Egyptian Informatics Journal 29 (2025) 100595

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Full Length Article

Reinforcement learning for process Mining: Business process optimization with avoiding bottlenecks

Ghada Soliman [1,*], Kareem Mostafa, Omar Younis

Department of Data Science, Orange Innovation Egypt, Cairo, Egypt

ARTICLE INFO

Keywords:
Process Mining
Business Process Optimization
Machine Learning
Reinforcement Learning
Q-learning
Deep Q-network

ABSTRACT

Process mining extracts knowledge from event data to understand, analyze, and improve processes. The activity of manually identifying the optimal sequence of transitions for process mining is challenging due to several difficulties, including the high complexity of business processes that may involve numerous activities and decision points, and the significant effort required to collect and analyze the necessary data, ex1plore different transition possibilities, and evaluate their impact on process performance. In our study, we have implemented Process Mining Environment in OpenAI gym format aimed at enhancing the development of reinforcement learning algorithms for process optimization tasks. The capabilities of these approaches of reinforcement learning using the Q-learning and Deep Q-network (DQN) techniques to identify the optimal path. This is achieved by constructing a reward matrix tailored to each method, designed to circumvent absorption states that signify bottlenecks. The environment was tested using a proprietary dataset containing 3,414 tickets, with event logs sourced from the ServiceNow ticketing system. The findings indicate a significant reduction in the action space, with Q-learning and DQN achieving a decrease of 75% and 67%, respectively.

1. Introduction

Process mining empowers businesses to drive process excellence, enhance competitiveness, and achieve strategic objectives by providing decision makers with a clear understanding of how their processes are executed [1]. Process mining techniques use the generated event logs to trace data about specific events or activities occurring [2]. Analyzing business processes through process mining techniques is valuable for identifying the optimal sequence of transitions based on real process data. Processes in real-life scenarios can be complex and involve numerous interactions, dependencies, and variations. Capturing and modeling these complexities accurately in process mining can be challenging, computationally intensive and time consuming [3].

Predictive business process monitoring is a process mining technique that involves using data analytics and machine learning techniques to forecast the future states of ongoing business processes [4]. This approach aims to predict potential outcomes, such as delays, bottlenecks, or failures, before they occur, allowing organizations to take proactive measures. Historical process data is analyzed based on the event log of a certain organization's ticketing system to identify patterns and trends. Utilizing reinforcement learning for automating the prediction of the business process events is considered one of the recent topics that has not been extensively researched and explored. The intelligent agent interacts with a customized environment that represents the possible states occurring within the ticket life cycle according to specific constraints aligned with business rules. Accordingly, the agent can predict valid transitions that conclude the optimal sequence of activity transitions, avoiding the bottleneck state toward the end of the activity trace [5].

1.1. Contribution

The initial stage of our pipeline is dedicated to extracting the event log, which records every phase of a ticket's life cycle within the system, from initiation to closure. During the preprocessing phase, essential preparatory steps for the experiment are undertaken, including a thorough analysis of the raw event log's Directly-Follow Graph (DFG). This analysis helps to detect events of high frequency and correlate them with

Abbreviations: DFG, Directly-Follow Graph; MDP, Markov Decision Process; RL, Reinforcement Learning; ANN, Artificial Neural Network; DQN, Deep Q-network.

* Corresponding author at: Cairo, Egypt.
E-mail addresses: ghada.soliman@orange.com, ghada.soliman@gmail.com (G. Soliman), kareem.mostafa.ext@orange.com (K. Mostafa), omar.younis.ext@orange.com (O. Younis).
[1] Lead Data Scientist.

https://doi.org/10.1016/j.eij.2024.100595
Received 4 December 2023; Received in revised form 18 August 2024; Accepted 14 December 2024
Available online 21 December 2024

business conditions and rules, providing insights into potential bottlenecks. Subsequently, all the successor events to the bottleneck state will be eliminated. This was done to create an 'absorption state' scenario, where once reached, the state persists indefinitely. A comprehensive table is then constructed, encapsulating all related events within the event log. This table delineates each event as a source/target transition, offering a complete overview of all process transitions to facilitate subsequent steps. The transition probability for each identified transition is then calculated, which serves as the reward value for reinforcement learning during training.

In our work, the pipeline was presented with the preprocessing of the event logs and then employing reinforcement learning techniques including Q-learning and Deep Q-network (DQN) to determine the optimal path towards the target state aiming to avoid a bottleneck state. In the Q-learning experiment, a Q-table is initialized to represent the evolving policy algorithm, reflecting the cumulative quality of executing a transition to another state, given the current state over time. For the DQN approach, the Q Network model was utilized to approximate the Q-values, enabling the handling of larger state spaces with complex patterns. The DQN approach incorporates the experience replay technique that facilitates unbiased sampling by enabling the agent to learn from a diverse and representative sample of past experiences, thereby reducing recent transition bias and breaking temporal correlations to infer the importance of historical information. The training is conducted for each approach for a predetermined number of episodes until convergence is achieved, resulting in an optimal policy within the Q-table and Q Network model for Q-learning and DQN respectively to avoid the bottleneck state. The reward function designed for each of Q-learning and DQN, steers the agent during training by offering rewards for desirable actions and imposing penalties for failing to progress to either a valid normal state or the end state at each time step. In more detail, transitions to the bottleneck state are assigned a significantly low value, discouraging the agent from such transitions as it aims to maximize overall reward. Fig. 1 illustrates the high-level components of our system pipeline. The subsequent sections will delve into the specifics of each component.

Our paper contributes into the optimization of the business processes using reinforcement learning in terms of the following:

- The study adopted Process Mining Environment conforming to OpenAI gym specification aimed at enhancing the development of reinforcement learning algorithms for process optimization tasks.
- The study implemented a predictive monitoring model for the business process to identify the next possible state for the process given a certain event by leveraging reinforcement learning value-based approaches using Q-learning and DQN in an aim to determine the optimal path towards the target state with avoiding the absorption state that acts as the bottleneck state in the business process.
- The study designed a reward matrix in separate for each approach of Q-learning algorithm and Deep Q-network such that lower rewards are assigned to discourage the agents from entering in the bottleneck state and higher rewards to encourage the agents navigating to the terminal and end state.
- The study used Adaptive epsilon-greedy exploration is used in reinforcement learning to balance the trade-off between exploration and exploitation. Initially, a higher epsilon value encourages more exploration to discover new strategies and then reducing epsilon gradually over time allows the agent to exploit the knowledge it has gained, focusing on the best-known actions.
- The study utilized the transition probability for DQN as a reward for the normal transition by dividing the count of the outgoing edges from the source to the target state by the sum of the outgoing edges from the source state. To maximize the benefit of the transition probability for the reward, a high reward was assigned for the normal transition from source to target state having maximum transition probability whilst low reward for the normal transition.
- The study employed the experience replay by randomly sampling a batch of experiences during the training of DQN and hence allows the agent to learn from the previous experiences' multiple times during the training phase in an aim to avoid forgetting previous experiences and helps to stabilize the learning process by reducing the likelihood of catastrophic oscillations or divergence in action values.
- The study evaluates the results by comparing the outputs of each of Q-learning and DQN against the expected business process outcomes and through the visualization of the cumulative reward and regret over time, the assessment was made on how quickly the agent is improving its policy and how much the agent's performance deviates from the optimal policy.

The rest of the paper is organized as follows: Section 3 describes the event log used in the experiment associated with the preparatory steps needed for applying the experiment. In Section 4, the applied reinforcement learning methodologies are explained including the Q-learning and Deep Q-network algorithms, Section 5 introduces the achieved results for the applied techniques. Finally, Section 6 concludes the paper.

2. Literature review

The application of advanced analytical techniques to business process monitoring has become a pivotal area of research, with significant contributions that aim to enhance decision-making and optimize process outcomes. There has been an increasing attention towards process mining techniques for discovering, monitoring, and optimizing the business processes. This literature review section will explore seminal works and recent advancements in the field, focusing on the integration of machine learning and reinforcement learning approaches for

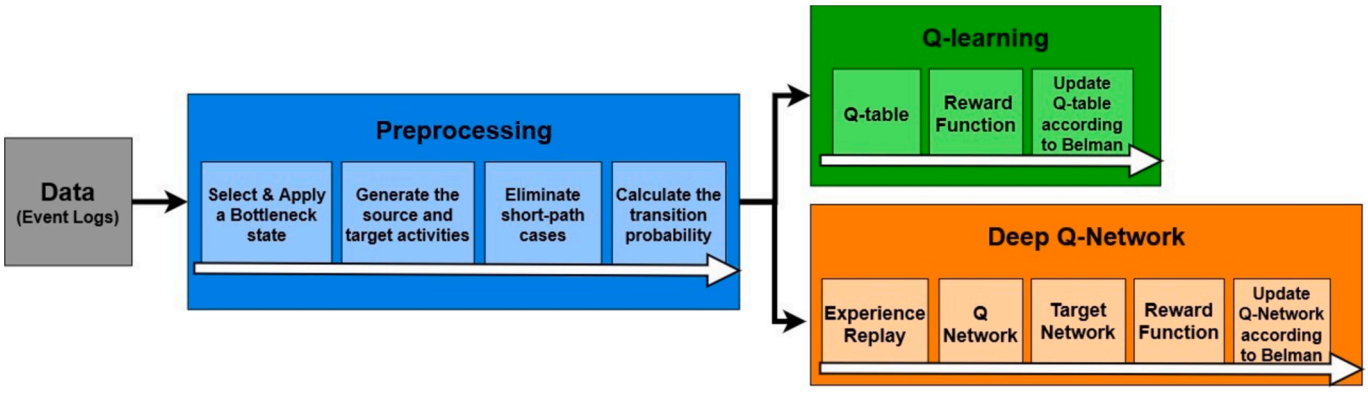

Fig. 1. The system's main pipeline.

predictive business process monitoring.

Zhengxing et al. (2011) [6] propose a mechanism that models the resource allocation optimization problem as Markov decision processes, which are solved using reinforcement learning. This approach allows for dynamic and intelligent decision-making based on environmental feedback, aiming to optimize long-term performance in BPM. The paper presents experimental results demonstrating that the proposed reinforcement learning mechanism outperforms heuristic and hand-coded strategies. It suggests that the mechanism can significantly improve the state of resource allocation in BPM.

Several works have been introduced in business process monitoring to apply machine learning techniques for introducing predictive capabilities to identify the next state given being at a certain state in the process. Some previous research in predictive process monitoring approaches utilized machine learning techniques for predicting the next state of the processes as the research introduced by Tax et al. (2017) [7]. They introduced a technique to predict the next activity of a running case and its timestamp using Long-Short-Term (LSTM) neural networks. They showed that this basic technique can be generalized to address two other predictive process monitoring problems: predicting the entire continuation of a running case and predicting the remaining cycle time. The authors also identified a limitation of LSTM models when dealing with traces with multiple occurrences of the same activity, in which case the model predicts overly long sequences of the same event.

