Intelligent Tutoring System: A Pedagogical Model Approach Based on Deep Reinforcement Learning

Fatema Alshaikh , Nabil Hewahi

***Abstract*—The primary goals of research in the fields of Artificial Intelligence (AI) and Machine Learning (ML) are to develop an autonomous agent that can perform tasks or make decisions without human intervention. Deep Reinforcement Learning (DRL) is a machine learning technique employed in Intelligent Tutoring Systems (ITS). Deep reinforcement learning is the integration of Reinforcement Learning (RL) and Deep learning (DL) techniques. In ITS, DRL provides students with individualized learning experiences based on their learning styles and preferences. This work uses a deep reinforcement learning architecture to build a pedagogical model for ITS. We present four different models. The first model is based on the Deep Q-Network (DQN) structure. The second and third models (Double DQN and Dueling DQN) are extensions of the DQN algorithm and have been further developed to address some limitations of the DQN approach. Double DQN solves the problem of overestimation of Q-values, while Dueling DQN improves the representation of the value function. The last proposed model is a hybrid model that combines the standard DQN with a learning classifier system based on Genetic Algorithms (GA). The findings of the study indicate that both Double DQN and Dueling DQN showed complete data efficiency with 100% accuracy, 1.0 F1 score, and 1.0 precision at the highest training level of 1000 episodes and 10 steps, whereas DQN performed fairly but was outperformed by advanced techniques at the same configuration level. The standard DQN achieves a precision of 0.975, an F1 score of 0.987, and an accuracy of 98%. On the other hand, the hybrid approach, which combines GA with DQN, consistently performed worse than the other three individual models in all training setups.**

***Index Terms*—Deep Q-Network, Deep Reinforcement Learning, Double DQN, Dueling DQN, Intelligent Tutoring System.**

I. Introduction

THE field of Artificial Intelligence (AI) and education aims to provide individualized, pedagogically sound, and accessible lifelong educational material to learners. The vision of AI in education is to create a "teacher for every student" or "community of teachers," transforming learning into a social activity, incorporating multimodal input from students, and supporting various teaching strategies like collaboration, inquiry, and discussion [1]. Machine learning (ML) provides automated techniques that can discover patterns in data and utilize them to accomplish specific tasks. Reinforcement Learning (RL) is a machine learning technique in which agents can gain knowledge through interactions with an environment to optimize a cumulative reward. RL depends on the principle of trial and error, in which the RL agent can learn by executing a series of actions in an environment. The purpose is to acquire knowledge through receiving feedback in the form of rewards or punishments.

Reinforcement learning has recently gained popularity because it can effectively handle difficult sequential decision-making challenges. Sequential decision-making is a fundamental concept in machine learning that helps determine the best course of action to follow in an uncertain situation to accomplish specific goals based on past experience. The development of reinforcement learning is moving into an advanced chapter toward integration with Deep learning (DL). Traditional reinforcement learning techniques are restricted to simple decision-making problems and have constraints on representing solutions. The development of deep learning overcomes this restriction, and when combined with reinforcement learning, it gives more strength to RL applications [2].

One application of reinforcement learning is Intelligent Tutoring Systems (ITS). ITS are computer-based educational systems that incorporate independent databases or knowledge bases containing educational content and teaching strategies. These systems can analyze the learner's comprehension and identify their strengths and weaknesses [3]. These systems aim to adapt the learning process dynamically based on this analysis. The ITS system consists of four fundamental components: a pedagogical model, a student model, a knowledge model, and a user interface [4]. The pedagogical model provides a benchmark of expert performance against which the learner's performance is assessed. ITS employs dynamic modeling techniques to simulate the learner's behavior across different situations or scenarios [5].

ITS mimics human tutors and aims to deliver immediate customized instruction or feedback to learners, typically without the need for teacher involvement [6]. It is a relevant method for individualized learning and provides a tailored educational experience for students [7]. The development of deep reinforcement learning opened new avenues for improving ITS, allowing them to learn optimal policies for guiding students through tailored learning pathways [1],[8]. Therefore, the ability of Deep Reinforcement Learning (DRL) algorithms to learn from interactions with the environment improves the capacity of the pedagogical model in ITS to model student behaviour to adapt to their individual needs and optimize their tutoring process [9].

Deep Q-Network (DQN), Double DQN, and Dueling DQN are three DRL algorithms that have gained significant attention due to their performance in several applications. The DQN algorithm is a milestone in utilizing deep neural networks to estimate the Q-function, allowing DRL agents to operate complex and high-dimensional environments efficiently [10]. Double DQN and Dueling DQN are extensions to the DQN algorithm that solve many problems and optimize the performance of the standard DQN algorithm. Double

Manuscript received September 20, 2024; revised February 22, 2025.

F. Alshaikh is a PhD candidate in the Computing and Information Science program at the University of Bahrain, Bahrain. (e-mail: faalshaikh@uob.edu.bh)

N. Hewahi is a professor of computer science working at the University of Bahrain, Bahrain. (e-mail: nhewahi@uob.edu.bh)

DQN solves the issue of overestimating the Q-values [11], while the Dueling DQN aims to improve the representation of the value function [12]. These developments have shown enhanced performance and increased data efficiency in several fields, making them suitable for investigating their potential in the field of intelligent tutoring systems [13].

While the applications of DRL algorithms have been extensively studied and applied in various domains like game playing [14], robotics [15], network intrusion detection [16] and resource management [17], their uses in intelligent tutoring systems are still mostly unexplored. Understanding the relative strengths and weaknesses of deep reinforcement algorithms, as well as their suitability for ITS, is crucial to unlocking their full potential in enhancing personalized and adaptive learning experiences [18]. This paper tries to fill the gap in research by exploring the effectiveness of DRL to enhance the pedagogical model in ITS and comparing the performance of these algorithms within an intelligent tutoring system context in detail, aiming to provide valuable insights that guide the development and deployment of DRL-powered ITS.

The primary contribution of this study is to propose four different architectures based on deep reinforcement learning algorithms to develop a pedagogical model in an intelligent tutoring system and determine the most effective model for students learning. The model improves the decision-making skills of the ITS. Based on the student's preferences, current level of knowledge, and other factors, the model would determine the most effective feedback from the teacher /ITS to the student from a list of possibilities. Furthermore, the study evaluates the effectiveness of the four proposed models, DQN, Double DQN, Dueling DQN, and Genetic Algorithm (GA) with DQN, where we explore the effect of GA on the DQN model.

The plan of the work is as follows. It begins with an introductory section that provides a framework for the topic and highlights its importance and contributions. The next section presents an overview of deep reinforcement learning algorithms, followed by the literature review section. The fourth section explains the four proposed models with the pseudocode of the algorithms. The model evaluation is shown in section five, while the experimentation and the paper's results are presented in section six. The discussion and the conclusion with future works are covered in section seven and eight respectively.

II. Background

This section provides a brief overview of deep reinforcement learning algorithms. Our main focus is on the Deep Q-Network, a recent breakthrough with extensions (Double DQN and Dueling DQN).

A. Deep Q-network (DQN)

Deep Q-network (DQN) is an extension of the Q-learning algorithm. DQN is a combination of two parts: a class of Artificial Neural Networks (ANNs) called Deep Neural Networks (DNNs) [19], which are capable of processing complicated environmental data, and a reinforcement learning part that analyzes these data and determines what should be done next. DQN was developed in 2015 by Mnih et al. [10]. The main idea of a deep Q-network is to use DNN rather than Q-tables to estimate the Q-values, which indicates the future expected reward for performing an action in a specific state. The difference between Q-learning and deep Q-network is clearly shown in Figure 1. In the Q-learning algorithm, both the current state and action are taken as input to calculate the Q-values for that state-action pair. In the DQN algorithm, the inputs are the states, and the outputs are the Q-values associated with all possible actions. The neural network in DQN learns to estimate the Q-value function directly by receiving only the current state as input. This means that DQN is more computationally efficient than Q-learning. It eliminates the need to calculate the Q-value for each expected action in each state. Furthermore, in complex problems in a real-world scenario, the number of states could be large, making the creation of a Q-table very difficult.

Figure 2 shows the difference between Q-Learning and DQN. An agent at every time step in a state s takes an action a, receives a reward r, and transitions to the next state s' from the environment. The agent attempts to learn a policy or map the states to actions in the form of a Q-table, as shown in Figure 2a, to maximize the rewards. However, in many decision-making problems with high-dimensional state space, we can use artificial neural networks instead of a Q-table to predict values for actions in a given state, as shown in Figure 2b. Once the AI agent has accumulated experience, it should be capable of optimizing goals in the form of rewards.

B. DQN main components

The following are the deep Q-network major components [20]:

• Environment: The DQN algorithm involves with an environment which can be a task or problem that the RL agent must solve. The environment provides the agent with observations and rewards for taking suitable actions.

• Replay Buffer: The DQN algorithm involves a memory called replay buffer to preserve past experiences. Every experience is a tuple consisting of four elements: state, action, reward, and next state. This tuple represents a single transition from one state to another. Experiences are stored in replay memory for random sampling to be used as input data.

• Q Neural Network: The DQN algorithm utilizes a deep neural network (Q-network) to approximate the Q-values associated with each (state, action) combination. The Q-network is the agent that learns to generate the optimal state-action value. The architecture of the neural network consists of the following layers [19]:

1. Input layer: This layer takes the state representation as input. The dimensionality of the state space determines the size of the input layer.

2. Hidden layers: These layers represent the links between the input state and the Q-values for actions. The number of layers depends on the nature and complexity of the problem. One common activation function used in the hidden layers is ReLU (Rectified Linear Unit).

3. Output layer: The output layer generates Q-values for all possible actions in the specified state. The total number of actions available in the environment sets the dimension of the output layer. The output layer commonly employs a linear activation function because it directly reflects the Q-values.

