NeuRL: A Standalone No-Code Web-Based Agent Environment to Explore Neural Networks and Reinforcement Learning

Scott Siegel
Department of Biomedical Engineering
University of Florida
Gainesville, Florida, USA
sns08j@ufl.edu

Amanpreet Kapoor
Department of Engineering Education
University of Florida
Gainesville, Florida, USA
kapooramanpreet@ufl.edu

Parisa Rashidi
Department of Biomedical Engineering
University of Florida
Gainesville, Florida, USA
parisa.rashidi@ufl.edu

Abstract

Neural networks and reinforcement learning (RL) are fundamental to machine learning (ML) and AI. Given the widespread adoption of AI algorithms in industrial sectors, ensuring students understand these concepts will prepare them for a technology-driven job market. In this experience report, we introduce NeuRL, a free and accessible no-code web-based application that allows innovative real-time exploration of RL and neural networks. NeuRL provides interactive 3D WebGL environments, enabling students to experiment with multiple popular RL algorithms and observe the evolution of agents and neural networks as agents learn to accomplish tasks. To ensure NeuRL runs smoothly on low-performance computers, we created a custom neural network and RL library written in the OpenGL Shading Language (GLSL). To evaluate NeuRL's effectiveness, we introduced it to teach RL fundamentals to 111 students enrolled in an ML course. After the lesson, students completed a survey that assessed NeuRL's usability and learning effectiveness. Students found NeuRL easy to use and enjoyed its inclusion during the lesson. To the best of our knowledge, NeuRL is the first tool that enables students from any background to explore RL and observe both neural networks and agent behaviors in real time. NeuRL demonstrates the feasibility and value of providing accessible web-based tools that empower students to explore AI concepts in a manner that transcends conventional teaching methodologies.

CCS Concepts

•Human-centered computing →Visualization •Applied computing →Education →E-learning •Computing methodologies → Machine learning

Keywords

Artificial Intelligence, Machine Learning, Reinforcement Learning, Computing Education, Interactive Visualization, Neural Networks, Web Application, Machine Learning Education, AI Education, Gamification

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA
© 2025 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701850

ACM Reference format:
Scott Siegel, Amanpreet Kapoor and Parisa Rashidi. 2025. NeuRL: A Standalone No-Code Web-Based Agent Environment to Explore Neural Networks and Reinforcement Learning. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V.1 (SIGCSE TS 2025), February 26 – March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701850

1 Introduction

Machine Learning (ML) and Artificial Intelligence (AI) have permeated everyday life and are poised to become even more prevalent in the future. Despite ML and AI's impact on society, the majority of people have little understanding of the technology involved, leading to a fear of automation and AI that overshadows its potential benefits [22]. Equipping students in K-12 and higher education with an understanding of these tools is essential for their future application in crucial domains, such as addressing societal challenges, interpreting complex information, and identifying cultural and social biases that affect existing models [34]. One way to educate our students about AI is to develop interactive Open Educational Resources (OERs) and Massive Open Online Courses (MOOCs) [18, 31]. OERs are defined as "teaching, learning and research materials that make use of appropriate tools, such as open licensing, to permit their free reuse, continuous improvement, and repurposing by others for education purposes" by the United Educational, Scientific and Cultural Organization (UNESCO) [21]. The need for online OERs became even more evident during the COVID-19 pandemic, which prevented 1.21 billion students from attending school in person [15]. In this context, readily accessible OERs are crucial for ML and AI education.

To address the need for online ML and AI education resources, we have developed NeuRL, an open-source, web-based application designed for interactive exploration of neural networks and RL. Neural networks and RL are integral components of ML and AI and are used in popular state-of-the-art AI systems, such as ChatGPT [30, 38] and large language models (LLMs) [6, 28]. NeuRL is compatible with most computers and mobile devices and is accessible via this link.[1] It offers an engaging, hands-on platform for students to interact directly with these concepts.

To support educators in integrating NeuRL into their classrooms, we developed tutorial videos, accessible on YouTube, to guide them through NeuRL's features and potential classroom applications. Additional resources are available for instructors on GitHub.[2] To evaluate NeuRL's effectiveness, we conducted a study during a 50-minute class session with 111 undergraduate and graduate students. Students, equipped with laptops, used NeuRL to participate in an introductory lesson on RL and deep Q-learning [24]. Following the lesson, students completed a system usability scale (SUS) survey and a few additional questions to assess their experience with NeuRL.