Taking advantage of one of the natural language processing applications, Evermann et al. (2017) [8] presented a novel approach motivated by the idea of predicting the next word in a sentence. By interpreting process event logs as text, process traces as sentences, and process events as words, these techniques can be applied to predict future process. They demonstrated that an explicit process model is not necessary for prediction. Deep learning models, where the process structure is only implicitly reflected, can perform as well as explicit process models. The approach is evaluated on two real datasets and the results surpass the state-of-the-art in prediction precision.

Motivated by past success of reinforcement learning approaches, the research presented by Chiorrini et al. (2020) [9] stands out as a foundational work in the domain of predictive process monitoring using reinforcement learning (RL). Their goal was to study the feasibility of applying reinforcement learning to the field of predictive process monitoring. Through their experiments on the BPI'2012 popular benchmark dataset, they showed that DQN agents can fully exploit time information, achieving results that significantly outperform state of the art approaches based on LSTM architectures.

Pasquadibisceglie et al. (2022) [10] investigates a multi-view deep learning approach to enhance predictive business process monitoring. The authors aim to improve the accuracy and robustness of predictions by integrating multiple views of process data, such as control-flow, dataflow, and resource perspectives, into a unified deep learning model. They used Long Short-Term Memory (LSTM) networks to capture temporal dependencies and Convolutional Neural Networks (CNNs) to extract features from the different data perspectives. The model was trained and tested on several real-life event logs to evaluate its performance. The experiments demonstrated that integrating multiple views of process data provides a more holistic understanding of the process, leading to better predictive performance.

Bousdekis et al. (2023) [11] introduced a reinforcement learning-based method for predictive business process monitoring. They used the inductive miner approach to map the structure of business processes and compute transition probabilities between states to handle uncertainty, providing a foundation for a Q-learning model. The authors then applied the same experiment using Deep Q-network approach that is based on an artificial neural network to estimate the quality for every possible action, given the environment existence at a certain state for comparing the two approaches. The proposed RL-based approach was evaluated in a use case from the banking sector and compared to traditional Q-learning and DQN methods. The results show that Q-learning has better performance in simple problems, while DQN is more suitable for complex problems due to its neural network (parametric) structure.

Kaftantzis et al. (2024) [12] proposed an approach for next activity prediction in business processes using Automated Machine Learning (AutoML), specifically the Tree-Based Pipeline Optimization Tool (TPOT). The methodology involves extracting event logs, preprocessing the data, discovering processes using the Heuristics Miner algorithm, and predicting the next activity using TPOT. The approach was tested on a real-life event log from the Business Process Intelligence Challenge (BPIC'12). Various window sizes were experimented with to determine the optimal size for prediction accuracy. The results showed that the proposed approach achieved higher prediction accuracy compared to other methods in the literature. The optimal window size was found to be 5, balancing accuracy and computational cost. The approach also demonstrated the potential of AutoML to make machine learning accessible to non-experts.

Van Luijken et al. (2024) [13] explores the use of transfer learning to improve the efficiency and accuracy of suffix prediction in predictive process monitoring. This involves predicting the remaining sequence of events in a process instance using deep learning models. The study uses two deep learning architectures, LSTM and GPT, trained on two public event logs. These base models are then fine-tuned on eight different event logs from various domains to evaluate the effectiveness of transfer learning. The results indicate that transfer learning can reduce training time and often improve prediction accuracy compared to models trained from scratch. However, the effectiveness varies depending on the characteristics of the event logs, such as vocabulary size.

Brennig et al. (2024) [14] explores the application of predictive process monitoring knowledge-intensive processes, which are characterized by their complexity and reliance on unstructured data. This study investigates whether incorporating textual information can enhance prediction accuracy. The authors developed a text-aware approach that integrates machine learning and natural language processing. This approach encodes and aggregates event logs by creating features from both structured and unstructured data, while largely neglecting the control flow perspective. The methodology was tested using two real-world event logs, with the aim of predicting the total lead time of ongoing processes. The study found that the text-aware approach significantly improves prediction accuracy compared to state-of-the-art methods. The results highlight the importance of considering unstructured data in predictive monitoring of complex processes.

Several studies have explored techniques in the field of predictive business process monitoring by optimizing business process resource management. These approaches typically leverage machine learning and process mining methods to predict the resources needs in business processes. Neubauer et al. (2022) [15] addresses the challenge of optimizing resource allocation in business processes. Efficient resource allocation is crucial for minimizing costs and maximizing productivity. The authors propose an approach that leverages reinforcement learning and process mining to enhance resource management in dynamic business environments. The approach models the resource allocation problem as a Markovian decision process. The authors apply batch reinforcement learning algorithms, specifically Fitted Q-Iteration and Neural Fitted Q-Iteration, to derive optimal resource allocation policies. These algorithms utilize historical data from event logs to learn and predict the best allocation strategies. The methodology involves integrating process mining techniques to extract relevant features from the event logs and using these features to train the reinforcement learning models. The study demonstrates that the proposed approach significantly improves resource allocation efficiency. The results from real-world business process data show that the method can reduce the time spent on executing tasks by up to 37.2 %. This indicates that reinforcement learning, combined with process mining, provides a robust framework for optimizing resource allocation in business processes.

Durán et al. (2024) [16] focuses on enhancing the allocation and

sharing of resources within business processes to overcome the challenge of optimizing business process resource management. The authors propose a novel approach that integrates rewriting logic with deep learning to predict resource needs and streamline process execution. The approach involves a timed and probabilistic rewrite theory specification to formalize the semantics of business processes. This specification is integrated with an external oracle, implemented as a long short-term memory (LSTM) neural network. The LSTM model predicts the progression of process traces within a given time frame. The methodology includes constructing the deep learning model and integrating it with the Maude system, a high-performance logical framework. The study demonstrates that the proposed approach can effectively predict resource requirements and optimize process execution. The results from several case studies show improvements in execution time and resource occupancy under different parameters. The integration of deep learning with rewriting logic provides a robust framework for predictive monitoring and resource management in business processes.

3. Data preparation

In this section, the event log structure is introduced along with the steps used for preparation. These steps involve defining the available states and the possible actions that can be taken to construct the required environment. The agent interacts with the constructed environment and discovers the structure of the business processes. To effectively guide the agent toward the desired behavior, a reward/penalty function is designed by considering realistic and valid transitions associated with their transition probabilities. [11,17,18].

3.1. Event log structure

The proposed experiment primarily relies on an event log that tracks the activities occurring within the initiated tickets of a specific organization's ticketing system. The event log consists of (3414) tickets extracted from an internal ticketing system. The event log comprises the following fields:

- case ID: incident (ticket) identifier number
- activity: an event that occurs throughout the ticket life cycle
- timestamp: time of event occurrence

Fig. 2 shows a plot of the Directed Flow Graph (DFG) for the events that occur during the life cycle of the initiated tickets, associated with the frequency of each possible transition. Table 1 introduces a description of the available events from the business perspective.

3.2. Selecting a bottleneck state

The first step involves identifying a state considered a bottleneck within the business process. The bottleneck state acts as an absorbing state that the agent should identify and avoid during training. The criteria for selecting this state includes:

- A state that once reached, it cannot be left [19]
- Having a high weight of outgoing edges indicating that it has a high impact within the flow of business processes

After conducting a visual analysis of the DFG graph depicted in Fig. 2, it was observed that the event *Pending Customer Action* is connected to (703) subsequent events distributed among (6) states with a

Table 1
Ticket events description from a business perspective.

Event	Description
Assigned	Initial status of a manually opened ticket.
Open	Initial status of an automatically opened ticket.
Investigating	Analyzing tickets and adding internal comments.
Cleared	When a task reaches its last activity in the life cycle and is successfully resolved.
Closed	Last activity in a ticket life cycle.
Awaiting Change	Waiting for a change request which is required to resolve the ticket.
Pending Close	The business analysis stated that it is not used.
Referred to Vendor	Add ≪ Vendor ≫ field appears.
Temporary Fixed	Temporary solution implemented to address the issue.
Customer Unavailable	Ticket is pending customer info if customer is responsible for carrier scope.
Pending Customer Action	Ticket is pending customer info if customer is responsible for scope.

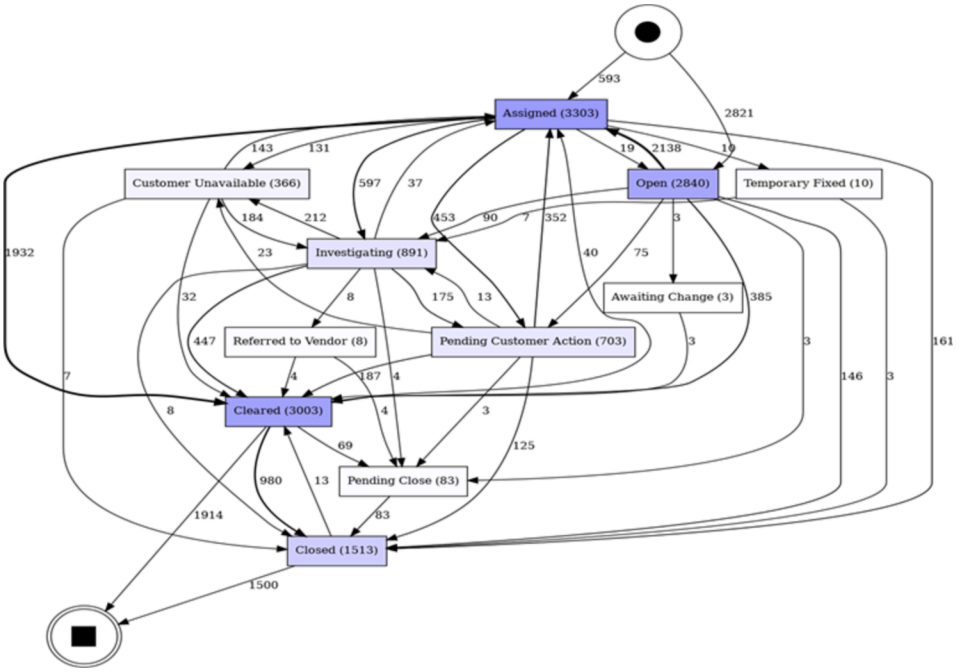

Fig. 2. Raw event log DFG.

frequency of transitions from that state toward the other states, as shown in Fig. 3. In addition to applying the essential preparation steps for the event log, the bottleneck state is introduced as an absorption state based on its definition. Then, the proposed system is designed to predict the optimal sequence of transitions according to the business policy with an avoidance of the predefined bottleneck state for business optimization.

