appear and how often.

Let L^{ref} be the reference log and L^{DRL} the DRL-generated log. We compute a distance $d_{\text{cf}}(L^{\text{ref}}, L^{\text{DRL}})$, where lower is better (more similar), optionally normalized to $[0, 1]$.

b) Absolute event distribution distance.: We compare the marginal distribution of activities (and optionally activity pairs) between logs. Let $p^{\text{ref}}(a)$ and $p^{\text{DRL}}(a)$ be the relative frequency of activity a in each log. We compute an absolute distribution distance such as:

$$d_{\text{act}} = \sum_{a} \left| p^{\text{ref}}(a) - p^{\text{DRL}}(a) \right|. \tag{5}$$

This captures how much the DRL agent alters the usage frequencies of activities.

c) Cycle time distribution distance.: For the temporal dimension, we compare the distribution of cycle times rather than only the mean. We can use, for instance, the Kolmogorov–Smirnov (KS) distance between the empirical CDFs of cycle times in both logs. Let F^{ref} and F^{DRL} be the empirical CDFs of cycle times. The KS distance is:

$$d_{\text{ct}} = \sup_{t} \left| F^{\text{ref}}(t) - F^{\text{DRL}}(t) \right|. \tag{6}$$

Together, $(d_{\text{cf}}, d_{\text{act}}, d_{\text{ct}})$ give a compact view of how much the DRL-driven behavior deviates from the natural behavior learned from the log.

D. Datasets and Simulation Logs

We evaluate the approach on multiple business processes, using a 2:1 ratio of real to synthetic logs, and selecting both widely used benchmarks and more recent datasets.

We select six event logs: four real-life logs and two synthetic logs. The real logs are chosen from widely used collections in process mining (ensuring comparability with prior work), complemented with more recent logs used in simulation and decision-mining studies. The synthetic logs are based on complex reference models designed to stress-test decision policies under high variability and concurrency.

- **Real logs (4):** A healthcare or financial process from a commonly used repository (e.g., BPI Challenge / hospital billing / loan application), plus additional real logs used in recent hybrid simulation and deep learning work where the environment is already validated against reality.
- **Synthetic logs (2):** Two complex synthetic processes with high concurrency and multiple decision points, similar to the elaborate synthetic processes used in hybrid simulation and DRL resource allocation studies. These allow controlled variation of process structure and resource constraints.

(Concrete dataset names can be plugged in once the final selection of public logs is made.)

E. Experimental Protocol

To reduce stochastic variance and obtain statistically meaningful comparisons, we follow the protocol below.

a) Number of simulation runs.: For each combination of process (6 logs) and policy (5 decision policies), we execute 10 independent simulation runs, each with a different random seed. Each run produces an event log with a fixed number of completed cases (e.g., 5,000–10,000) or a fixed simulated time horizon.

b) Aggregation.: For each run, we compute $\overline{ct}(L)$ and the behavioral distance metrics with respect to the reference log. For each policy and process, we report mean and standard deviation across the 10 runs.

c) Statistical tests.: Optionally, we perform statistical tests to assess significance of differences, such as paired t-tests or Wilcoxon signed-rank tests comparing DRL-AR against each baseline per process (on average cycle times). We also report effect sizes (e.g., Cohen's d) to quantify the magnitude of improvements.

d) Reference log for similarity.: For similarity metrics, we consider as reference: (i) the original real log (for real processes), or (ii) the log generated by the DM-RR policy calibrated to the original process (for synthetic processes where the environment is the source of truth). This allows us to interpret DRL deviations as "how far from the realistic process" the optimized policy moves.

e) Hyperparameter stability.: DRL training is inherently stochastic. To control for this, we fix a set of hyperparameters for PPO (learning rate, batch size, discount factor, clipping range), tuned on a validation process (not used for final reporting), and train each DRL policy multiple times with different seeds, selecting either (i) the best-performing checkpoint according to validation cycle time, or (ii) an average over the last K checkpoints to reduce variance.