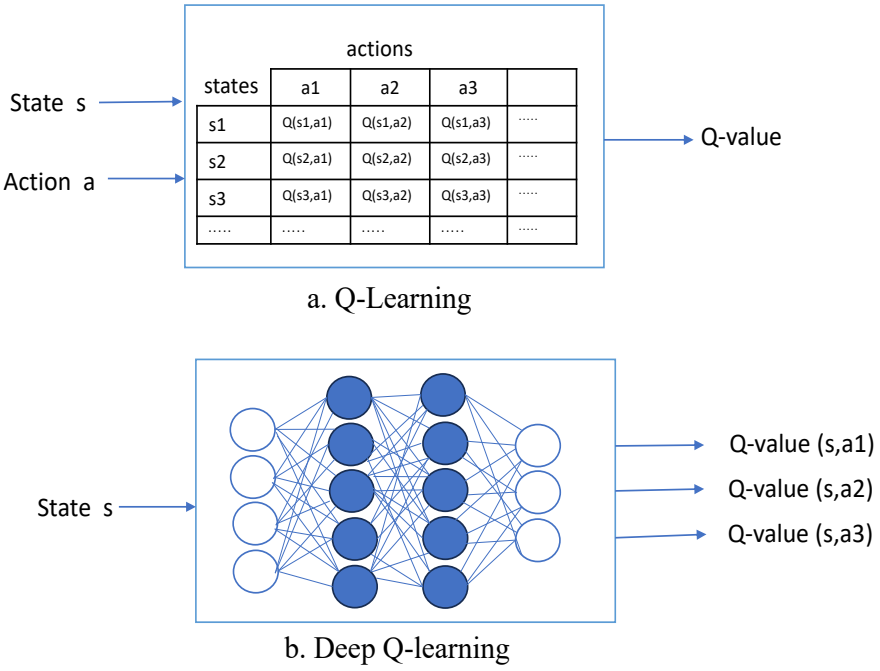


Fig. 1. Q-Learning vs Deep Q-Learning.

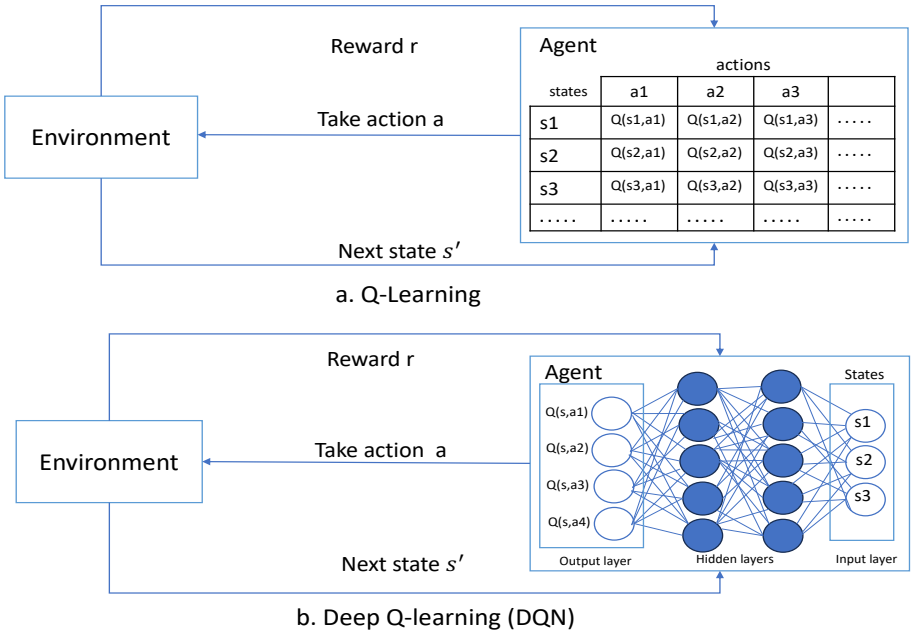


Fig. 2. Reinforcement learning vs Deep reinforcement learning.

The framework of the neural network used in DQN can vary depending on the specific problem and the complexity of the state space. For example, employing convolutional layers for image-based inputs [21] or the integration of recurrent layers for sequential data [22]. Moreover, advanced approaches such as dueling networks or prioritized experience replay [23] can enhance the effectiveness and consistency of the standard DQN in certain situations. However, the main idea of the neural network is to receive the state as input and produce the Q-value for each action as output.

• Target Neural Network: The target network (Q-target) is a copy of the primary neural network with fixed parameters. It estimates the target Q-values in DQN. The parameters of the Q-network are copied over to the target network in the training phase [23].

The DQN approach employs a deep neural network to estimate the Q-values for the current state, a replay memory buffer to store previous experiences, and a target network that is identical to the main network to estimate the Q-values for the next state. During the training phase, the agent uses an exploration strategy known as epsilon-greedy [24] to select an action. The epsilon value represents the probability of selecting a random action versus selecting the action with the highest predicted Q-value. The agent chooses an action randomly with a probability of epsilon value and selects the action with the greatest Q-value with a chance

of (1 - epsilon). Both networks use random batches from the experience replay to calculate Q-values and then do the backpropagation [25]. The Q-network trains by computing the loss function using the predicted Q-values, the target Q-values, and the observed reward from the data sample. After several rounds of iterations, the Q-network directly synchronizes its parameters with the Q-target, which is not involved in the training process. To update the network, the experience replay method [26] stores observed transitions for a certain period and then takes a uniform sample from this memory bank. The target network and the experience replay significantly enhance the algorithm's performance. Figure 3 illustrates the flow structure of the DQN algorithm.

Mathematically, a deep Q-network is represented as a neural network that, for a given state s, produces a vector of action values $Q(s, a; \theta)$, where θ represents the Q-network parameters. The Q-target network is identical to the Q-network, with parameters denoted by θ^-.

However, its parameters are updated by copying the parameters from the Q-network every t step, resulting in $\theta_t^- = \theta_t$. Equation 1 defines the target value Y that DQN uses, while Equation 2 displays the updated Q-values. The loss function calculation in equation 3 uses the squared difference between the target Q-value and the predicted Q-value.

$$Y = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{1}$$

where r represents the reward, γ indicates the discount factor, and $Q(s', a'; \theta^-)$ denotes the Q-value for the next state s' and the optimal action a' with parameter θ^-.

$$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right] \tag{2}$$

where r is the reward, γ is the discount factor, α is the learning rate, $Q(s, a)$ is the predicted Q-value for state s and action a, $Q(s', a')$ is the Q-value for the next state s' and the optimal action a' as computed by the target network.

$$\text{Loss} = \left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \tag{3}$$

where $r + \gamma \max_{a'} Q(s', a')$ is the target Q-value and $Q(s, a)$ is the predicted Q-value.

To improve the performance and stability of the deep Q-network, two extensions were proposed: the Double DQN (DDQN) [11] and the Dueling DQN [12]. The following subsections show the main concept and purpose of these algorithms.

C. Double DQN

One issue with the DQN algorithm is the overestimation of the Q-values. This problem refers to the tendency of the Q-learning algorithm to overestimate the true future rewards, particularly in the beginning stages of learning, when the agent knows nothing about the environment. Selecting the action with the highest Q-value can lead to overestimating future rewards [27]. The maximum operator with environmental noise and uncertainty might induce the algorithm to choose the most optimistic Q-value estimate, resulting in an upward bias in Q-value estimations.

Hasselt et al. [28] proposed Double DQN, a new version of the DQN algorithm, to address the problem of Q-value overestimation by splitting the max term in DQN into action selection and action evaluation. The DQN algorithm employs a single network for both action selection and evaluation, whereas the Double DQN algorithm utilizes two distinct networks: the main neural network for action selection and the target network for action evaluation. So, only the target is changes in Double DQN, as shown in equation 4.

$$Y = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-) \tag{4}$$

where r is the reward, γ represents the discount factor, $Q(s', a'; \theta^-)$ is the Q-value for the next state s' and the optimal action a' with parameter θ^-.

Initially, the primary neural network (Q-network) selects the optimal next action a' from a range of potential actions. After that, the target neural network (Q-target) evaluates this action to determine its Q-value. Therefore, the evaluation policy replaces the Q-network's weights with the Q-target's, and it regularly updates the Q-target by copying parameters from the Q-network. Using two separate networks reduces the Q-value overestimation problem and improves the final policy [29].

D. Dueling DQN

Dueling Deep Q-network architectures were proposed in 2016 by Wang et al. [12]. The main idea is to split the network into two streams: the value stream (V), which evaluates the value of a state regardless of actions taken, and the advantage stream (A), which calculates the additional value of a given action from a state compared to other actions. Then, by combining the value and the advantage streams, we can get the final Q-values. This change is beneficial, as there are situations where knowing the exact value of each action is unnecessary. In such cases, simply learning about the state-value function will be enough. Figure 4 presents the differences between the neural networks of standard DQN and dueling DQN.

How Dueling DQN integrates the value and advantage streams is the key innovation. The aggregation is performed in a manner that keeps the relative advantage of each action, while also ensuring that the Q-values are based on the actual value of being present in that state. One common way to do this is to take advantage of each action and deduct its mean or maximum from the advantage stream. In this way, the Q-value is still determined by the value function.

The Q-value estimate may be obtained by using the two aggregation techniques shown below: average advantage aggregation, as shown in equation 5, and maximum advantage aggregation, as shown in equation 6.

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a') \right) \tag{5}$$

$$Q(s, a) = V(s) + \left(A(s, a) - \max_{a'} A(s, a') \right) \tag{6}$$

where $Q(s, a)$ is the Q-value for state s and action a, $V(s)$ is the value function for state s, $A(s, a)$ is the advantage function, $\max_{a'} A(s, a')$ is the maximum advantage value for all possible actions a', $\sum_{a'} A(s, a')$ is the sum of the advantage values for all possible actions, $|A|$ is the number of possible actions.