3.2.1. Definition of Pending Customer action events

From a business perspective, the event indicates that a ticket or incident is awaiting attention through a direct contact with the customer to facilitate ticket resolution by gathering essential information related to the issue causing a delay in resolving the issue.

3.3. Applying the absorption state effect

Absorption states have a transition probability of (1) for staying in the same state and a probability of (0) for applying a transition to any other state according to the definition in [19]. In this step, the aim was to convert our event log to include the absorption state effect with respect to the state of interest as defined in section 3.2. All subsequent events that occur after the bottleneck state are removed.

In Fig. 4, the effect of the absorption state is illustrated by the red arrow, indicating that no events occur after that state.

Fig. 5 shows a simple structure for the event log before setting the event '*Pending Customer Action*' as an absorption state (bottleneck), visualized using the Network X package considering that the target state as '***end***'.

3.4. Generating source and target activities from event log

An essential transformation is applied to the event log by associating a 'target activity' column to calculate the transition probability for each unique source/target pair to be incorporated within the design of the reward function. A common termination point (target) is defined as '*end*' state for each ticket (case) that the agent tends to reach. Table 2 shows the applied steps above, considering the anonymization of ticket (case) IDs for data confidentiality.

3.5. Filtering out the short-path cases and calculating the translation probabilities

Tickets with very short event sequences were dropped from the event log to avoid misleading the agent with undesired sequences during the training process while maintaining a realistic sequence of transitions aligned with business logic. Table 3 shows some samples of the eliminated event sequences.

With respect to the filtered sequences after eliminating the short event sequences, the transition probability was computed for each unique source/target transition after applying the above filtration step. For each source state, the number of outgoing edges toward other states is divided by the total number of outgoing edges associated with that state. Table 4 presents the calculated transition probabilities after removing tickets with undesired and short sequences. In the table, a bottleneck state transition is highlighted in red and the transitions toward the '*end*' state are highlighted in green, representing the target of the agent.

4. Methodology

Reinforcement learning is a branch of machine learning, alongside supervised and unsupervised learning. It introduces an intelligent agent capable of applying a sequence of actions based on a learned policy, which serves as a decision-making unit to achieve a specific goal. Reinforcement learning includes techniques for training agents to make decisions by interacting with an environment as shown in Fig. 6.

In our study, the value-based methods which belong to model-free approaches were adopted, considered as one type of the reinforcement learning family including Q-learning and Deep Q-network (DQN). Q-learning consumes the value of actions (Q-values) to determine the best action at each state. A Q-table is used as a reference policy that defines the value of the action given being in a certain state. Deep Q-network (DQN) extends Q-learning by using deep neural networks to approximate Q-values. It aims to enable agents to learn optimal actions in complex, high-dimensional environments where many states and possible actions are available for reducing complexity. The value-based reinforcement learning methods consider the expected long-term return or reward that an agent can gain from a particular state or by performing a certain action. Although these methods do not directly learn the policy, the policy can be derived by choosing the action with the highest value at each state.

Through the environment-agent interaction mechanism in Fig. 6, the agent observes the current state of the environment and selects one of the available actions by referring to its decision-making unit which can be a tabular method, as the case in Q-learning [20,21], or a parametric method, as in Deep Q-network. Based on the action taken, the agent receives a reward or penalty from the environment [18,22,23]. This reward/penalty is predetermined based on what is needed to be maximized (achieved) by the agent and what is intended to be avoided, taking into consideration to maintain a greedy behavior that maximizes the agent's overall cumulative reward toward the desired target.

The agent observes the subsequent state provided by the environment. Through training its decision-making unit, known as the '*policy*', the agent learns to take appropriate actions to reach the expected objective.

Fig. 6 illustrates the reinforcement learning cycle paradigm and the training process of the agent. Through an iterative process, the agent adjusts its behavior to gradually approach the optimal policy representation, and the training episode is designed to terminate whenever the agent reaches the objective or reaches a bottleneck state. Then, the agent starts a new training episode.

In this section, we present the methodologies employed in this paper. These methodologies encompass Q-learning and deep Q-network algorithms. Q-learning is a conventional reinforcement learning algorithm that acquires the optimal policy through iterative updates to a Q-value table while Deep Q-network is a more advanced approach that leverages artificial neural networks to approximate the Q-value function [22,23,24] (See Fig. 7). Both methodologies were applied to the same processed event log data.

4.1. Q-learning algorithm

The Q-learning algorithm is considered a model-free, value-based, and off-policy reinforcement learning algorithm that aims to maximize

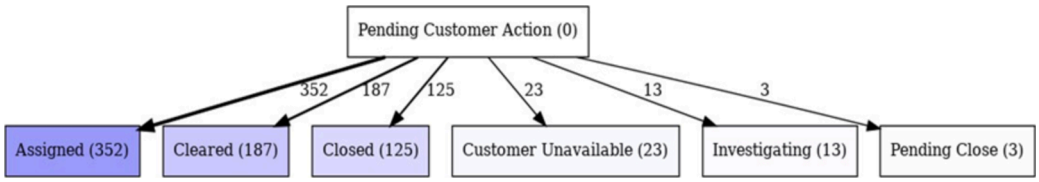

Fig. 3. Pending Customer Action outgoing edges DFG extraction.

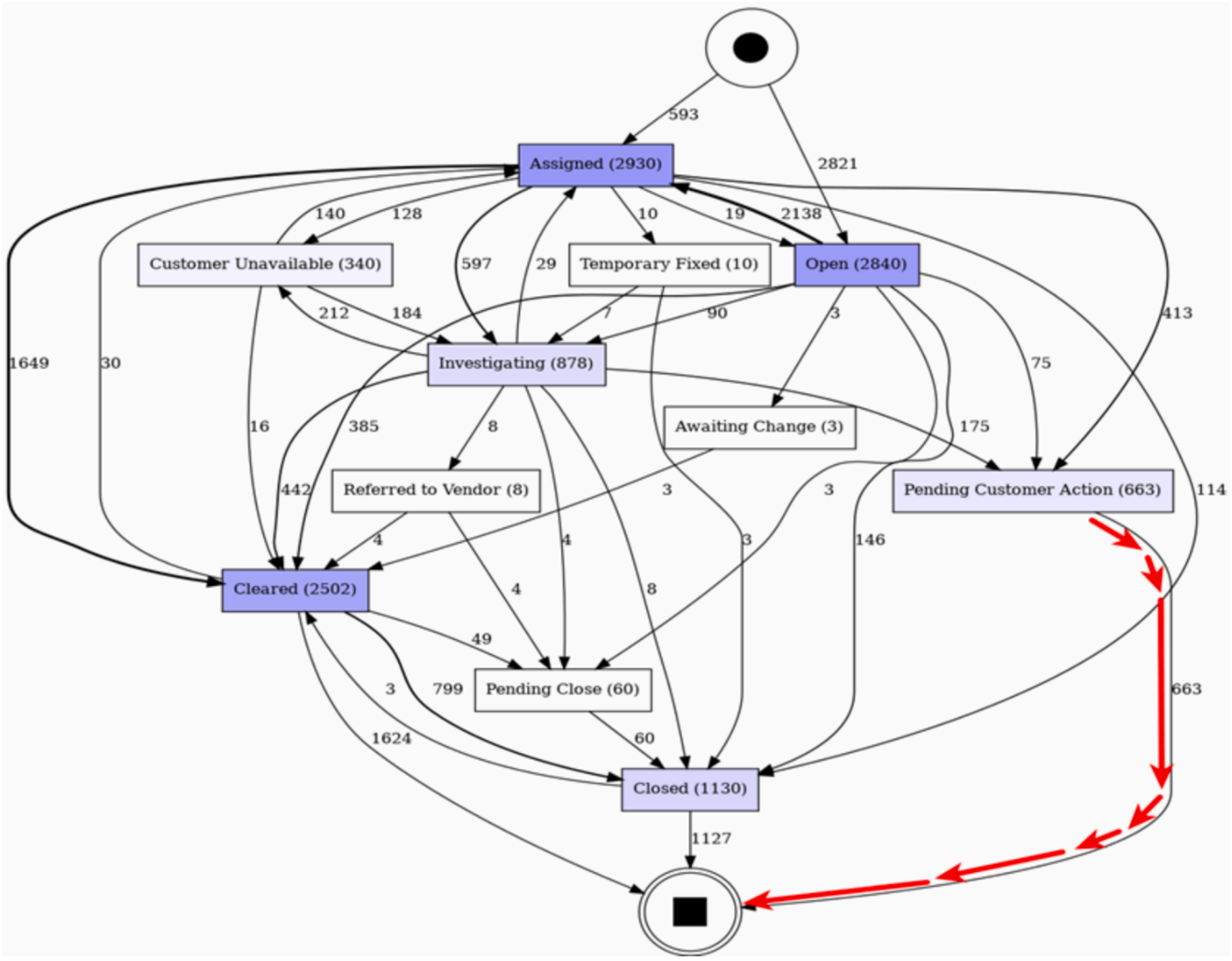

Fig. 4. The event log DFG after setting the absorption state.

the q function [20].

- **Model-free:** the algorithm learns the optimal policy or value function directly from the experience of interacting with the environment, without explicitly modeling the transition probabilities or the reward function.
- **Value-based:** the algorithm learns the optimal value function, such as the state-value function or the action-value function and derives the optimal policy from it.
- **Off-policy**: It refers to a type of algorithm that learns the value of the optimal policy independently of the agent's actions. As such, the strategy (policy) used to generate behavior (action choices) is separate from the policy being evaluated and improved. In our study, the adaptive epsilon-greedy strategy was used, which is a variation of the epsilon-greedy policy applied in reinforcement learning to balance exploration and exploitation. In an epsilon-greedy policy, with probability epsilon, the agent selects a random action (exploration), and with probability greater than epsilon, it selects the best-known action (exploitation) based on the current Q-value estimates. In the adaptive version, the value of epsilon is not fixed but decreases over time, typically as the agent becomes more confident in its value estimates. This means that as learning progresses, the agent explores less and exploits more, under the assumption that it has learned sufficiently about the environment to make good decisions. The exploration factor (epsilon) was updated based on a decaying factor as shown in Equation (1).

$$epsilon = \max(epsilon_{minimum}, e^{-epsilon_{decay} * episode} \quad (1)$$

Where:

- epsilon: The exploration rate used in the epsilon-greedy policy
- epsilon minimum: The minimum value that epsilon can take and prevents epsilon from reaching zero, ensuring that there is always some exploration
- epsilon decay: A positive constant that determines the rate at which epsilon decreases
- episode: The current episode number

Q-learning aims to build an agent capable of reaching a specific objective by learning an action-value function $Q_\pi(s, a)$ that can estimate the cumulative reward value if the agent starts at a certain state, takes an action, and then follows the policy accordingly [18,25].