F. Expected Analysis

The analysis will first compare average cycle times across policies to assess absolute and relative improvements of the DRL agent. Next, we will study behavioral similarity metrics to understand to what extent the DRL policy alters the control-flow and temporal characteristics of the process. In particular, we are interested in configurations where the DRL agent achieves significant cycle time reductions while maintaining low control-flow and temporal distances, as these correspond to policies that are both efficient and realistic from a business perspective.

X. Conclusions and Future Work

A. Expected Conclusions

We expect the DRL agent to generate processes with a reduced cycle time while preserving a high degree of similarity to the original process in both control-flow and temporal dimensions. In essence, the optimized policies should improve performance without producing unrealistic or structurally implausible traces.

B. Future Work

a) Multi-objective reward function.: A natural extension is the transition from a single-objective reward (cycle time) to a multi-objective formulation. This would allow the agent to jointly optimize outcome-oriented metrics, as in [14], fairness in workload distribution, economic cost, and other operational indicators, either through weighted combinations or Pareto-based approaches.

b) Zero-shot DRL with latent representations.: Since the current state representation is a structured JSON, and given evidence from NLP and multimodal models that latent spaces allow broad generalization [20], as well as the emerging results of LLMs in process mining tasks [17], an interesting research direction is *zero-shot DRL*. Here, instructions or goals could be reformulated at inference time so that the agent solves tasks not explicitly present during training, leveraging shared latent structures between processes, objectives, or scenarios.

c) Evaluation on diverse business process patterns.: The proposed architecture can be evaluated on broader classes of business processes, including batching scenarios [12] and intervention-based prescriptive monitoring [14]. This would assess whether the same state–action–reward framework can generalize to variants such as batching decisions, intervention timing, or outcome-oriented optimization with minimal modification.

d) Variation of simulation parameters and environments.: Although full validation of the simulation environment is beyond the scope of this work, the environment remains a critical part of the architecture. Future research should explore different simulators and parameterizations, including alternative next-activity selection strategies, queueing disciplines, and resource calendars. Investigating these variants, grounded in queueing theory and hybrid simulation techniques, may reveal how robust the DRL agent is to modeling assumptions and how much performance depends on the environment versus the agent architecture itself.

XI. Methodology and Schedule

This project will follow the Design Science Research (DSR) methodology, structured into five phases to be developed over a 12-month period.

Phase	Guideline	Start Date	Weeks	Output
State of the art Revision	5	01/09/2025 - 3/10/2025	5	article SLR
Opportunities of Investigation and Scope	2	06/10/2025 - 31/10/2025	3	Definition of state for the agent, tasks, training logs and success metrics
Prototype implementation	1	04/11/2025 - 21/11/2025	2	Create a simple prototype

TABLE II: Thesis 1 Planning

for the second semester we are planning the following stages

Phase	Guideline	Start Date	Weeks	Output
Implementation	1,4,6	19/01/2026 - 10/04/2026	11	Training pipeline, log to environment definition, agent model and architecture diagrams
Experimentation	3,5	13/04/2026 - 08/05/2026	3	results of test, evaluation metrics, advantages, disadvantages
Conclusions and results divulgation	7	11/05/2026 - 22/05/2026	2	Research article Master thesis Open-source code

TABLE III: Thesis 2 Planning

References

[1] A. M. Law, *Simulation Modeling and Analysis*, 5th ed. McGraw-Hill, 2015.

[2] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 5th ed. Prentice Hall, 2010.

[3] M. Laguna and J. Marklund, *Business Process Modeling, Simulation and Design*, 3rd ed. CRC Press, 2019.

[4] D. Barón-Espitia, M. Dumas, and O. González-Rojas, "Automated generation of process simulation scenarios from declarative control-flow changes," *PeerJ Computer Science*, vol. 10, p. e2094, 2024. [Online]. Available: https://doi.org/10.7717/peerj-cs.2094

[5] S. Khraiwesh and L. Pufahl, "Review of design of business process simulation models," in *Research Challenges in Information*

Science, J. Grabis, T. E. J. Vos, M. J. Escalona, and O. Pastor, Eds. Cham: Springer Nature Switzerland, 2025, pp. 452–469.