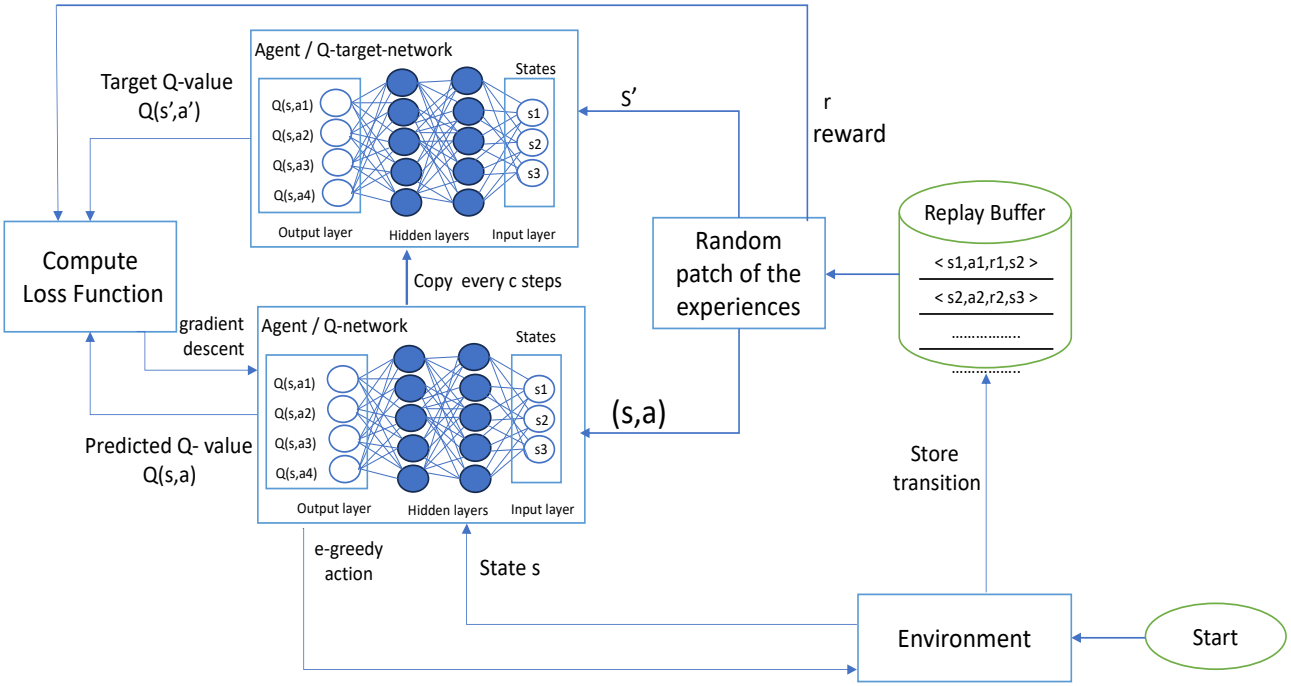
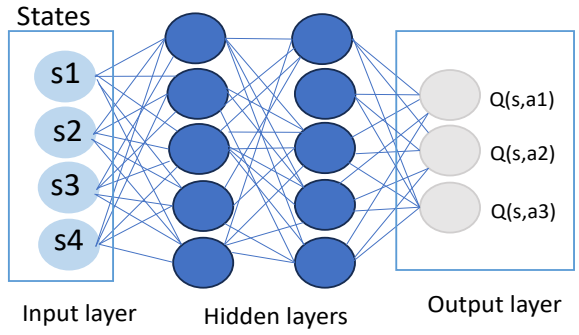
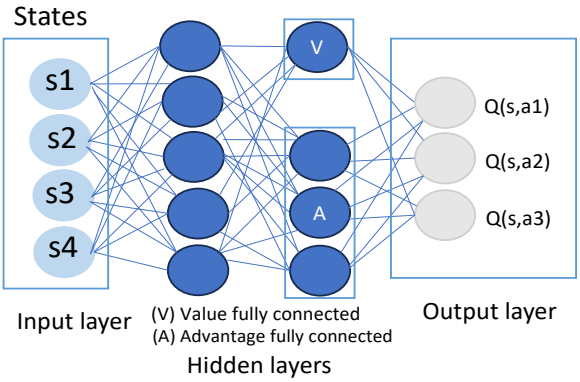


Fig. 3. DQN flow structure.



a. Standard DQN architecture



b. Dueling DQN architecture

Fig. 4. Neural network architecture vs Dueling neural network architecture.

Dueling DQN enhances learning efficiency by separating state value estimation from action advantages, especially when the action does not significantly affect the result. This distinction allows the agent to focus on gaining a deeper understanding of the significance of each state, resulting in effective learning and improved policy-making.

III. Related Work

Artificial intelligence has achieved significant advancements in reinforcement learning (RL) throughout the past century. The 1990s saw the introduction of deep learning (DL) and the subsequent breakthrough of convolutional networks in computer vision. Early in the 2000s, researchers combined deep learning neural networks with RL to create a comprehensive framework called deep reinforcement learning (DRL). This framework extended the capabilities of human-level agents and autonomous systems. A large and growing body of literature has investigated the effect of deep reinforcement learning in intelligent systems, games, robotics, and natural language processing.

Several comprehensive reviews have been published on deep reinforcement learning. For instance, Mousavi et al. [30] summarized various early RL algorithms with an overview of fundamental and contemporary issues in reinforcement learning, covering core elements, important mechanisms, and applications. A survey by Wang et al. [31] classified the existing deep RL algorithms into three categories: model-based methods, model-free methods, and advanced RL methods. They examined some recent developments, such as exploration, inverse RL, and transfer RL. In addition, research in 2018 by Fenjiro and Benbrahim [32] delved into the development timeline of reinforcement learning and deep learning technologies, emphasizing powerful advancements in these fields. Further, it discussed DRL's current challenges, real-world applications, and the hardware and frameworks used. Arulkumaran et al. [33] presented a comprehensive overview of the field, focusing on the promise of deep reinforcement learning and its application in robotics.

A considerable amount of literature has explored the potential of deep reinforcement learning in education. Reddy

et al. [34] and Li et al. [35] demonstrated the effectiveness of DRL in adaptive learning systems. It assists in creating individualized learning plans and optimizing student learning material selection. Recently, Ruan et al. [36] supported these findings, showing that deep reinforcement learning can provide adaptive pedagogical support, particularly for lower-performing students.

A range of studies have explored the use of deep reinforcement learning in intelligent tutoring systems [29], [37], [38]. Ausin et al. [37] and Abdelshiheed et al. [29] highlight the potential of DRL in inducing pedagogical policies and providing metacognitive interventions, respectively. In Ausin et al.'s work, a credit assignment problem was addressed using DRL with Gaussian processes and adaptive pedagogical tactics were induced using the DQN and Double DQN algorithms. The main findings showed that combining the DQN policy with inferred rewards outperformed the random policy, especially for educators with high pre-test scores. Abdelshiheed et al. compared the effectiveness of metacognitive interventions for student learning using a Random Forest Classifier (RFC) for static interventions and Deep Reinforcement Learning (DRL) for adaptive interventions on ITS. The result shows that, despite what the RFC said, the DRL method offered adaptive interventions by using Double DQN to avoid overestimation issues and keep the focus on metacognitive interventions. In 2021, Subramanian and Mostow [38] used DRL to train pedagogical policies in ITS, achieving higher learning gains than traditional methods. Furthermore, Paduraru et al. [39] developed the Agent to Human Recommender System (AHRec) framework to construct ITS using DRL techniques. Using texts, images, and arrows, their proposed framework can provide students with appropriate online suggestions from their teacher agents.

Other studies focus on exploration strategies and using a tutor-student network to improve system performance. Koroveshi and Ktona [24] focus on the ITS's pedagogical module by investigating different exploration strategies in RL and comparing their performance during training and testing phases. Their study [24] explored four methods: random, greedy, epsilon-greedy, and Boltzmann. According to the findings, the Boltzmann policy outperformed all other strategies during the testing phase, showing efficient learning and decision-making. One issue with using RL is that it requires many iterations and data for effective training. On the other hand, Zeng et al. [40] introduced a tutor-guided policy that consists of tutor and student modules to accelerate deep reinforcement learning. Their paper focuses on the following algorithms: deep Q-learning, deep deterministic policy gradient, asynchronous advantage actor-critic, trust region policy optimization, and proximal policy optimization. This approach [40] improves the ability to navigate by offering more information, resulting in accelerated learning and higher rewards. Another study by Markel Ausin [37] focuses on assessing the impact of offline DRL on promoting pedagogical policies in intelligent tutoring systems. Its goal is to improve DRL algorithms so they work better in educational environments. It tackled challenges such as delayed and noisy rewards, enhanced communication between tutors and students, and improved policy generalizability and transferability.

A number of authors have explored the integration of genetic algorithms into deep reinforcement learning. In 2019, research by Sehgal et al. [41] utilized a GA to optimize parameters in the deep deterministic policy gradient algorithm, leading to improved performance and faster learning in robotic manipulation tasks. In 2021, Kryvenchuk et al. [42] concentrate on selecting network hyperparameters and model design to enhance model performance and decrease development time. These works highlighted the potential of genetic algorithms in enhancing the effectiveness of deep reinforcement learning systems. In short, prior research has shown the effectiveness of deep reinforcement learning algorithms in various domains, like education. Moreover, most researchers confirmed the ability of DRL to build an intelligent tutoring system. However, DRL's major contribution to ITS has not yet been fully explored, and there has been a lack of studies on creating a pedagogical strategy that improves students' learning efficiency.

IV. Proposed Models Architectures

This section explains the main procedure for building a pedagogical model for an intelligent tutoring system. It describes the structure of four different proposed models in detail. The first model is based on a DQN structure. The second and third models are advanced versions of the standard DQN, which solve some of the problems in DQN (Double DQN and Dueling DQN). Fourth, we present a hybrid approach integrating GA with DQN.

The primary goal of implementing those models is to determine the most efficient strategies for decision-making by identifying the best policies (tactics) to choose from various options. For instance, the instructor, or ITS, may employ these strategies to direct learners according to their responses to certain challenges. This technique assists teachers /ITS in providing students with optimal feedback, thus helping them fix their errors and solve problems successfully. Therefore, various tactics might have a substantial impact on students' academic advancement.

The following outline is the main procedures for building such models.

1. Determine the models dimensions

Initialize the model dimensions by determining the input and output sizes. The input size is equal to the number of states, and the output size is equal to the number of actions.

- States: We defined states as the problems that each student must solve. We assigned six problems to each student. Each answer to the problem can either be 0, 1, or 2. A value of 0 indicates no solution or a totally incorrect solution, a value of 1 indicates a partial solution, and a value of 2 indicates a complete solution.
- Actions: We introduce eight different actions called tactics (T1, T2, T3, T4, T5, T6, T7, and T8), which are the manners and behaviors of the system that should be adopted as a teaching tactic for the student based on his/her answer.

2. Define the model parameters

After determining the model's input and output sizes, we should define some crucial parameters for each algorithm's training.

- Number of episodes: is the number of iterations where each episode corresponds to one problem case.
- Number of steps: each episode has a maximum number of steps where the agent interacts with the environment during the training.
- Epsilon parameters: the variables that control the balance between exploration and exploitation (epsilon start, epsilon end, and epsilon decay)

3. Define the neural network architecture

The neural network's primary architecture is a feedforward neural network with three fully connected layers:

- The input layer is a fully connected layer that receives the state representation as input. The size of the input layer is determined by the dimensionality of the state space, which in our situation is 6.
- Hidden layers: Two fully connected layers, each including 64 neurons, are followed by a ReLU activation function.
- The output layer is a fully connected layer that produces the Q-values for each action in the state. The output layer's size is determined by the number of actions, which is 8 (tactics from T1 to T8).