Equation (2) defines the Q-learning action-value function as the expected cumulative discounted reward value starting from state (s), taking an action (a), and then following the policy, represented as goal (G) [23,26]:

$$Q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi\left[\sum_{t=0} \gamma^t r_t|S_t = s, A_t = a\right] \quad (2)$$

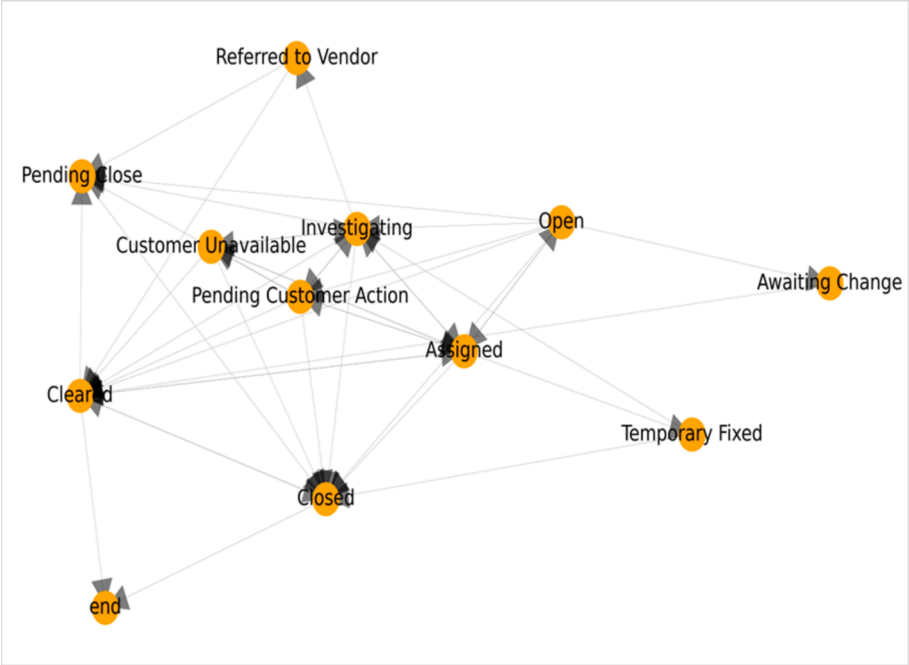

Fig. 5. The event log DFG using Network X package.

Table 2
Each source/target pair associated with the new terminal state (target) with anonymized case IDs.

case ID	source activity	source timestamp	target activity	target timestamp
beec43xa	Open	2022–10–06 03:11:03	Assigned	2022–10–06 03:20:24
beec43xa	Assigned	2022–10–06 03:20:24	Cleared	2022–10–06 04:06:19
beec43xa	Cleared	2022–10–06 04:06:19	Closed	2022–10–30 08:10:32
beec43xa	Closed	2022–10–30 08:10:32	end	2022–10–30 08:10:32
ceec44xb	Open	2023–01–23 02:16:08	Assigned	2023–01–23 02:19:03

Table 3
Sample of eliminated tickets with short sequences.

CaseID	Event
aa2947ax	(Open, Cleared)
aa2948ay	(Assigned, Closed)
aa2949az	(Open, Cleared, Closed)

Where:

- **Policy** π: the policy followed as a greedy policy to select the action that maximizes the agent's cumulative reward for reaching an optimal action-value function Q^* (s, a)
- **Discount Factor** γ: the discount factor that determines how far future rewards are considered in the return value

In Q-learning, the agent learns an optimal action-value function, $Q^*(s, a)$, encoded as a Q-table, which determines the expected reward for taking a particular action in each state. The Q-value function is updated iteratively, where the new Q-value is a combination of the old Q-value and the expected discounted reward that can be obtained from the next state according to the Bellman Equation. The equation considers the application of a greedy policy that aims to take actions for maximizing the agent's cumulative reward [23,26]. Equation (3) represents the formula used to update the Q-value during the training process of the agent in a simple form.

$$Q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi\left[\sum_{t=0}\gamma^t r_t|S_t = s, A_t = a\right] \quad (3)$$

The Q-table is continuously updated until an optimal table is achieved, enabling the identification of the best action for each state within the table. The agent learns an optimal policy that maximizes its expected reward over time. The reward function assigns an immediate numerical value for each action taken by the agent, and the learning rate determines the speed at which the agent updates its Q-value function. The state space includes all possible states in which the agent can exist whereas the action space includes all possible actions the agent can take in each state. The Q-learning algorithm follows a straightforward principle: the agent selects an action in given state, receives a reward, observes the new state, and updates its Q-value function accordingly following Equation (3).

4.1.1. *Q-table Initialization*

The Q-table for the proposed use case is composed of 12 states, according to the event log of the ticketing system for an organization. Each state represents a particular event that can occur within the ticket life cycle until its termination. The actions taken by the agent are defined as transitioning from one state to another. The Q-table represents the available states in rows, the possible actions in columns, and the table cells refer to the learned Q-value for each possible state-action pair during the agent's training. At the beginning of the training, the Q-table is initially set to an arbitrary value, typically zero.

4.1.2. *Reward function design*

To effectively guide the agent toward the target of interest, it is crucial to formulate a specific reward function that maximizes desired outcomes and penalizes undesired ones [26]. The objective is to guide the agent to reach the terminal state *end*, while avoiding the bottleneck state through valid transitions that hold a series of transitions that are aligned with business rules. The reward function is designed to guide the agent to behave according to the following criteria as shown in Table 5:

Table 4
Transition probabilities for each source/target pair after the preparation steps. It can be observed that the bottleneck state can only be transitioned to itself as a self-loop with a transition probability of 1, according to the effect of the absorption state. In addition, there exist two states that can be transitioned to the target ('end') state: (Cleared, Closed).

Source Activity	Target Activity	Transition Probability
Assigned	Cleared	0.104107
Assigned	Customer Unavailable	0.122254
Assigned	Investigating	0.362942
Assigned	Open	0.009551
Assigned	Pending Customer Action	0.394460
Assigned	Temporary Fixed	0.006686
Awaiting Change	Cleared	1.000000
Cleared	Assigned	0.032258
Cleared	Closed	0.407625
Cleared	Pending Close	0.073314
Cleared	end	0.486804
Customer Unavailable	Assigned	0.415430
Customer Unavailable	Cleared	0.038576
Customer Unavailable	Investigating	0.545994
Closed	end	1.000000
Investigating	Assigned	0.036683
Investigating	Cleared	0.338118
Investigating	Customer Unavailable	0.333333
Investigating	Pending Customer Action	0.279107
Investigating	Referred to Vendor	0.012759
Open	Assigned	0.831021
Open	Awaiting Change	0.003783
Open	Investigating	0.070618
Open	Pending Customer Action	0.094578
Pending Close	Closed	1.000000
Pending Customer Action	Pending Customer Action	1.000000
Referred to Vendor	Cleared	0.500000
Referred to Vendor	Pending Close	0.500000
Temporary Fixed	Investigating	1.000000

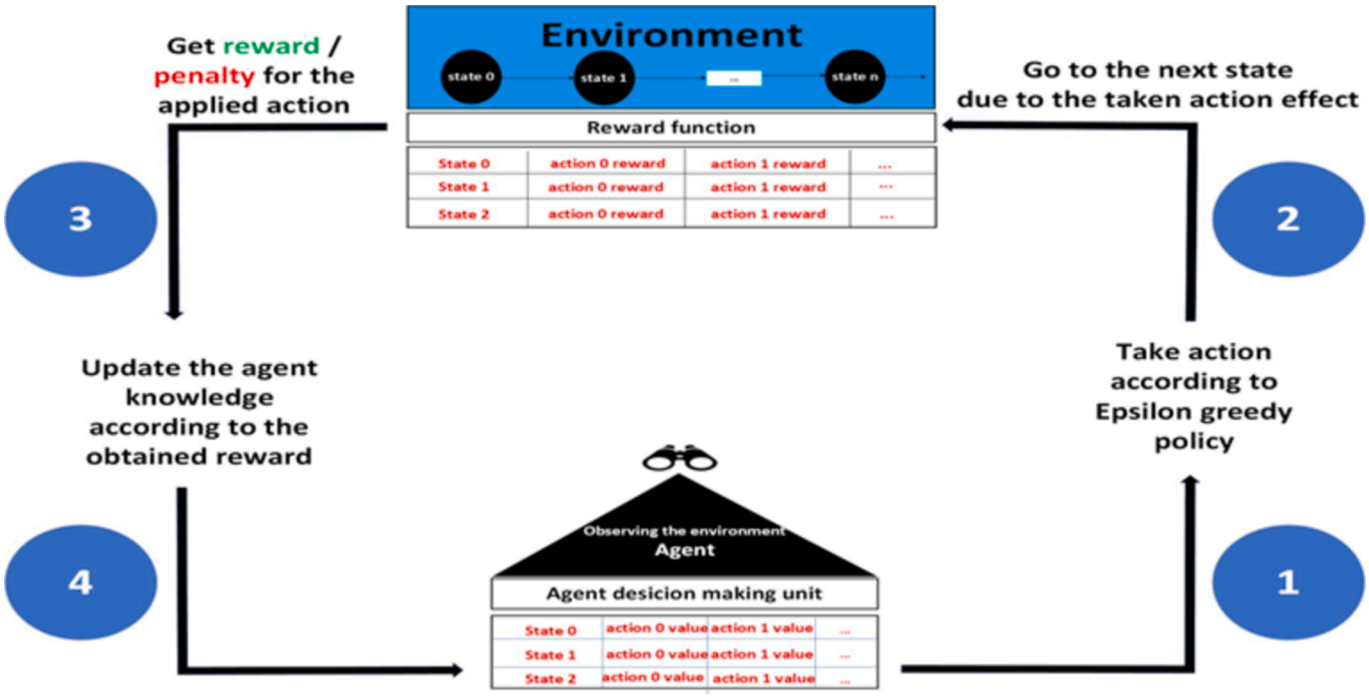

Fig. 6. Reinforcement learning cycle paradigm.