[6] M. D. O. G.-R. Manuel Camargo, Daniel Báron, “Learning business process simulation models: A hybrid process mining and deep learning approach,” *Information Systems*, vol. 117, p. 102248, 2023. [Online]. Available: https://doi.org/10.1016/j.is.2023.102248

[7] F. Meneghello, C. D. Francescomarino, C. Ghidini, and M. Ronzani, “Runtime integration of machine learning and simulation for business processes: Time and decision mining predictions,” *Information Systems*, vol. 128, p. 102472, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437924001303

[8] F. Meneghello, J. Middelhuis, L. Genga, Z. Bukhsh, M. Ronzani, C. Di Francescomarino, C. Ghidini, and R. Dijkman, “Optimizing resource allocation policies in real-world business processes using hybrid process simulation and deep reinforcement learning,” in *Business Process Management*, A. Marrella, M. Resinas, M. Jans, and M. Rosemann, Eds. Cham: Springer Nature Switzerland, 2024, pp. 167–184.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: The MIT Press, Nov 2018, adaptive Computation and Machine Learning series.

[10] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems*, 3rd ed. Birmingham, UK: Packt Publishing Ltd., Nov 2024, includes new content on human feedback, MuZero, and LLMs.

[11] J. Bejarano, D. Barón, O. González-Rojas, and M. Camargo, “Discovering optimal resource allocations for what-if scenarios using data-driven simulation,” *Frontiers in Computer Science*, vol. 5, p. 1279800, 2023. [Online]. Available: https://doi.org/10.3389/fcomp.2023.1279800

[12] O. López-Pintado, J. Rosenbaum, and M. Dumas, “Optimization of activity batching policies in business processes,” in *Business Process Management*, A. Senderovich, C. Cabanillas, I. Vanderfeesten, and H. A. Reijers, Eds. Cham: Springer Nature Switzerland, 2026, pp. 397–414.

[13] M. Yaghoubi and M. Zahedi, “Resource allocation using task similarity distance in business process management systems,” in *2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS)*, 2016, pp. 1–5.

[14] M. Shoush and M. Dumas, “Prescriptive process monitoring under resource constraints: A reinforcement learning approach,” *KI - Künstliche Intelligenz*, vol. 39, no. 2, pp. 119–140, 2025. [Online]. Available: https://doi.org/10.1007/s13218-024-00881-6

[15] S. Branchi, A. Buliga, C. Di Francescomarino, C. Ghidini, F. Meneghello, and M. Ronzani, “Recommending the optimal policy by learning to act from temporal data,” *arXiv*, 2023. [Online]. Available: https://arxiv.org/abs/2303.09209

[16] F. Schumann, M. Ehrendorfer, M. Kunkler, K. Busch, H. Leopold, L. Urny, M. Schmauch, O. Rodzik, S. Lanz, E. Tıraş, A. Al-Zamkan, J. El Kari, and R. Dijkman, “The business process optimization competition,” in *Business Process Management Workshops*, K. Gdowska, M. T. Gómez-López, and J.-R. Rehse, Eds. Cham: Springer Nature Switzerland, 2025, pp. 61–72.

[17] A. Berti, H. Kourani, and W. M. P. van der Aalst, “Pm-llm-benchmark: Evaluating large language models on process mining tasks,” in *Process Mining Workshops*, A. Delgado and T. Slaats, Eds. Cham: Springer Nature Switzerland, 2025, pp. 610–623.

[18] H. Kourani, A. Berti, D. Schuster, and W. M. P. van der Aalst, “Evaluating large language models on business process modeling: framework, benchmark, and self-improvement analysis,” *Software and Systems Modeling*, September 2025. [Online]. Available: https://doi.org/10.1007/s10270-025-01318-w

[19] J. Middelhuis, R. L. Bianco, E. Sherzer, Z. Bukhsh, I. Adan, and R. Dijkman, “Learning policies for resource allocation in business processes,” *Information Systems*, vol. 128, p. 102492, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437924001509

[20] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, “Hierarchical Text-Conditional Image Generation with CLIP Latents,” 2022, arXiv preprint.

David Sequera MSc candidate in Engineering at Universidad de los Andes with a focus on AI-driven business process analysis. Interested in the intersection of simulation, decision automation, and machine learning.