4. Generate random problem cases

Generate a random problem case, which is defined as a list of six characters with values '0', '1', or '2'. The problem case presents the input (state) to the neural networks.

5. Maturity function

A mathematical equation known as the maturity function determines the appropriate tactic for the student's performance. After determining the tactic to be used, it is compared with the predicted tactic during the training phase to measure whether the model produces a correct tactic or not. The range of maturity function is between 0 and 7.8 based on a predefined weight for each type of problem given to the student. If the student solves all the problems correctly, the maturity function will be 7.8, and if the student does not solve any problem correctly, the maturity function will be 0. In case some problems are partially solved correctly, or some are solved incorrectly, and others are solved correctly, the maturity function will be some value between 0 and 7.8. The predefined weight is assumed to be assigned based on the importance of the problem and its objective.

Let *(E)* be a set of problem evaluations, where *E=[*e_1,e_2,....e_n*]*. The maturity equation can be written as follows:

$$\text{Maturity function} = \sum_{i=1}^{n} e_i \tag{7}$$

Where e_i is calculated by multiplying s_i by its corresponding weight w_i. For example, e_1 is calculated by multiplying s_1 by the corresponding weight w_1. Similarly, e_2 is calculated by multiplying s_2 with w_2 , and so on as the following:

$$\begin{aligned}\text{Maturity function} = (s_1 \cdot w_1) + (s_2 \cdot w_2) + (s_3 \cdot w_3) \\ +(s_4 \cdot w_4) + (s_5 \cdot w_5) + (s_6 \cdot w_6) + (s_n \cdot w_n)\end{aligned} \tag{8}$$

where e_i is the evaluation result of p_i, and p_i has a corresponding value s_i, which is one of three possible values, 0 (not solved the problem), 1 (partially solved the problem), and 2 (solved the problem).

The result of the maturity function can be matched against a specific tactic that could be suitable for an individual based on maturity ranges. The ranges used here are only for scientific research and do not reflect real tactics. The used maturity ranges are as follows:

- Assign tactic T1 if the maturity range is between 0 and 1.
- Assign tactic T2 if the maturity range is more than 1 and up to 2.
- Assign tactic T3 if the maturity range is more than 2 and up to 3.
- Assign tactic T4 if the maturity range is higher than 3 and up to 4.
- Assign tactic T5 if the maturity range is higher than 4 and up to 5.
- Assign tactic T6 if the maturity range is higher than 5 and up to 6.
- Assign tactic T7 if the maturity range is higher than 6 and up to 7.
- Assign tactic T8 if the maturity range is higher than 7 and up to 7.8.

6. Model 1: Deep Q-Network (DQN)

Deep Q-network architecture is based on two neural networks (Q-network and Q-target). The Q-network calculates the Q-value in state s, whereas the Q-target calculates the Q-value in the next state s'.

A large number of episodes are used to train the DQN over multiple steps. At each time step, it performs a series of operations as presented in Algorithm 4 and Figure 5. More details about the training procedures of the model in the following:

• For each episode:

o Generate a random problem case (state) and provide it to the environment.

o For each step:

- The agent chooses an action using an epsilon-greedy strategy. The epsilon value starts at 1.0 and gradually decreases to 0.01 over the training process.
- Execute the action in the environment.
- Compute the actual action based on the current state. If the actual action matches the agent-selected action (predicted action), then the agent receives a positive reward of 1. Otherwise, it receives a negative reward of -1.
- Generate a new problem case (next state)
- The agent stores the experience tuple (current state, action, reward, next state) in the replay buffer.
- Sample random batches from the experience replay memory.
- The neural network model is updated using the stored transition which helps in making the right decisions following these steps:
- Compute the target Q-values using the target network (Q-target) for the sampled next states as presented previously in equation 1.
- Compute the loss between the predicted Q-values and the target Q-values and the observed rewards as shown

in equation 3.

- The neural network weights are modified by using the method of backpropagation and stochastic gradient descent.
- The agent's target Q-network is updated every specific number of steps to match the main Q-network and stabilize the learning process.

o The total reward for each episode is recorded and the process repeats for the next episode.

o DQN tests the learn policy by making environmental decisions after training. The agent chooses the action with the highest Q-value for a specific condition.

Algorithm 1. Deep Q-Network

Input: Input size, output size, learning rate, discount factor, number of episodes, number of steps, epsilon start, epsilon end, epsilon decay.

Output: Q-values.

1. Initialize Q-network weights θ
2. Initialize Q-target network with weights $\theta^- = \theta$
3. Initialize replay memory size.
4. For each episode from 1 to number of episodes:
5. Select a random problem case (state)
6. For each step from 1 to number of steps:
7. Select action based on the state using an greedy strategy.
8. Get current state and predicted action
9. Calculate the actual action using the maturity function
10. **if** predicted action = actual action:
11. reward = 1
12. **else:**
13. reward = -1
14. Select a random problem case (get the next state)
15. Store the transition in replay memory
16. Sample a batch of transitions from replay memory
17. Calculate the target Q-value:
18. target Q-value $= r + \gamma \max_{a' \in A} Q(s', a'; \theta^-)$
19. Compute loss:
20. Loss $=$ (target Q-value $- Q(s, a))^2$
21. Copy Q-network weights to Q-target network every C steps
22. End for (steps).
23. End for (episodes).

7. Model 2: Double DQN

The second proposed model in this study is based on DQN's structure, with some modifications in calculating the target Q-values. Using the maximum Q value as the target value for training the DQN might lead to a bias towards maximizing learning, resulting in overestimation over time. The problem may be resolved using a Double Deep Q-Network (Double DQN) algorithm. We selected Hasselt's version [11], which uses two networks that share weights at regular intervals. The Q-network is used for action selection, whereas the Q-target is used for action evaluation. Algorithm 2 presents the pseudocode of the model.

8. Model 3: Dueling DQN

The third model, Dueling DQN, modifies the standard DQN to calculate the Q-value. The Q-value may be decomposed as the combination of $V(s)$, which represents the value of being in a certain state, and $A(s, a)$, which represents the advantage of executing a specific action at that state compared to all other possible actions. The system utilizes two distinct estimators for these two components, merging them via a specific aggregation layer to generate an estimate of $Q(s, a)$. Algorithm 6 shows the pseudocode of the Dueling DQN.

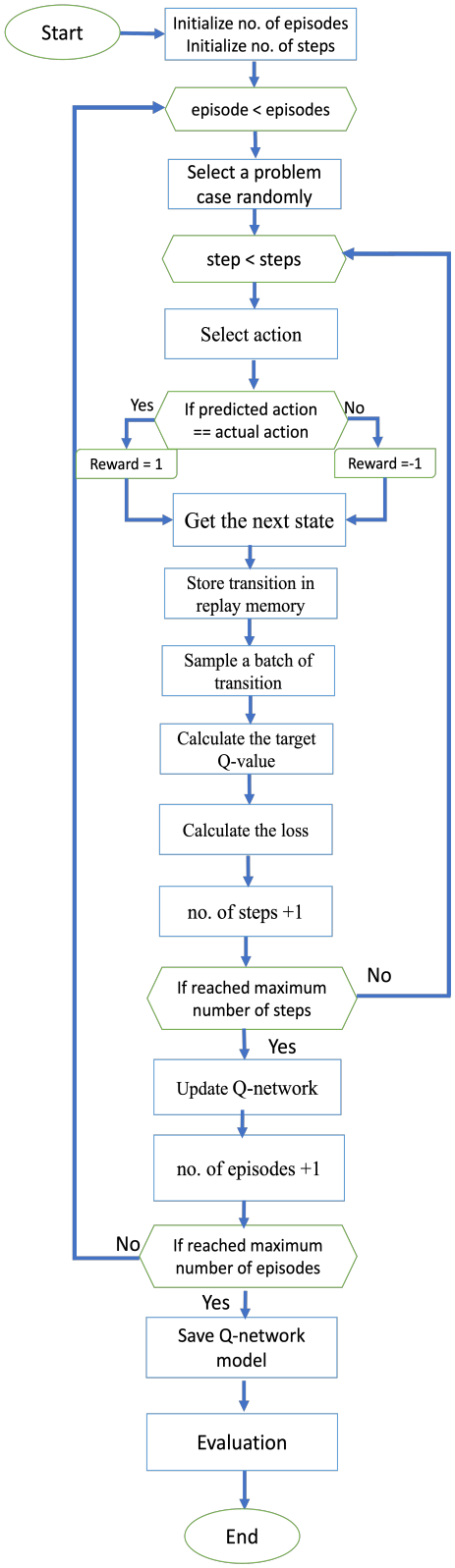


Fig. 5. Model 1: DQN

Algorithm 2. Double DQN

Input: Input size, output size, learning rate,
discount factor, number of episodes, number of steps,
epsilon start, epsilon end, epsilon decay.

Output: Q-values.

1. Initialize Q-network weights θ
2. Initialize Q-target network with weights $\theta^- = \theta$
3. Initialize replay memory size
4. For each episode from 1 to number of episodes:
5. Select a random problem case (state)
6. For each step from 1 to number of steps:
7. Select action based on the state using a greedy strategy
8. Get current state and predicted action
9. Calculate the actual action using the maturity function
10. **if** predicted action = actual action:
11. reward = 1
12. **else:**
13. reward = -1
14. Select a random problem case (get the next state)
15. Store the transition in replay memory
16. Sample a batch of transitions from replay memory
17. Calculate the target Q-values using the target Q-network
$Y = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-)$
18. Compute loss (Q-target values – Q-network values)
Loss $= (\text{target Q-value} - Q(s, a))^2$
19. Update Q-network
20. Update target Q-network
21. End for (steps)
22. End for (episodes)

Algorithm 3. Dueling DQN

Input: input size, output size, learning rate,
discount factor, number of episodes, number of steps,
epsilon start, epsilon end, epsilon decay.

Output: predicted Q-values.