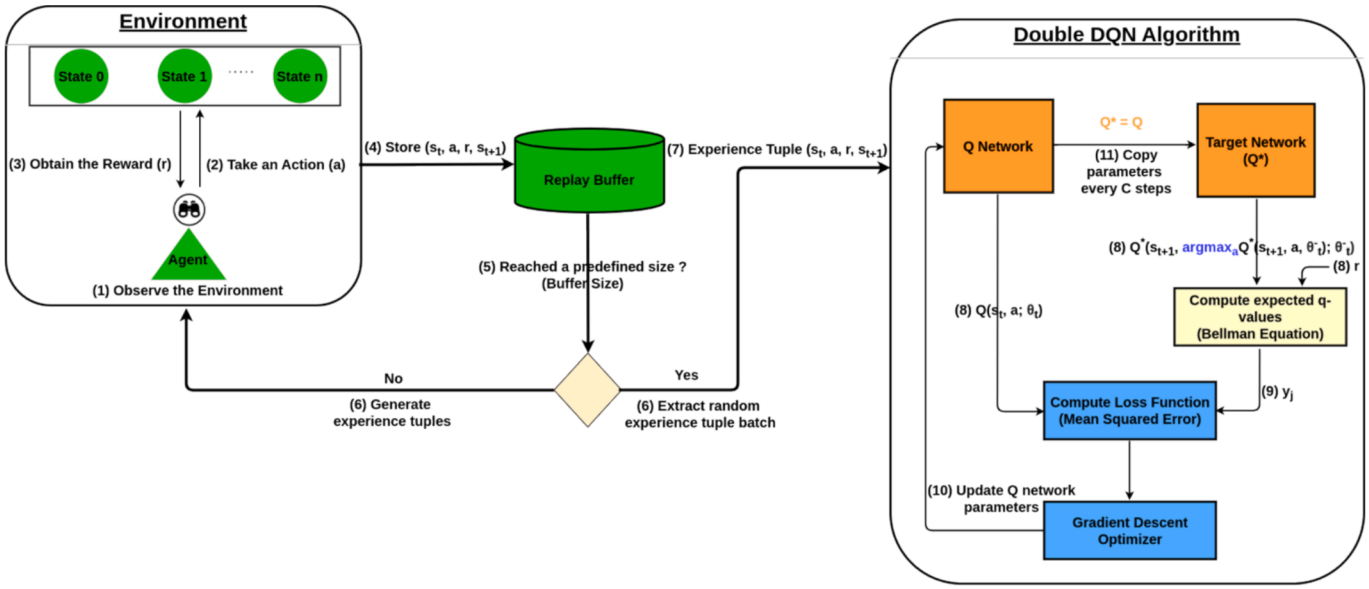

Fig. 7. Double Deep Q-network algorithm.

Table 6 shows the overall reward matrix applied in Q-learning in more detail. It can be observed that all invalid transitions are penalized with a score of −1. Conversely, valid transitions are assigned a value corresponding to their respective transition probabilities. The target transition is rewarded with the highest value 10, which can be observed for the transition from state (Cleared or Closed) to the (end) state.

4.1.3. Training process

To achieve the optimal action-value function that represents the optimal policy toward the target, at the beginning of training, the Q-table is initialized with zeros. On each episode, the environment is designed to specifically set the agent to start its path toward an end alternatively between the states 'Assigned' or 'Open' with a probability distribution of 0.5.

Then, by applying the training for a certain number of episodes (4000), at each time step, a uniformly distributed value (p) is generated and compared to an epsilon ∈ value where $0 < \in < 1$ in order to control the action selection process and to balance between the exploration of the possible actions to be taken at each time step, and the exploitation of the already learned knowledge according to the following criteria [18]:

$$Taken\ action(a) = \begin{cases} Select\ a\ random\ action\ if\ p < \in \\ Act\ with\ \max imum\ Q - value\ else\ p \geq \in \end{cases}$$

The exploration factor ∈ initially has a value of 1.0, indicating a full exploration behavior. The exploration rate is gradually reduced over time using an exponential decay factor of (0.001). This gradual reduction allows the agent to rely more on its learned knowledge and follow a greedy policy to maximize its reward by exploiting its already learned policy in the Q-table. When the agent encounters a certain state, it selects an action based on the exploration–exploitation criteria mentioned earlier, and then it observes the resulting next state and immediate reward. The q-value for the corresponding state-action pair is updated according to the mentioned Equation (3). Throughout the training, outlined as a pseudocode in Listing 1, the agent learned to make sequential decisions with the goal of maximizing the reward over an episode, which was a sequence of states, actions, and rewards that concluded at a terminal state.

Table 10 presents the updated Q-table after the completion of training. The Q-table shows the learned optimal policy that defines the best transition action to be taken according to the associated maximum Q-value per state. According to the resulting Q-table after the training, it was found that whenever the initial state is'Open', the expected sequence of transitions is**'Open' —'Assigned' —'Cleared' —'end'** based on the exploitation of the maximum computed Q-values given each state.

Algorithm 1: Q-learning algorithm.

```
Input: env, episodes, max iter per episode, adaptive epsilon
Initialize empty lists: rewards per episode, regrets per episode, epsilon per episode
for idx = 0 to episodes do
    Initialize state by resetting the environment with initial state name "Assigned"
    Initialize done as False
    for step = 0 to max iter per episode do
        action = choose action based on current state
        next state, reward, done, info = execute action in environment
        update agent state, action, reward, and next state
        state = next state
        if done is True then
            break the loop
        end if
    end for
    Print 'Episode:', episode index + 1,'Total reward:', total reward,'Regret:', total regret
    Append total reward to rewards per episode list
    Append total regret to regrets per episode list
    Reset total reward and total regret attributes to 0
    if adaptive epsilon is True then
        Append current epsilon to epsilon per episode list
        Update epsilon using the exponential decay formula if it is greater than epsilon min
    end if
end for
return Q, rewards per episode, regrets per episode, epsilon per episode
```

4.2. Deep Q-network

A Deep Q-network (DQN) is a specific reinforcement learning algorithm that combines Q-learning with deep neural networks [23]. DQN falls under the same category of the Q-learning from the perspective of being a model-free, value-based and off-policy RL algorithm, and those concepts are explained in section 4.1. It learns the optimal policy through a direct interaction with the environment using the observed rewards and transitions, without needing a model of the environment's dynamics with a greedy policy during the agent acting that selects the action with the highest Q-value given a certain state towards maximizing the agent's overall reward overtime, another policy is applied

Table 5
Q-learning reward design.

Transition	Reward value
Invalid transition	-1
Absorption transition	-1
Valid transition	**Transition Probability**
Target transition	**10**

Where:

- **Invalid transition**: a transition between two states with no direct connection between them in the event log. Such transitions incur a penalty of (−1) indicating the undesired behavior.
- **Absorption transition:** a transition toward the bottleneck state.
- **Valid transition**: a transition between two states with an actual and direct connection between them. Such transitions are rewarded with an immediate value as the transition probability representing the likelihood of transitioning between those states, which is the desired behavior.
- **Target transition**: The transition toward the target '*end*' state. Such transitions are given a relatively significant reward to motivate the agent toward that objective.

Table 6
Q-learning algorithm reward matrix: Red (invalid/bottleneck) transition, Yellow (valid) transition, and Green (target) transition.

	Assigned	Awaiting Change	Cleared	Closed	Customer Unavailable	Investigating	Open	Pending Close	Pending Customer Action	Referred to Vendor	Temporary Fixed	end
Assigned	-1.000	-1.000	0.104107	-1.000	0.122	0.363	0.010	-1.000	-1.000	-1.000	0.007	-1.000
Awaiting Change	-1.000	-1.000	1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000
Cleared	0.032	-1.000	-1.000	0.408	-1.000	-1.000	-1.000	0.073	-1.000	-1.000	-1.000	10
Closed	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	10
Customer Unavailable	0.415	-1.000	0.039	-1.000	-1.000	0.546	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000
Investigating	0.037	-1.000	0.338	-1.000	0.333	-1.000	-1.000	-1.000	-1.000	0.013	-1.000	-1.000
Open	0.831	0.004	-1.000	-1.000	-1.000	0.071	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000
Pending Close	-1.000	-1.000	-1.000	1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000
Pending Customer Action	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000
Referred to Vendor	-1.000	-1.000	0.500	-1.000	-1.000	-1.000	-1.000	0.500	-1.000	-1.000	-1.000	-1.000
Temporary Fixed	-1.000	-1.000	-1.000	-1.000	-1.000	1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000

during the environment exploration based on the epsilon greedy policy that utilizes an adaptive exploration–exploitation factor (epsilon) with a decaying factor to balance between the agent's selection for new actions (exploration) and overtime the agent starts to exploit the learnt knowledge according to the Q-function to exploit those actions (off-policy).

DQN combines the power of artificial neural networks (ANNs) into the concept of Q-learning to approximate the quality of all actions, given being in a certain state (value-based approach).

The Q-values (quality) for all actions are computed at once given a certain state using the neural network weights (parameters) at every step within the training episode until reaching the termination state of the episode instead of taking every (state-action) pair as an input to the Q-table to estimate the Q-value for that particular action at that specific state causing a higher complexity and time until reaching the optimal policy especially in case of large number of states and actions within the environment of interest [22,24].

DQNs are designed to handle high-dimensional state spaces that are typically found in real-world problems in which the environment involves large number of states and possible actions to be taken by the agent causing an increased computational complexity for using the traditional tabular Q-learning approach [23].

In our study, our event log data involves a few numbers of discrete states and actions that formulate the environment and that our task complexity is not very high. By applying the experience replay concept within our DQN model, the model could learn from a diverse range of past experiences, which can indirectly help the agent infer the importance of historical information to break temporal correlations between

consecutive experiences, leading to more stable and robust learning, which could be beneficial in dynamic environments like business processes.

The Experience replay technique was applied with defining the replay buffer of size (128) to sample randomly a batch size of 128 from the past experiences for updating the neural network (Q-function) weights during the training to help capture relations depicted within past experiences with a relatively sufficient size as a memory for the agent so as not to forget important relations between the events.

Utilizing these technical concepts as for the experience replay and target networks in DQN have shown their effectiveness to stabilize training and improve convergence in practice that are explained in detail in the following subsections. Fig. 7 summarizes the applied approach for the double deep Q-network algorithm.

4.2.1. Experience replay

Experience replay in the context of Deep Q-networks (DQN) can help provide a mechanism for handling unbiased sampling in which the agent learns from a more representative sample of past experiences, reducing the bias towards recent transitions. Also, the concept allows for handling the temporal correlations between successive experiences, which helps the agent learn about the long-term dependencies between actions and delayed rewards by sampling experiences randomly from the replay buffer. The agent can repeatedly sample and learn from important past experiences, including those with long-term consequences, which might be infrequent in more recent experience. We explain the concept in detail in this section.