1. Initialize Q-network weights θ.
2. Initialize Q-target network with weights $\theta^- = \theta$
3. Initialize replay memory size
4. For episode from 1 to episodes do:
5. Select a random problem case (state)
6. For step from 1 to steps do:
7. Select action based on the state using a greedy strategy
8. Get current state and predicted action
9. Calculate the actual action using the maturity function
10. **if** predicted action = actual action:
11. reward = 1
12. **else:**
13. reward = -1
14. Select a random problem case (get the next state)
15. Store the transition in replay memory
16. Sample a batch of transitions from replay memory
17. Calculate the predicted Q-values using the Q-network:
18. $Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a')\right)$.
19. Calculate the target Q-value:
20. target Q-value $= r + \gamma \max_{a' \in A} Q(s', a'; \theta^-)$
21. Compute loss:
22. Loss $= (\text{target Q-value} - Q(s, a; \theta))^2$
23. Copy Q-network weights to Q-target network weights every C steps
24. End for (steps)
25. End for (episodes)

9. Model 4: Combination of GA and DQN

Under this model, we try to measure the effect of the classifier system based on a genetic algorithm on a deep Q-network by first training the classifier system to generate an initial population of classifiers that represent our rules of conditions and actions, where the conditions are the student performance values for each of the six problems, and the action is the tactic that should be adopted. According to Alshaikh and Hewahi [43], a classifier system could improve very well the performance of Q-learning, and the main question here is whether DQN needs the help of a classifier system or not to get high-accuracy results. Algorithm 4 provides the pseudocode, while Figure 6 depicts the fourth proposed model procedure.

Algorithm 4. GA with DQN

Input: population size, max_num_gen, mutation_rate,
crossover_rate, max_num_iteration, max_num_problem_case,
input size,output size,learning rate,
discount factor, number of episodes, number of steps,
epsilon start, epsilon end, epsilon decay.

Output: Q-values.

1. Set GA and DQN parameters.
2. Represent the classifier.
3. Generate initial population.
4. num_problem = 0
5. While num_problem ¡ max_num_problem_case do:
6. Select a random problem case
7. Apply bucket brigade algorithm
8. num_gen = 1.
9. While num_gen ¡ max_num_gen do:
10. Select parents from population
11. Produce offspring (crossover) from chosen parents
12. Mutate the resulting offspring.
13. Add new offspring to the temp list.
14. num_gen++
15. End while
16. Update population based on temp list.
17. Make temp list empty.
18. num_problem++.
19. End while
20. Return trained classifier from GA
21. Input the trained classifier from GA to DQN
22. Initialize Q-network weights θ
23. Initialize Q-target network with weights $\theta^- = \theta$
24. Initialize replay memory size
25. For episode from 1 to episodes do:
26. Select a random problem case (state)
27. For step from 1 to steps do:
28. Select action based on the state using a greedy strategy
29. Get current state and predicted action
30. Calculate the actual action using the maturity function
31. **if** predicted action == actual action:
32. reward = 1
33. **else:**
34. reward = -1
35. Select a random problem case (get the next state)
36. Store the transition in replay memory
37. Sample a batch of transitions from replay memory
38. Calculate the target Q-value: $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$
39. Compute loss: Loss $= (\text{target Q-value} - Q(s, a; \theta))^2$
40. Copy Q-network weights to Q-target weights every C steps
41. End for
42. End for

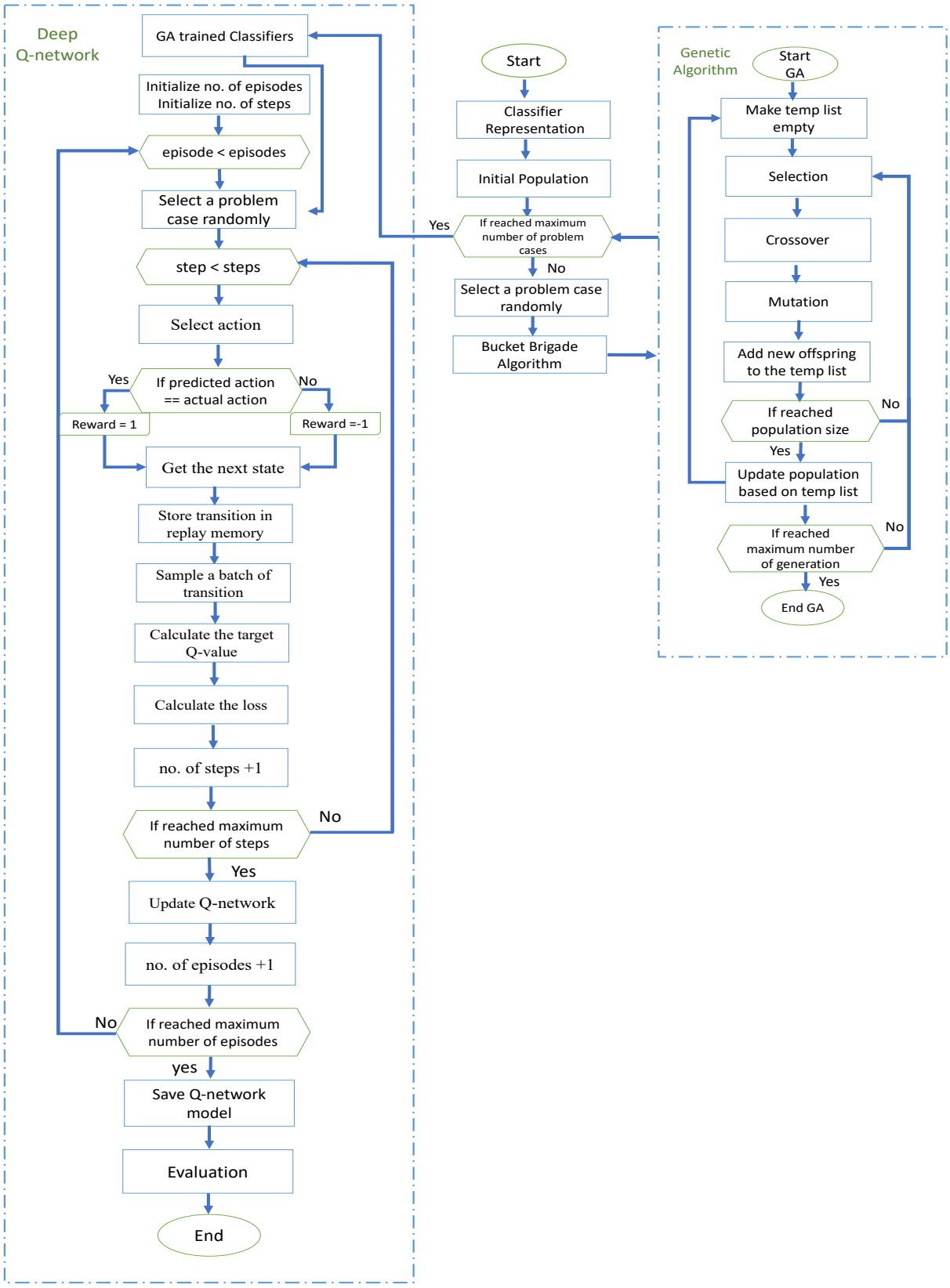


Fig. 6. Model 4: GA with DQN .

V. Models Evaluation

This paper uses the following major evaluation metrics to evaluate the performance of the deep reinforcement learning models:

- Accuracy: It estimates the proportion of all correct predictions or decisions that the acting agent makes. It is calculated as the ratio between the number of actions taken correctly and the total number of actions taken in the episodes of evaluation. This metric provides a straightforward and intuitive way to compare the overall performances of the different deep RL algorithms across the various training configurations.

• Precision: It measures the proportion of true positive predictions against all positive predictions made by the model. It gives insights into how well the model works without making false positive predictions.

• Recall: It can also be referred to as sensitivity or the true positive rate. It is the proportion of true positive predictions the model will make from all actual positive instances.

• F1-Score: It is the harmonic mean of precision and recall, making it a balanced metric to consider, all at once, a model's precision and its ability to detect all positive instances.

The values for each evaluation metric used in this study can be calculated as the following equations:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{9}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{10}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{11}$$

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{12}$$

where TP is true positive, TN is true negative, FP is false positive, and FN is false negative.

These additional performance metrics add more dimensions to the evaluation of deep reinforcement learning models and provide nuanced, informative comparisons. Precision, recall and F1-score convey the degree of accuracy of a model in identifying and classifying the correct actions relevant to real-world applications. Such metrics, put together with accuracy, will help build an idea of all models' strengths, weaknesses, and general characteristics.

VI. Experimentation and Results

This section presents the experimental approaches and outcomes of the four proposed models. In each experiment, we used a different number of episodes and steps. Overall, five main scenarios break down the steps to get the final results. Initially, we obtained the results from the standard DQN model. Subsequently, we determined the outcomes from the double DQN model. Following this, we observed the results of the dueling model. Next, we obtained the results from GA based on our previous work [43] and applied them to the DQN model. Finally, we compared the results of these approaches with our prior research [43].

For each experiment, we test the correct prediction of the problems in two different situations. The first situation uses the same collection of problem cases, while the second situation consistently uses new problem cases in each test. Furthermore, we compute the true positive, false positive, recall, precision, F1 score, and accuracy. Nevertheless, the recall outcome for all experiments is 1 due to the absence of negative data.

TABLE I
Parameters of DQN

Parameter	Value
Number of Episodes	100 to 1000
Number of Steps	1, 5, 10, 30
Epsilon Start	1.0
Epsilon End	0.01
Epsilon Decay	0.99

A. Set the hyper parameters of the models

In this experiment, we used different numbers of episodes ranging from 100 to 1000. In addition, we used different numbers of steps ranging from 1 step to 30 steps, as shown in Table I. At the beginning of the training, the initial epsilon value is typically set to a high value (1.0) so the agent can explore more by selecting a random action. Throughout training, the value of epsilon gradually decreased to a lower value (0.01) by selecting the action with the highest predicted Q-value. The decay rate (0.99) determines how quickly the epsilon value is reduced over the training process. A lower decay rate allows the agent to explore more, while a higher decay rate leads to more exploitation.

B. Results of Model 1 DQN

Table II and Table III provide the results obtained from Model 1 using the same set of problem cases and different set of problem cases, respectively. Initially, we set the parameters of the DQN model, as shown in Table I. The data in Table II shows that when there are 100 episodes and 5 steps, the number of correct predictions equals 18 out of 40, with an accuracy of 45%, using the same set of problem cases. However, this accuracy drops to only 35% when using a new set of problem cases, as shown in Table III. However, when the number of iterations exceeds 300 and the number of steps exceeds 30, the number of true positives is 32, in contrast to the number of false positives, which is only 6 out of the total 40 problem cases. The precision, F1 score, and accuracy are 0.80, 0.89, and 80%, respectively, when testing the same set of problem cases. Table II indicates a consistent growth in the performance metrics as the number of episodes and steps improves. In contrast, Table II presents some variations in the performance across different test cases. According to the results, as the number of iterations increases, the model's performance increases with higher precision, F1 score, and accuracy for both tables (Table II and Table III). In addition, the best performance is achieved at 500 episodes with 30 steps and 1000 episodes with 10 steps.

C. Results of Model 2 Double DQN

Tables IV and V present the results obtained from the double DQN model. The testing began with 100 episodes and one step, then progressed until it reached 1000 episodes and 10 steps. We used the same set of problem cases as depicted in Table IV and a different set of problem cases as presented in Table V. The total number of problem cases tested is 40. As shown in Tables IV and V, there is a clear trend toward improving the performance of the Double DQN as the number of episodes and steps increases.

TABLE II
DQN - THE RESULT OBTAINED USING THE SAME SET OF TEST CASES

No	Episodes	Steps	TP	FP	Precision	F1 Score	Accuracy
1	100	1	6	34	0.15	0.26	0.15
2	100	5	18	22	0.45	0.62	0.45
3	100	10	20	20	0.50	0.67	0.50
4	200	10	34	6	0.85	0.92	0.85
5	300	10	32	8	0.80	0.89	0.80
6	500	10	38	2	0.95	0.97	0.95
7	500	30	40	0	1.00	1.00	1.00
8	1000	10	40	0	1.00	1.00	1.00

TABLE III
DQN - THE RESULT OBTAINED USING DIFFERENT TEST CASES

No	Episodes	Steps	TP	FP	Precision	F1 Score	Accuracy
1	100	1	8	32	0.20	0.33	0.20
2	100	5	14	26	0.35	0.52	0.35
3	100	10	22	18	0.55	0.71	0.55
4	200	10	32	8	0.80	0.89	0.80
5	300	10	38	2	0.95	0.97	0.95
6	500	10	37	3	0.93	0.96	0.93
7	500	30	40	0	1.00	1.00	1.00
8	1000	10	39	1	0.98	0.99	0.98

For example, as illustrated in Table IV, with 100 episodes and 5 steps, the model achieves a precision of 0.45, an F1 score of 0.62, and a test accuracy of 45%. The number of correct predictions is 18 out of 40. Despite this, after running 200 episodes and 10 steps, the model attains a precision of 0.75, an F1 score of 0.86, and an accuracy of 75%.

TABLE IV
DOUBLE DQN - THE RESULT OBTAINED USING THE SAME SET OF TEST CASES

No	Episodes	Steps	TP	FP	Precision	F1 Score	Accuracy
1	100	1	8	32	0.20	0.33	0.20
2	100	5	18	22	0.45	0.62	0.45
3	100	10	28	12	0.70	0.82	0.70
4	200	10	30	10	0.75	0.86	0.75
5	300	10	28	12	0.70	0.82	0.70
6	500	10	38	2	0.95	0.97	0.95
7	500	30	39	1	0.98	0.99	0.98
8	1000	10	40	0	1.00	1.00	1.00

TABLE V
DOUBLE DQN - THE RESULT OBTAINED USING DIFFERENT TEST CASES

No	Episodes	Steps	TP	FP	Precision	F1 Score	Accuracy
1	100	1	7	33	0.16	0.30	0.16
2	100	5	19	21	0.48	0.64	0.48
3	100	10	21	19	0.53	0.69	0.53
4	200	10	28	12	0.70	0.82	0.70
5	300	10	31	9	0.78	0.87	0.78
6	500	10	39	1	0.98	0.99	0.98
7	500	30	39	1	0.98	0.99	0.98
8	1000	10	40	0	1.00	1.00	1.00

D. Results of Model 3 Dueling DQN

For model 3, we follow the same procedure as the previous models. Tables VI and VII present the results. For the first case with 100 episodes and 1 step, the dueling DQN model correctly identified 8 correct predictions out of a total of 40 (8 is TP, and 32 is FP) with a precision of 0.2, an F1 score of 0.33, and an accuracy of 20%, as shown in Table VI. However, in the second case, when we increase the number of steps to 5, the model's performance improves gradually, resulting in a precision of 0.4, an accuracy of 40%, and an F1 score of 0.57. Furthermore, Table VI indicates that precision, F1 score, and accuracy increase as the number of episodes increases. The model achieves an accuracy of 63%, a precision of 0.63, and an F1 score of 0.77 at 200 episodes and 10 steps. However, when the model episodes increase to 500, the model reaches a precision of 0.98, an accuracy of 98%, and an F1 score of 0.97, indicating excellent performance.

TABLE VI
DUELING DQN - THE RESULT OBTAINED USING THE SAME SET OF TEST CASES

No	Episodes	Steps	TP	FP	Precision	F1 Score	Accuracy
1	100	1	8	32	0.20	0.33	0.20
2	100	5	16	24	0.40	0.57	0.40
3	100	10	21	19	0.53	0.69	0.53
4	200	10	25	15	0.63	0.77	0.63
5	300	10	32	8	0.80	0.89	0.80
6	500	10	39	1	0.98	0.99	0.98
7	500	30	38	2	0.95	0.97	0.95
8	1000	10	40	0	1.00	1.00	1.00

TABLE VII
DUELING DQN - THE RESULT OBTAINED USING DIFFERENT TEST CASES

No	Episodes	Steps	TP	FP	Precision	F1 Score	Accuracy
1	100	1	9	31	0.23	0.37	0.23
2	100	5	16	24	0.40	0.57	0.40
3	100	10	22	18	0.55	0.71	0.55
4	200	10	24	16	0.60	0.75	0.60
5	300	10	33	7	0.83	0.90	0.83
6	500	10	37	3	0.93	0.96	0.93
7	500	30	38	2	0.95	0.97	0.95
8	1000	10	40	0	1.00	1.00	1.00

In addition to the results obtained from the Dueling DQN using the same set of problem cases, we employed different set of problem cases for each test, as presented in Table VII. Despite the minor variations in the individual metrics, the overall trend remains consistent as the number of episodes and steps increases, the dueling DQN model's performance improves across all the metrics, including precision, F1 score, and test accuracy.

E. Results of Model 4 (GA - DQN)

The results obtained from Model 4, based on the combination of GA and DQN, are shown in Tables VIII and IX, using the same set of problem cases and different sets of problem cases, respectively. The proposed model was trained

with various episodes, ranging from 100 to 1000 episodes, and steps, ranging from 1 to 30. However, the number of test problem cases with correct predictions ranges from 4 to 9 out of the total 40 test cases. This means the model cannot correctly predict the outcomes of test cases. Furthermore, the precision values range from 0.13 to 0.30, as shown in Table VIII, showing that the model has a low ratio of true positives to the total of true positives and false positives. The F1 scores vary between 0.22 and 0.46. In addition, the accuracy values range from 13% to 30%. The low precision, F1 score, and accuracy levels indicate that the GA-DQN model performs poorly and is ineffective in predicting outcomes.

TABLE VIII
DQN+GA - THE RESULT OBTAINED USING THE SAME SET OF TEST CASES

No	Episodes	Steps	TP	FP	Precision	F1 Score	Accuracy
1	100	1	9	31	0.23	0.37	0.23
2	100	5	8	32	0.20	0.33	0.20
3	100	10	9	31	0.23	0.37	0.23
4	200	10	8	32	0.20	0.33	0.20
5	300	10	4	36	0.10	0.18	0.10
6	500	10	7	33	0.18	0.30	0.18
7	500	20	6	34	0.15	0.26	0.15
8	1000	10	8	32	0.20	0.33	0.20

TABLE IX
DQN + GA - THE RESULT OBTAINED USING DIFFERENT TEST CASES

No	Episodes	Steps	TP	FP	Precision	F1 Score	Accuracy
1	100	1	12	28	0.30	0.46	0.30
2	100	5	12	28	0.30	0.46	0.30
3	100	10	12	28	0.30	0.46	0.30
4	200	10	8	32	0.20	0.33	0.20
5	300	10	7	33	0.18	0.30	0.18
6	500	10	7	33	0.18	0.30	0.18
7	500	30	5	35	0.13	0.22	0.13
8	1000	10	10	30	0.25	0.40	0.25

Table X presents the difference in the accuracy results across the four proposed models. The comparison is based on a different set of problem cases with varying numbers of training episodes and steps. The DQN, Double DQN, and Dueling DQN models generally show an improvement in accuracy values as the number of episodes and steps increases, indicating their ability to learn and enhance performance with more training. Nevertheless, the fourth model (GA with DQN) has a low level of accuracy, fluctuating between 18% and 30%, suggesting that adding genetic algorithm components does not significantly increase the performance of the DQN model to predict the correct tactics. The DQN model reaches a near-perfect performance (100%) of accuracy at 500 episodes and 30 steps. While the Dueling DQN and Double DQN models reach 100% accuracy at 1000 episodes and 10 steps, In contrast, Model 4 reaches only 30% accuracy at 100 episodes.

Furthermore, Figures 7 and 8 show the behavior of the proposed models using the same set of problem cases and different sets of problem cases, respectively. The x-axis represents the form (episodes-steps), and the y-axis represents the accuracy. We can observe from Figure 7 that the accuracy of DQN and its extensions (DDQN and Dueling DQN) increases steadily as the number of episodes and steps increases, but at specific points, it goes up and down. For instance, in the Double DQN, when the number of episodes is 200, and the number of steps is 10, it reaches an accuracy of 0.85 but drops to 0.7 when the number of episodes increases to 300. On the other hand, as illustrated in Figure 8, when we tested the models using different sets of problem cases, the accuracy of DQN and its extensions (DDQN and Dueling DQN) increased steadily as the number of episodes and steps increases. This indicates a consistent growth in the performance of the models. In model 4, the line decreases, indicating a decline in accuracy as we change the number of episodes and steps.

In terms of precision and F1 score, Table XI compares the results between all models using different sets of problem cases. At 100 episodes and 1 step, the best model is Model 4, with 0.3 precision and 0.46 F1 score. However, when we increase the number of steps to 5, the Double DQN shows the highest precision and F1 scores, which are 0.48 and 0.64, respectively. As we increase the number of iterations and steps, the DQN, Dueling DQN, and Double DQN models show a higher performance level than the fourth model. In addition, the Double DQN generally outperforms the other models, especially in higher training complexity settings with 500 episodes and above. At 500 episodes and 10 steps, Double DQN has a 5% higher precision and a 3% higher F1-score compared to the standard DQN Model 1. However, when we increased the steps to 30, the DQN model achieved the highest precision with an F1 score of 1.0, outperforming the Dueling DQN and Double DQN. At the highest training level of 1000 episodes and 10 steps, the Dueling DQN and Double DQN models achieved perfect precision with an F1-score of 1.0, outperforming the standard DQN. This demonstrates the ability of the more advanced architectures to fully leverage the additional training data and steps to achieve optimal performance.