At each time step (t), the agent's experience is represented in the form of (state(t), action(t), reward(t + 1), state(t + 1), done). Done indicates whether the episode has terminated in case the agent has reached either the target or the bottleneck state. These experience tuples are stored in a replay buffer of a predefined size, allowing the agent to reuse past experiences during the training and prevent the loss of valuable knowledge [21,22,23]. At the start of training, the target network is initialized with the same parameters as the Q-network, and the replay buffer is initialized with a specific size. Using an epsilon factor probability, a uniformly distributed value between 0 and 1 is generated. If this *random value* < *epsilon factor*, a random action was chosen, enabling the agent to explore the action space of the environment (exploration). However, if the random value was greater than or equal to the epsilon factor, the action with the highest q-value was selected, allowing the agent to exploit the learned policy and take actions that maximize its reward (exploitation).

Based on these action selection criteria, experience tuple samples were generated and stored in the replay buffer until it reached its predefined threshold exceeding the buffer of the batch size of 128. Subsequently, a mini-batch of experience tuples is randomly selected from the buffer to train the Q-network and estimate the q-values based on the selected actions within the previous experience replay.

4.2.2. Target network

To train the neural network to approximate the q-values, a second network called the target network that has the same architecture as the primary Q-network is defined to ensure stable training, avoid oscillation, and compute the loss function between the predicted Q-values of the states sampled from the replay memory and the expected Q-target values of the corresponding next states [20,22,23]. The loss function was optimized using the gradient descent algorithm to update the Q-network parameters and improve the Q-value predictions.

During the training of the Q-network used for predicting the Q-values, the network parameters were updated during the optimization process using the gradient descent algorithm. To stabilize the training, a separate target network is used, which is updated less frequently than the Q-network. The Q-network parameters are copied to the target network after a certain number of episodes (every 10 episodes) to ensure a fixed and stable estimation of target values, leading to a stable and optimal solution [25].

The expected Q values of the target network is calculated as follows [22,23]:

$$y^i = r_i + \gamma \max{}' Q(\phi, a'; \theta^-) \tag{4}$$

Where:

- The q-target value y^j is set as the maximum q-value for a certain action given the next state ($\phi_i + 1$) from the target network, discounted by the gamma factor γ (how much we value future rewards) plus the immediate reward for taking that action within the current state, as mentioned in Equation (4)
- Otherwise, the q-target value y^j is estimated as the immediate reward r_i for taking an action at instant *i,* in case the episode terminates at the next step ($i + 1$).
- Y^i: the target network Q-value estimation.
- θ^-: the target network weights (parameters).
- γ: the discount factor that determines the importance of future rewards. It is a value between [0, 1], where 0 means only considering immediate rewards and 1 means considering all future rewards equally.
- ϕ: the available state space.
- *i*: current step or time index in the episode.

4.2.3. Reward function design

To apply the Deep Q-network algorithm to the process mining event log, a specific environment was defined as a class called ProcessMiningEnvironment that was inherited from the Gym package. This class was used to define the space of available states and actions, as well as the immediate rewards for taking actions according to the business process structure identified in section 3.4, such that the environment was designed with a criteria to assist the agent in reaching a specific target by maximizing the reward for certain actions given certain states. The criteria for designing the rewards are given in Table 7.

Each case in the table above, is defined as follows:

- **Invalid transitions**: invalid transitions show that the agent selected to apply a transition from one state to another that is not available or not applicable within the event log.
- **Absorption state transition:** a transition towards the bottleneck state.
- **Valid not of interest transitions**: a valid and not of interest transition shows that the agent. selected to apply an applicable transition but toward any possible subsequent state other than the one with the highest transition probability.
- **Valid state of interest (Normal) transitions**: The transition from a normal state to another, that has a direct connection to it and is the transition to the state of the highest transition probability among the other possible valid states. This type of transition takes a reward of 0.1.
- **Target (Terminal) Transitions**: Transition from a normal state to the '*end*' state. This transition is assigned a relatively significant reward of 100, which motivates the agent to head toward the target state.

The intuition behind designing a distinct reward function for the Deep Q-network as opposed to the Q-learning stems from the differences in how each algorithm updates its policy. After conducting many experiments, it was concluded that employing the same reward matrix as Q-learning leads to sequences that encompass invalid states not aligned with the business process. By using distinct reward matrix for Deep Q-network shown in Table 8, the agent can traverse valid states until reaching a terminal state.

According to Table 8, the red color refers to either an invalid transition or not of interest transition that is not toward the state with the

Table 7
DQN algorithm reward design.

Case	Immediate reward
Invalid transitions	-10
Absorption state transitions	-10
Valid not of interest transitions	-10
Valid of interest transitions (normal)	0.1
Target transitions	100

maximum transition probability as stated in the reward design within Table 7, the yellow color refers to the valid of interest transitions, and the green color shows the target transition.

Table 9 presents the applied artificial neural network architecture, which consists of three layers with dimensions (12, 32), (32, 32), and (32, 12) for the input, hidden, and output layers respectively. These dimensions are determined by the presence of 12 states, including the bottleneck and terminal state, as well as 12 possible actions that correspond to transitioning toward those states. Table 10 summarizes the DQN model parameters that include trainable and non-trainable params along with the input sizes and forward/backward passes.

Table 9
DQN Q-Network and Target Network Architecture.

Layer	Output Shape	Param
Linear-1	[-1,12,32,32]	416
Linear-2	[-1,12,32,32]	1056
Linear-3	[-1,12,32,12]	396

Table 10
Summary of DQN Model Parameters.

Total params	Trainable params	Non-Trainable params	Input size (MB)	Forward/ Backward Pass size (MB)	Total size (MB)
1868	1868	0	0.02	0.22	0.25

5. Results

This section introduces the results of applying reinforcement learning techniques based on Q-Learning and Deep Q-Network, to predictive business processes, along with a comparison of the results obtained from both methods.

5.1. Q-learning approach results

This section provides an analysis of the results obtained from our

Table 8
Deep Q-network reward matrix: Red (invalid/bottleneck) transition, Yellow (valid) transition, and Green (target) transition.

	Assigned	Awaiting Change	Cleared	Closed	Customer Unavailable	Investigating	Open	Pending Close	Pending Customer Action	Referred to Vendor	Temporary Fixed	end
Assigned	-10	-10	-10	-10	-10	**0.1**	-10	-10	-10	-10	-10	-10
Awaiting Change	-10	-10	**0.1**	-10	-10	-10	-10	-10	-10	-10	-10	-10
Cleared	-10	-10	-10	0.1	-10	-10	-10	-10	-10	-10	-10	**100**
Closed	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	100
Customer Unavailable	-10	-10	-10	-10	-10	**0.1**	-10	-10	-10	-10	-10	-10
Investigating	-10	-10	0.1	-10	-10	-10	-10	-10	-10	-10	-10	-10
Open	**0.1**	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10
Pending Close	-10	-10	-10	0.1	-10	-10	-10	-10	-10	-10	-10	-10
Pending Customer Action	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10
Referred to Vendor	-10	-10	0.1	-10	-10	-10	-10	-10	-10	-10	-10	-10
Temporary Fixed	-10	-10	-10	-10	-10	0.1	-10	-10	-10	-10	-10	-10
end	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10

experiments with Q-learning algorithm. According to Table 11, the study concluded that Q-learning was able to produce the optimal sequence of transitions in the form of '***Open***' → '***Assigned***' → '***Cleared***' → '***end***' that is mostly aligned with the business process with the avoidance of the bottleneck state. By exploiting the resulted Q-table given that the initial state is ('Open'), the next possible state transition is identified through the selection of the state with the maximum Q-value that is 'Assigned' with a value of 1.52, followed by state 'Cleared' with a value of 2.57 then 'end' of value 5.

By the end of the training, the epsilon greedy policy causes the bottleneck state, 'Pending Customer Action' and the target state 'end' as source states, to remain at their initial state with keeping their values set to zero (initial values). Whenever the agent chooses to apply a transition toward either the bottleneck or the target state, the episode instantly terminates, and the Q-table values are no longer changed for both states.

The Heuristic Miner process discovery algorithm was used on the preprocessed event log to assess the optimal path output produced by the Q-Learning approach. As shown in Fig. 8, it was found that the optimal transition sequence derived from the Q-learning model aligns with the optimal path identified by the heuristic miner. This demonstrates that the Q-learning model can produce feasible and realistic paths for the business process of the ticketing system.

Fig. 9(a) illustrates the rewards per episode for the Q-learning agent. The graph displays a gradual increase in rewards over time, suggesting that the agent is learning to select appropriate actions to maximize its total reward through applying an adaptive exploration factor (epsilon) that balances between the exploration of new possible actions and exploitation of the already learnt policy. These actions involve transitioning from one state to another while avoiding the bottleneck state, "Pending Customer Action," and aiming for the terminal state, "end," which signifies the completion of a ticket. The maximum reward is maximized at 10, as per our reward matrix, and the agent appears to converge to optimal behavior after approximately (1500) out of the (4000) episodes. The overall model training time is approximately equal to (2.04) seconds, which shows promising performance and relevance for the use case.

Fig. 9(b) depicts the loss function, defined as the regret function, which presents the difference between the maximum reward of the next state and the reward of the current state, aggregated per step of each episode. Initially, the agent frequently selects suboptimal actions, as indicated by the high regret values for approximately the first (600) episodes. However, as the training continues, the regret decreases, showing an improvement in the agent's policy.

5.2. Deep Q-network approach results

This section provides an analysis for the results obtained from our experiments with Deep Q-network algorithm. Table 12 presents the approximated Q-values extracted from the output layer of Deep Q-network after the model being trained to construct the final Q-table with dimension of 12 states by 12 actions. According to Table 12, we concluded that Q-learning was able to produce the optimal sequence of transitions in the form of **'Open'** → **'Assigned'** → **'Investigating'** → **'Cleared'** → **'end'** that is mostly aligned with the business process with the avoidance of the bottleneck state. By exploiting the resulted Q-network weights given that the initial state is ('Open'), the next possible state transition is identified through the selection of the state with the maximum Q-value that is 'Assigned' with a value of 98.20, followed by state 'Investigating' with a value of 98.26, followed by state 'Cleared' with a value of 99.18 then 'end' of value 99.55.

As with Q-learning, the epsilon greedy policy causes the bottleneck state, 'Pending Customer Action' and the target state 'end' as source states, to remain at their initial values. Whenever the agent chooses to apply a transition toward either the bottleneck or the target state, the episode instantly terminates, and the corresponding approximated Q-values of both states are no longer changed.