Table XII presents the evaluation metrics for several proposed machine learning models using different sets of problem cases. Learning Classifier System (LCS), Reinforcement Learning (RL), and PMCR (LCS with RL) models were proposed previously by Alshaikh and Hewahi [43]. It is apparent from Table XII that the deep reinforcement learning algorithms, including DQN, Dueling DQN, and Double DQN models, reach the highest level of performance with perfect accuracy, precision, and an F1 score of 1.0 compared to other models. The PMCR model [43], which is based on the combination of the learning classifier system and reinforcement learning using the Q-learning algorithm, performs well, with an accuracy of 82%, a precision of 0.82, and an F1 score of 0.90. The learning classifier system [43] has a moderate level of performance, with an accuracy of 68%, a precision of 0.68, and an F1 score of 0.81. On the contrary, reinforcement learning has the lowest performance among the proposed models, with an accuracy of 33%, a precision of 0.33, and an F1 score of 0.49. On the other hand, the integration of genetic algorithms with DQN performs worst, with 30% accuracy, 0.30 precision, and a 0.46 F1 score. Figure 9 shows clearly the different evolution metrics used between various proposed machine learning models.

TABLE X
COMPARISON OF THE ACCURACY OF THE FOUR PROPOSED MODELS USING DIFFERENT SETS

No	Episodes	Steps	Model 1 DQN	Model 2 Double DQN	Model 3 Dueling DQN	Model 4 GA + DQN
1	100	1	0.20	0.16	0.23	0.30
2	100	5	0.35	0.48	0.40	0.30
3	100	10	0.55	0.53	0.55	0.30
4	200	10	0.80	0.70	0.60	0.20
5	300	10	0.95	0.78	0.83	0.18
6	500	10	0.925	0.98	0.93	0.18
7	500	30	1.00	0.98	0.95	0.13
8	1000	10	0.975	1.00	1.00	0.25

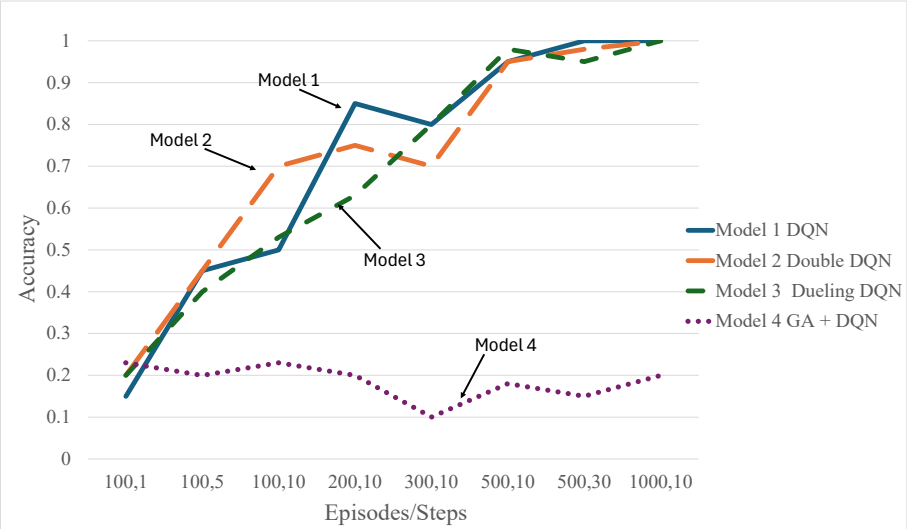


Fig. 7. Comparison of the accuracy of the proposed models using the same test set

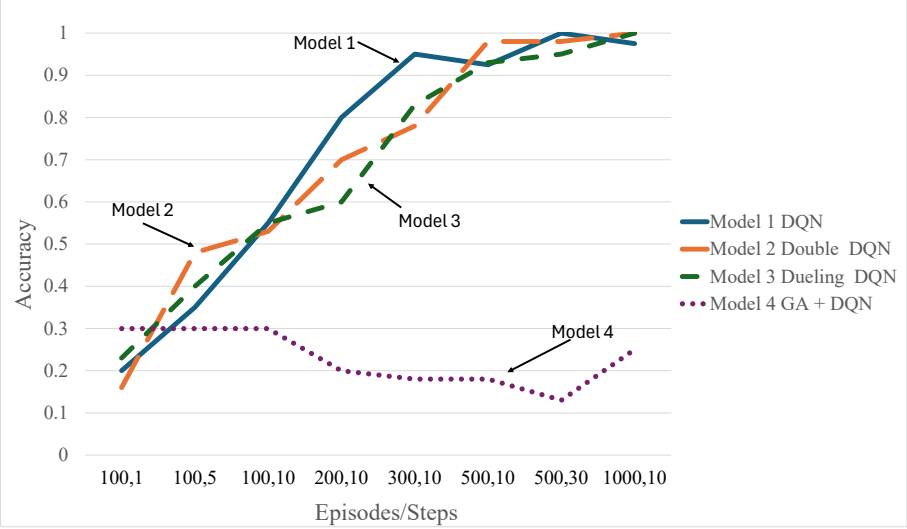


Fig. 8. Comparison of the accuracy of the proposed models using different test set

VII. DISCUSSION

The core significance of this research is to examine the effectiveness of deep reinforcement learning techniques in predicting the appropriate feedback from teachers to students in intelligent tutoring systems. The results of the study demonstrate four different proposed models. Each of them had their own strengths and trade-offs, depending on the specific training configuration. In general, the DQN, Double DQN, and Dueling DQN, as the number of episodes and steps goes up, they show a trend toward improvement in accuracy, precision, and F1 score. The optimal configuration for DQN is 500 episodes with 30 steps, achieving 100% accuracy, while the Double and Dueling DQN reach the highest performance at 1000 episodes with 10 steps. Although the Double DQN model consistently outperforms the DQN and Dueling DQN models at higher training complexities.

TABLE XI
COMPARISON OF THE PRECISION AND F1 SCORE FOR THE FOUR PROPOSED MODELS USING DIFFERENT TEST SET

No	Episodes	Steps	DQN		Double DQN		Dueling DQN		GA + DQN	
			Precision	F1 Score	Precision	F1 Score	Precision	F1 Score	Precision	F1 Score
1	100	1	0.20	0.33	0.16	0.30	0.23	0.37	0.30	0.46
2	100	5	0.35	0.52	0.48	0.64	0.40	0.57	0.30	0.46
3	100	10	0.55	0.71	0.53	0.69	0.55	0.71	0.30	0.46
4	200	10	0.80	0.89	0.70	0.82	0.60	0.75	0.20	0.33
5	300	10	0.95	0.97	0.78	0.87	0.83	0.90	0.18	0.30
6	500	10	0.93	0.96	0.98	0.99	0.93	0.96	0.18	0.30
7	500	30	1.00	1.00	0.98	0.99	0.95	0.97	0.13	0.22
8	1000	10	0.975	0.987	1.00	1.00	1.00	1.00	0.25	0.40

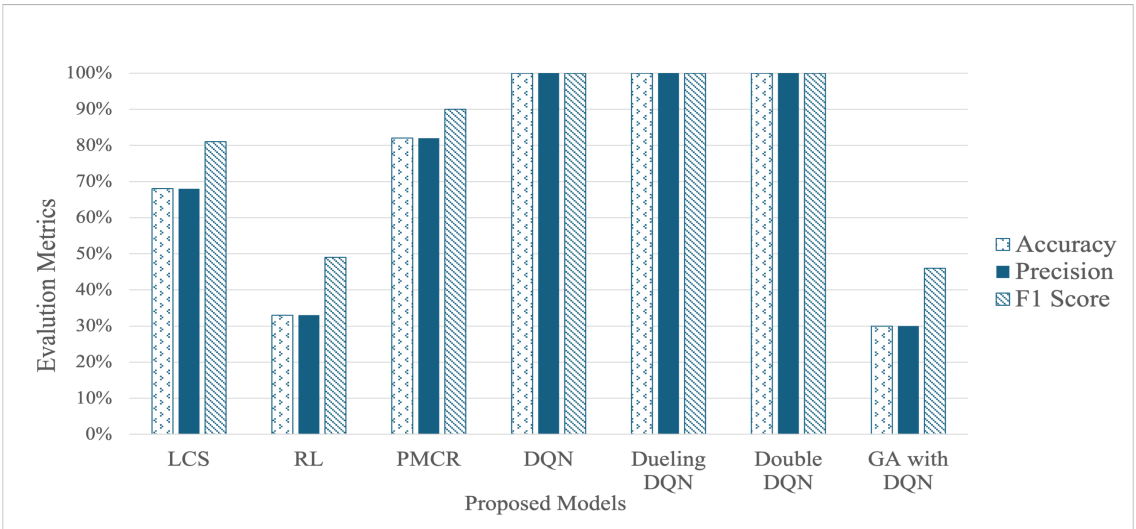


Fig. 9. Comparing the evaluation metrics for several machine learning models using different test set

TABLE XII
COMPARISON OF THE EVALUATION METRICS FOR SEVERAL PROPOSED MACHINE LEARNING MODELS USING DIFFERENT TEST SET

Models	Accuracy %	Precision	F1 Score
Learning classifier system (LCS)	68%	0.68	0.81
Reinforcement learning (RL)	33%	0.33	0.49
PMCR (LCS with RL)	82%	0.82	0.90
Deep Q-Networks (DQN)	100%	1.00	1.00
Dueling DQN	100%	1.00	1.00
Double DQN	100%	1.00	1.00
Genetic algorithm (GA) with DQN	30%	0.30	0.46

This suggests that the second proposed model (Double DQN) architecture is better suited to leverage the additional training data and complexity to continue improving its performance at higher training levels. The key advantage of Double DQN seems to be its ability to more effectively minimize the overestimation bias inherent in standard DQN, allowing it to maintain high accuracy even as the training becomes more extensive.

On the contrary, the fourth model, which is based on the integration of genetic algorithms and DQN, shows a different pattern of results compared to other algorithms. The results were not encouraging, the accuracy is generally low, with a maximum of 30% even with more episodes and steps. However, the GA-DQN model has lower precision and F1 score values, and they do not improve significantly with training complexity. Moreover, the proposed deep reinforcement learning models have higher performance outcomes compared with standard reinforcement learning. This suggests that utilizing deep learning through the neural networks empowers the model to optimize the most effective strategy from teachers to learners.