By referring to the bottleneck state column 'Pending Customer Action' (highlighted in red), it holds the smallest values of all columns as the agent learned to avoid reaching this step due to the design of the reward adopted for DQN. The target state column 'end' has the highest

Table 11
Q-Table Result after Training.

	Assigned	Awaiting Change	Cleared	Closed	Customer Unavailable	Investigating	Open	Pending Close	Pending Customer Action	Referred to Vendor	Temporary Fixed	end
Assigned	0.510	0.394	**2.527**	1.358	0.437	1.007	0.423	0.165	-0.465	0.287	0.417	-0.442
Awaiting Change	0.064	-0.039	2.672	0.119	-0.110	-0.011	-0.064	-0.035	-0.145	-0.040	-0.089	-0.152
Cleared	0.904	0.273	1.307	1.886	-0.044	0.460	0.033	0.611	-0.397	0.347	-0.002	**5.000**
Closed	0.027	-0.041	0.390	0.190	-0.116	-0.035	-0.084	-0.054	-0.173	0.021	-0.058	4.956
Customer Unavailable	1.329	-0.042	0.276	0.145	-0.047	0.120	-0.074	-0.051	-0.138	-0.026	-0.121	-0.131
Investigating	0.334	-0.031	2.602	0.309	0.072	0.014	-0.074	-0.063	-0.198	0.126	-0.076	-0.179
Open	**1.520**	0.057	0.270	0.252	-0.096	0.084	-0.143	-0.053	-0.123	-0.017	-0.069	-0.100
Pending Close	0.071	-0.029	0.108	2.487	-0.069	-0.048	-0.072	-0.070	-0.166	-0.043	-0.094	-0.186
Pending Customer Action	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Referred to Vendor	0.003	-0.040	2.530	0.107	-0.065	0.025	-0.051	0.144	-0.115	-0.065	-0.050	-0.123
Temporary Fixed	-0.051	-0.045	0.306	0.262	-0.090	1.536	-0.070	-0.034	-0.131	-0.047	-0.120	-0.138
end	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

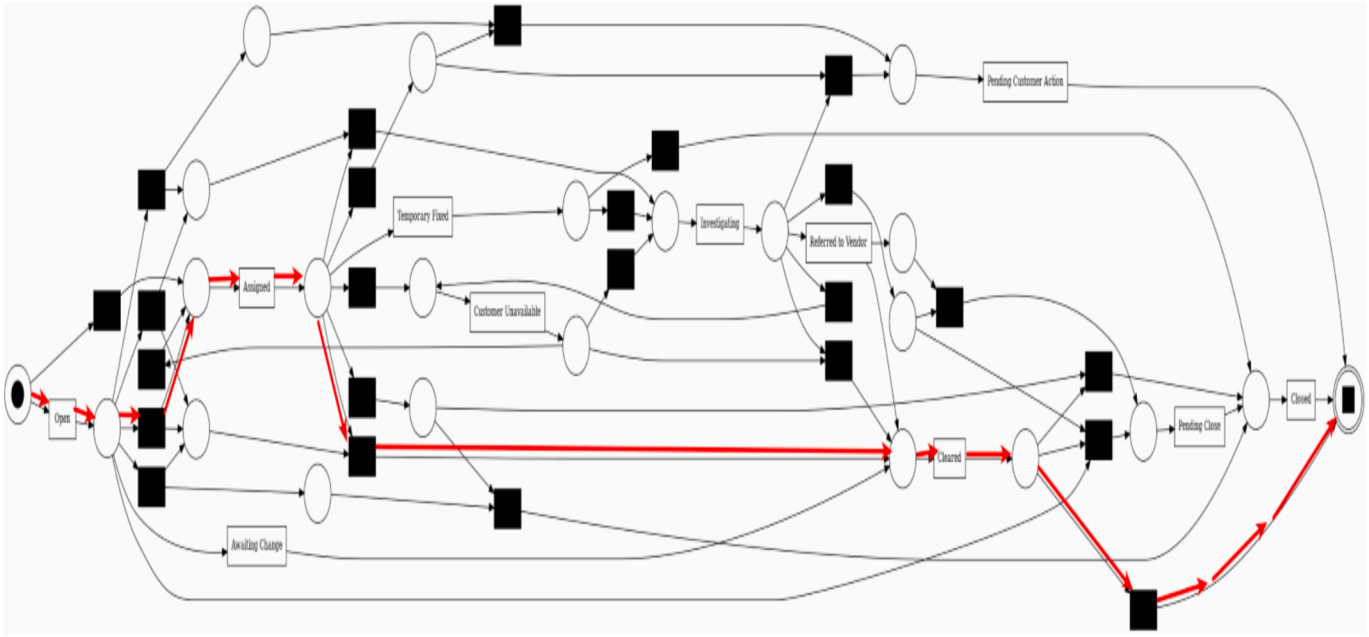

Fig. 8. Optimal path generated by the heuristic miner process discovery approach.

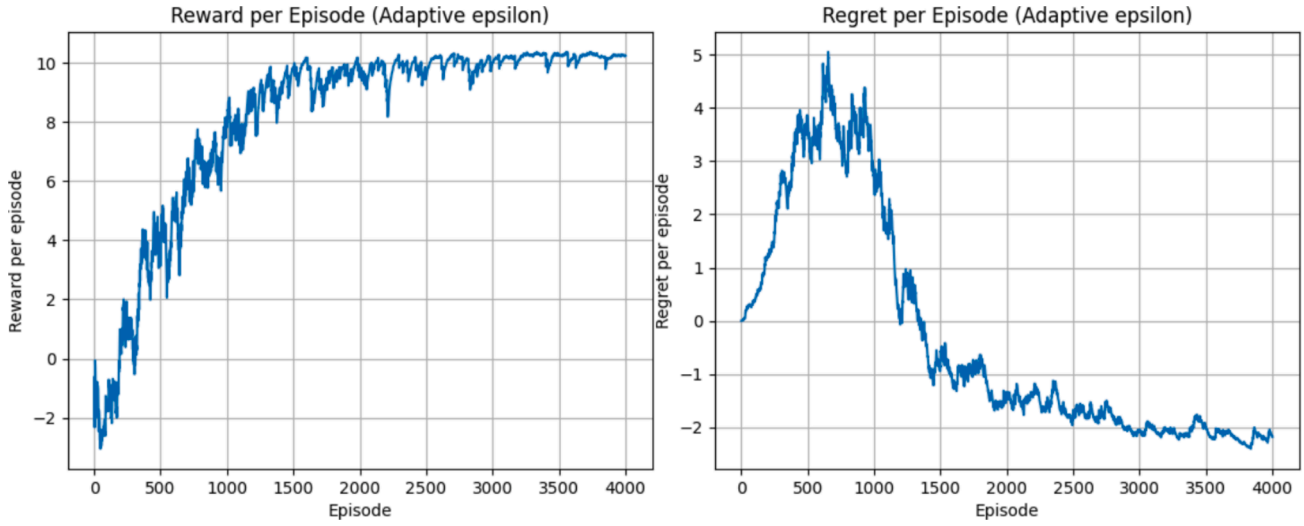

Fig. 9. (a) Reward per episode (Q-learning), (b) Regret per episode (Q-learning).

Q-value with respect to the two states 'Cleared' and 'Closed' as these are the only possible states to occur before the ticket termination.

Fig. 10(a) typically illustrates the agent's performance over time, showing how rewards are accumulated as the agent learned to reach the maximum reward value. The regret function for the DQN agent decreases sharply as shown in Fig. 10(b) indicating the agent can learn and make better decisions over time. The regret can be defined as the difference between the maximum approximated Q-value of the next state and the approximated Q-value of the current state, aggregated per steps of each episode. On the training of DQN, the agent appears to converge to optimal behavior after approximately (2000) out of the (4000) episodes.

The impact of using the adaptive epsilon factor is illustrated in Fig. 10 (a and b) as the agent shifts from exploring the environment randomly to exploiting the knowledge it has gained, which is reflected in the Q-values. This shift should ideally lead to an increase in the accumulative reward as the agent starts to take more optimal actions. By gradually reducing epsilon, the agent is less likely to take random actions that could lead to lower rewards and is more likely to follow the optimal policy that has been learned.

5.3. *Comparative results*

This section shows quantitative measures to compare the performance between the two approaches. We computed the average number of steps per episode per model, the average time duration consumed per episode per model, the total training time duration per model in addition to the average reward value obtained per episode per model as shown within Table 13.

According to the generated performance per model, the overall model training time of DQN took about 86.44 s that is higher than that of the Q-learning model. As such Q-learning was able to converge faster to the optimal path. However, both approaches were able to reach the optimal paths aligned with the business process.

Table 12
Deep Q-network (Q-values) after Training.

	Assigned	Awaiting Change	Cleared	Closed	Customer Unavailable	Investigating	Open	Pending Close	Pending Customer Action	Referred to Vendor	Temporary Fixed	end
Assigned	87.79	88.4	89.13	89.01	87.90	**98.26**	86.47	88.72	-10.02	88.07	87.44	-10.20
Awaiting Change	87.34	88.18	98.76	88.45	87.80	87.76	86.05	88.22	-10.03	87.66	87.01	-10.46
Cleared	88.12	88.45	89.11	99.13	88.16	87.98	86.48	88.63	-10.13	88.26	87.29	**99.55**
Closed	88.19	88.63	89.23	89.19	88.34	88.16	86.80	88.75	-10.07	88.51	87.29	**99.88**
Customer Unavailable	88.25	88.63	89.34	88.94	88.08	98.43	86.62	88.88	-10.06	88.25	87.59	-10.42
Investigating	87.73	88.52	**99.18**	88.86	88.12	88.09	86.38	88.57	-10.12	88.02	87.36	-10.22
Open	**98.20**	88.66	89.25	89.23	88.26	88.48	86.55	88.76	-10.10	88.27	87.57	-10.23
Pending Close	87.84	88.40	89.04	98.91	87.97	87.88	86.11	88.41	-10.07	87.91	87.23	-10.15
Pending Customer Action	55.52	55.88	57.66	59.91	55.70	58.79	54.72	55.91	-6.64	55.68	55.38	10.20
Referred to Vendor	87.73	88.58	99.15	89.02	88.22	88.23	86.48	88.58	-10.06	88.08	87.41	-9.82
Temporary Fixed	88.07	88.70	89.43	89.07	88.16	98.69	86.72	88.98	-10.09	88.34	87.69	-10.26
end	52.11	51.63	53.09	56.02	51.58	54.02	50.52	51.90	-6.38	51.61	51.32	7.66

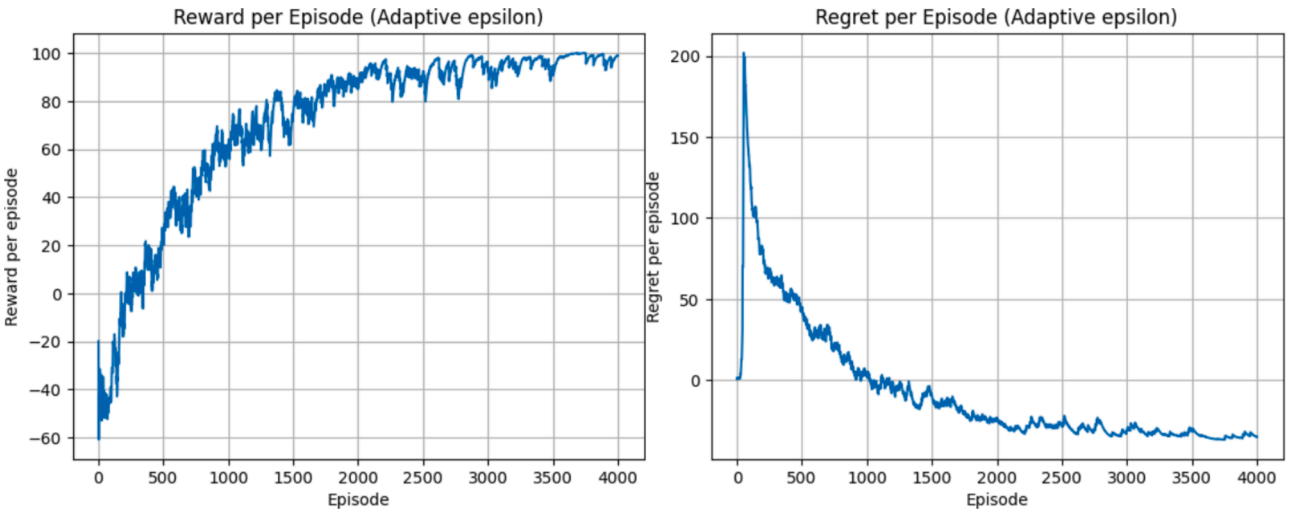

Fig. 10. (a) Reward per episode (Deep Q-network), (b) Regret per episode (Deep Q-network).

Table 13
Comparison of Q-learning and Deep Q-network Performance across key metrics.

	Average steps per episode	Average duration for episode (sec.)	Total training duration (sec.)	Average reward per episode	Optimal Path
Q-learning	2.8	0.0004	~2.04	7.7	*'Open'* – *'Assigned'* – *'Cleared'* – *'end'*
Deep Q-network	4.1	0.015	~86.44	70	*'Open'* – *'Assigned'* – *'Investigating'*- *'Cleared'* – *'end'*

6. Conclusion and future work

Our research contributes to the field of business process predictive monitoring field through adopting Process Mining Environment conforming to OpenAI gym specification aimed at enhancing the development of reinforcement learning algorithms for process optimization tasks. In the study, a proactive model was developed to determine the optimal transitions among states for an operation real-life business process. The study's findings revealed the effectiveness of the reinforcement learning approaches as for Q-learning and DQN towards learning the optimal path of the transitions in the business process with avoidance of the bottleneck state. Q-learning was able to produce the optimal sequence of transitions in the form of'**Open**' –'**Assigned**' –'**Cleared**' –'**end**' whilst DQN was able to generate the optimal sequence of transitions in the form of **'Open' → 'Assigned' → 'Investigating' → 'Cleared' → 'end'.**

Comparative performance metrics of the Q-learning and the Deep Q-network approaches were computed with respect to the total training time duration in which Q-learning converge faster for 4000 episodes within a duration of ~ 2.04 s compared to the DQN model time duration of ~ 86.44 s. Both approaches show a conformance of the optimal paths with the real business process due to the reward matrix tailored to each approach. The results of both approaches demonstrate an increase in the reward with a decrease in regret values towards the end of training.

In our future work, a plan is to be set to expand the application of the Deep Q-network methodology to a more extensive and intricate real-life business process. This larger process will feature an increased number of states and more complex transitions, which will enable us to fully harness DQN's capability to handle the greater dimensionality of states and actions effectively. The goal is to capitalize on the algorithm's proficiency in comprehending the process environment to identify the most optimal transition sequence for enhancement. Additionally, an idea of incorporating time/ throughput as a factor in formulating the reward/ penalty system that motivates the agent to pursue alternative pathways.

CRediT authorship contribution statement

Ghada Soliman: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Kareem Mostafa:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Omar Younis:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Van Der Aalst W, Weijters T, Maruster L. Workflow mining: discovering process models from event logs. *IEEE Trans Knowl Data Eng* 2004;16(9):1128–42. https://doi.org/10.1109/TKDE.2004.47.

[2] Van Der Aalst W. Process mining: overview and opportunities. *ACM Trans Manag Inf Syst* 2012;3(2):1–17. https://doi.org/10.1145/2229156.2229157.

[3] Garcia C, Meincheim A, Junior E, Dallagassa M, Sato D, Carvalho D, et al. Process mining techniques and applications—Asystematic mapping study. *Expert Syst Appl* 2019;133:260–95. https://doi.org/10.1016/j.eswa.2019.05.003.

[4] Francescomarino C, Ghidini C, Maggi F, Milani F. Predictive process monitoring methods: Which one suits me best? In: Weske M, editor. *Proceedings of 16th International Conference on Business Process Management*. Cham: Springer; 2018. p. 462–79.

[5] Imran M, Ismail M, Hamid S, Nasir M. "Complex process modeling in process mining: a systematic review". *IEEE Access* 2022;10:101515–36. https://doi.org/10.1109/ACCESS.2022.3208231.

[6] Huang Z, van der Aalst W, Lu X, Duan H. Reinforcement learning based resource allocation in business process management. *Data Knowl Eng* 2011;70:127–45. https://doi.org/10.1016/j.datak.2010.09.002.

[7] Tax N, Verenich I, La Rosa M, Dumas M. "Predictive business process monitoring with LSTM neural networks". Cham: Springer; 2017. p. 483–98. 10.1007/ 978-3-319-59536-8 30.

[8] Evermann J, Rehse JR, Fettke P. "Predicting process behavior using deep learning",. *Decis Support Syst* 2017;100:129–40. https://doi.org/10.1016/j.dss.2017.04.003. Smart Business Process Management.

[9] Chiorrini A, Diamantini C, Mircoli A, Potena D. "A preliminary study on the application of reinforcement learning for predictive process monitoring,". Cham: Springer; 2020. p. 124–35. 10.1007/978-3-030-72693-51.

[10] Pasquadibisceglie V, Appice A, Castellano G, Malerba D. A Multi-View deep learning approach for predictive business process monitoring. *IEEE Trans Serv Comput* 2022;15(4):2382–95. https://doi.org/10.1109/tsc.2021.3051771.

[11] Bousdekis A, Kerasiotis A, Kotsias S, Theodoropoulou G, Miaoulis G, Ghazanfarpour D. "Modelling and predictive monitoring of business processes under uncertainty with reinforcement learning". *Sensors* 2023;23(15):6931. https://doi.org/10.3390/s23156931.

[12] Kaftantzis S, Bousdekis A, Theodoropoulou G, Miaoulis G. "Predictive business process monitoring with AutoML for next activity prediction". *Intell Decis Technol* 2024:1–16. https://doi.org/10.3233/idt-240632.

[13] Luijken M, Ketykó I, Mannhardt F. An experiment on transfer learning for suffix prediction on event logs. In: *Lecture Notes in Business Information Processing*; 2024. p. 31–43. 10.1007/978-3-031-50974-2_3.

[14] Brennig K, Benkert K, Löhr B, Müller O. "Text-Aware predictive process monitoring of knowledge-intensive processes: does control flow matter?". In: *Lecture notes in business information processing*; 2024. p. 440–52. 10.1007/978-3-031-50974-2_33.

[15] Neubauer TR, Da Silva VF, Fantinato M, Peres SM. Resource allocation optimization in business processes supported by reinforcement learning and process mining. In: *Lecture Notes in Computer Science*; 2022. p. 580–95. 10.1007/978-3-031-21686-2_40.

[16] Durán F, Pozas N, Rocha C. Business processes resource management using rewriting logic and deep-learning-based predictive monitoring. *Journal of Logical and Algebraic Methods in Programming* Jan. 2024;136:100928. https://doi.org/10.1016/j.jlamp.2023.100928.

[17] Kotsias S, Kerasiotis A, Bousdekis A, Theodoropoulou G, Miaoulis G. "Predictive and prescriptive business process monitoring with reinforcement learning". In: Krouska A, Troussas C, Caro J, editors. *Novel and Intelligent Digital Systems Proceedings of 2nd International Conference NiDS 2022*. Cham: Springer; 2022. p. 245–54.

[18] Sutton R, Barto A. Reinforcement learning. An Introduction. Second Edition. Cambridge, MA, USA: MIT Press; 2018.

[19] Voskoglou M. Applications of finite markov chain models to management. *American Journal of Computational and Applied Mathematics* 2016;6(1):7–13. https://doi.org/10.5923/j.ajcam.20160601.02.

[20] J. Sharma, P. Andersen, O. Granmo, M. Goodwin, "Deep Q learning with Q-Matrix transfer learning for Novel Evacuation Environment," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 12, pp. 7363–7381, doi: 10.1109/TSMC.2020.2967936.

[21] Lin L-J. Reinforcement learning for robots using neural networks. Carnegie Mellon University; 1992. PhD thesis.

[22] S. Sim, L. Liu, H. Bae, "Automatic Discovery of Multi-perspective Process Model using Reinforcement Learning", pre-print, DOI: https://doi.org/10.48550/arXiv.2211.16687.

[23] Mnih V, Kavukcuoglu K, Silver D, Rusu A, Veness J, Bellemare M, et al. Human-level control through deep reinforcement learning. Nature 2015;518(7540):529–33. https://doi.org/10.1038/nature14236.

[24] Arulkumaran K, Deisenroth M, Brundage M, Bharath A. Deep reinforcement learning: a brief survey. *IEEE Signal Process Mag* 2017;34(6):26–38. https://doi.org/10.1109/MSP.2017.2743240.

[25] Wanga Y, Liua Y, Chenb W, Mac Z, Liu T. Target transfer Q-learning and its convergence analysis. *Neurocomputing* 2020;392:11–22. https://doi.org/10.1016/j.neucom.2020.02.117.

[26] Watkins C, Dayan P. Q-learning. Boston: Kluwer Academic Publishers; 1992.