The findings from these experiments highlight the superior performance of DQN, Dueling DQN, and Double DQN models compared to the GA-DQN model. The poor performance and the lack of convergence of the GA-DQN model indicate that the genetic algorithm approach is not an effective modification approach for improving the DQN model's efficiency. The findings from these experiments highlight the superior performance of DQN, Dueling DQN, and Double DQN models compared to the GA-DQN model. The poor performance and the lack of convergence of the GA-DQN model indicate that the genetic algorithm approach is not an effective modification approach for improving the DQN model's efficiency. However, the integration of Q-learning with a genetic algorithm had a promising result, as proven by Alshaikh and Hewahi [43]. Due to the complexity of the neural network structure compared to the simplicity of Q-learning, the GA can easily leverage QL to explore the search space more effectively and find optimal solutions.

In contrast, the non-linear optimization landscape of DQN could cause difficulties for the GA in navigating successfully, leading to lower performance.

From all these, we can conclude that deep reinforcement learning techniques are successful in building a pedagogical model of an intelligent tutoring system and helping teachers give their students the appropriate feedback according to their answers to problems.

VIII. Conclusions and Future Work

In this work, we conducted a comparative analysis of the performance of four different deep reinforcement learning architectures, including DQN, Double DQN, Dueling DQN, and GA-DQN, with different levels of training complexity. The main goal of building these models is to help ITS selects the appropriate feedback for the students. In the early stages of training, with 100 episodes, the Dueling DQN model outperforms the other models. However, when the training progressed to a moderate level at 200 to 300 episodes, the standard DQN model learned robust representations and policies, reaching high accuracy. After 500-1000 episodes, Double DQN and Dueling DQN achieved 100% data efficiency, whereas DQN performed well but was outperformed by advanced approaches. In contrast, combining GA with DQN, the hybrid approach consistently underperformed the other three standalone models across all training configurations. These results are consistent with the literature on the comparative advantages of these deep RL algorithms. Previous studies indicated the enhanced capacities of Dueling DQN in exploration and generalization [12], and the improved efficiency and stability of Double DQN have also been documented [11]. Future research should investigate the performance of these techniques in more varied environments and task domains, despite this work providing valuable insights into the comparative advantages of various DRL architectures. Moreover, it would be very important to explore the combination of the dueling DQN and Double DQN architectures to leverage the strengths of both approaches.

References

[1] B. P. Woolf, *Building Intelligent Interactive Tutors: Student-Centered Strategies for Revolutionizing E-Learning*. Morgan Kaufmann, 2010.

[2] C. LiChun and ZhiMin, "An overview of deep reinforcement learning," in *ACM International Conference Proceeding Series*. Association for Computing Machinery, Jul. 2019.

[3] A. Alkhatlan and J. Kalita, "Intelligent tutoring systems: A comprehensive historical survey with recent developments," *arXiv preprint arXiv:1812.09628*, 2018.

[4] F. Alshaikh and N. Hewahi, "AI and Machine Learning Techniques in the Development of Intelligent Tutoring System: A Review," in *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*. IEEE, 2021, pp. 403–410.

[5] M. J. Mosa, I. Albatish, and S. S. Abu-Naser, "Asp. net-tutor: Intelligent tutoring system for leaning asp. net," *International Journal of Academic Pedagogical Research*, vol. 2, pp. 1–8, 2018. [Online]. Available: www.ijeais.org/ijapr

[6] H. A. S. Alrakhawi, H. A. Alrakhawi, N. Jamiat, and S. S. Abu-Naser, "Intelligent tutoring systems in education: A systematic review of usage, tools, effects and evaluation," *Article in Journal of Theoretical and Applied Information Technology*, vol. 28, 2023. [Online]. Available: https://www.researchgate.net/publication/369019319

[7] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," *Educational Psychologist*, vol. 46, no. 4, pp. 197–221, 2011.

[8] Y. K. C. Liao, "Effects of computer-assisted instruction on students' achievement in taiwan: A meta-analysis," *Computers and Education*, vol. 48, pp. 216–233, Feb. 2007.

[9] B. Memarian and T. Doleck, "A scoping review of reinforcement learning in education," *Computers and Education Open*, vol. 6, p. 100175, Jun. 2024.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[11] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[12] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2016, pp. 1995–2003.

[13] B. Fahad Mon, A. Wasfi, M. Hayajneh, A. Slim, and N. Abu Ali, "Reinforcement learning in education: A literature review," in *Informatics*, vol. 10, no. 3. MDPI, 2023, p. 74.

[14] V. Mnih, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[15] M. S. Ausin, *A Transfer Learning Framework for Human-Centric Deep Reinforcement Learning With Reward Engineering*. North Carolina State University, 2021.

[16] H. KAMAL IDRISSI and A. KARTIT, "Network intrusion detection using combined deep learning models: Literature survey and future research directions." *IAENG International Journal of Computer Science*, vol. 51, no. 8, pp. 998–1010, 2024.

[17] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *HotNets 2016 - Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. Association for Computing Machinery, Inc, Nov. 2016, pp. 50–56.

[18] A. Ezzaim, A. Dahbi, A. Haidine, and A. Aqqal, "Development, implementation, and evaluation of a machine learning-based multi-factor adaptive e-learning system." *IAENG International Journal of Computer Science*, vol. 51, no. 9, pp. 1250–1271, 2024.

[19] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, pp. 1–15, 2018.

[20] B. C. Phan, Y. C. Lai, and C. E. Lin, "A deep reinforcement learning-based mppt control for pv systems under partial shading condition," *Sensors (Switzerland)*, vol. 20, Jun. 2020.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[22] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 AAAI Fall Symposium Series*, 2015.

[23] S. Kumar, "Balancing a cartpole system with reinforcement learning–a tutorial," *arXiv preprint arXiv:2006.04938*, 2020.

[24] J. Koroveshi and A. Ktona, "A comparison of exploration strategies used in reinforcement learning for building an intelligent tutoring system." in *RTA-CSIT*, 2021, pp. 11–17.

[25] J. Kim, D. Kwon, S. Y. Woo, W. M. Kang, S. Lee, S. Oh, C. H. Kim, J. H. Bae, B. G. Park, and J. H. Lee, "On-chip trainable hardware-based deep q-networks approximating a backpropagation algorithm," *Neural Computing and Applications*, vol. 33, pp. 9391–9402, Aug. 2021.

[26] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992.

[27] A. Ly, R. Dazeley, P. Vamplew, F. Cruz, and S. Aryal, "Elastic step dqn: A novel multi-step algorithm to alleviate overestimation in deep q-networks," *Neurocomputing*, vol. 576, Apr. 2024.

[28] H. Hasselt, "Double q-learning," *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[29] M. Abdelshiheed, J. W. Hostetter, T. Barnes, and M. Chi, "Leveraging deep reinforcement learning for metacognitive interventions across intelligent tutoring systems," in *International Conference on Artificial Intelligence in Education*. Springer, 2023, pp. 291–303.

[30] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," in *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016: Volume 2*. Springer, 2018, pp. 426–440.

[31] X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao, "Deep reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 4, pp. 5064–5078, 2022.

[32] Y. Fenjiro and H. Benbrahim, "Deep reinforcement learning overview of the state of the art," *Journal of Automation, Mobile Robotics and Intelligent Systems*, vol. 12, pp. 20–39, 2018.
[33] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, Aug. 2017. [Online]. Available: http://arxiv.org/abs/1708.05866
[34] S. Reddy, S. Levine, and A. Dragan, "Accelerating human learning with deep reinforcement learning," in *NIPS Workshop: Teaching Machines, Robots, and Humans*, 2017.
[35] X. Li, H. Xu, J. Zhang, and H.-H. Chang, "Deep reinforcement learning for adaptive learning systems," *Journal of Educational and Behavioral Statistics*, vol. 48, no. 2, pp. 220–243, 2023.
[36] S. Ruan, A. Nie, W. Steenbergen, J. He, J. Q. Zhang, M. Guo, Y. Liu, K. D. Nguyen, C. Y. Wang, R. Ying, J. A. Landay, and E. Brunskill, "Reinforcement learning tutor better supported lower performers in a math task," *Machine Learning*, vol. 113, pp. 3023–3048, May. 2024.
[37] M. S. Ausin, "Leveraging deep reinforcement learning for pedagogical policy induction in an intelligent tutoring system," in *Proceedings of the 12th International Conference on Educational Data Mining (EDM 2019)*, 2019.
[38] J. Subramanian and J. Mostow, "Deep reinforcement learning to simulate, train, and evaluate instructional sequencing policies," in *Spotlight Presentation at Reinforcement Learning for Education Workshop at Educational Data Mining Conference*, 2021.
[39] C. Paduraru, M. Paduraru, and S. Iordache, "Using deep reinforcement learning to build intelligent tutoring systems." in *ICSOFT*, 2022, pp. 288–298.
[40] F. Zeng, C. Wang, and S. S. Ge, "Tutor-guided interior navigation with deep reinforcement learning," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 13, pp. 934–944, Dec. 2021.
[41] A. Sehgal, H. La, S. Louis, and H. Nguyen, "Deep reinforcement learning using genetic algorithm for parameter optimization," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2019, pp. 596–601.
[42] Y. Kryvenchuk, D. Petrenko, D. Cichon, Y. Malynovskyy, and T. Helzhynska, "Selection of deep reinforcement learning using a genetic algorithm." in *COLINS*, 2022, pp. 1129–1138.
[43] F. Alshaikh and N. Hewahi, "Adaptive Pedagogical Model Based on Classifier System and Reinforcement Learning," *The Computer Journal (Accepted with Revision)*, 2024.

Fatema Alshaikh is a Ph.D. candidate in the Computing and Information Science program at the University of Bahrain. In 2012, she completed her Master's degree in computer science from the Open University Malaysia, Bahrain, and in 2005, she graduated with a B.Sc. in Computer Science from the University of Bahrain. Her research interests mainly focus on artificial intelligence, machine learning, and intelligent systems.

Nabil M. Hewahi is a professor of computer science since 2006. He is currently working at the University of Bahrain. He obtained his B.Sc degree from Al-Fateh University, Libya, in 1986, M.Tech degree from the Indian Institute of Technology (IIT), Bombay, India, in 1991, and Ph.D. degree from Jawaharlal Nehru University, New Delhi, India, in 1994. All in Computer Science. The main research focus and interest is in the fields of AI and ML. Prof. Hewahi has published around 100 papers in international journals and conferences.