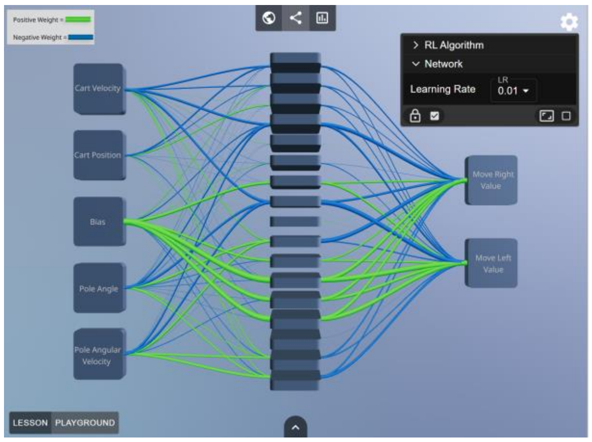
The remainder of this experience report is structured as follows: 2 presents an overview of neural networks and RL. Section 3 compares NeuRL with other similar tools for AI education. Section 4 provides the motivation for NeuRL and outlines its design and key features. Section 5 provides details about the classroom study and presents the survey results. Finally, section 6 outlines the importance of NeuRL for AI education and our future goals.

2 Neural Networks and RL

Supervised learning, unsupervised learning, and RL are the three primary subtypes of ML [10]. Models trained using supervised learning rely on labeled data, while those trained using unsupervised learning rely on unlabeled data. In RL, models act as agents within an environment [3, 20]. These agents receive information about their environment, known as their state, and take actions to achieve predefined goals. Agents receive feedback in the form of rewards, indicating whether their previous actions led to more favorable states. For example, an agent could be a self-driving car. If the agent can successfully avoid obstacles, it will receive a reward, increasing the value of the actions that helped it accomplish its goal.

Neural networks, also known as artificial neural networks (ANNs), were originally inspired by biological neurons in the brain [14]. Just as biological neurons rely on axon terminals to connect to other neurons' dendrites, neurons in ANNs are interconnected by individual weights. These artificial neurons, commonly called nodes, are organized into layers in classic ANNs. The neural network utilized in NeuRL consists of three layers, as depicted in Figure 1. The first layer is known as the input layer, where each node corresponds to a single input to the network. In RL, an example of an input could be an agent's position on a single axis. The last layer is termed the output layer, where each node corresponds to a single output produced by the network. In RL, an example of an output could be the predicted value of performing a specific action, such as moving left or right. The second layer is called the hidden layer, as it is typically not directly observed during neural network training. Hidden layers apply activation functions that transform information received from previous layers, enabling neural networks to model nonlinear relationships. Although NeuRL utilizes a single hidden layer, ANNs can contain multiple hidden layers.

[1] https://neu-rl.netlify.app



Learning Rate LR 0.01

Figure 1: Illustration of a neural network provided within NeuRL. The rectangular boxes represent individual nodes. The nodes are organized into three columns, commonly referred to as layers. The leftmost column is the input layer, the column in the middle is the hidden layer, and the column to the right is the output layer. The weights are represented by the curved cylinders that are either green or blue. Green weights correspond to positive values, while blue weights correspond to negative values. Students can control the learning rate, which determines the speed at which weights change.

3 Tools for AI Education

Many state-of-the-art (SOTA) models, such as ChatGPT, contain billions of parameters and demand substantial computing resources to operate [38]. Running large models on cloud servers removes the need for users to purchase hardware, but maintaining cloud-based servers becomes increasingly more expensive as the number of simultaneous users increases. Fortunately, the fundamental concepts that empower SOTA models can often be taught using smaller model variants, which can run on low-power machines. Providing such models for free in web browsers expands public access to ML and AI education because it removes the need to understand code or preinstall deep learning (DL) libraries on specialized hardware. Goh et al. [13] provides a detailed list of web applications for ML visualization and education.

Deep Playground [7] provides a no-code solution for users to experiment with various neural network hyperparameters, such as the learning rate, the number of nodes in the hidden layer, or the number of hidden layers. The tool includes 2D visualizations of neural network weights as well as heatmaps to display the outputs of individual neural network nodes. Deep Playground relies on a custom neural network library written in JavaScript to train neural networks. Training neural networks in JavaScript requires the machine's CPU to perform each computation. Because JavaScript is inherently a single-threaded programming language, designating CPUs to perform neural network

[2] https://github.com/snsie/neurl-resources

computations can quickly overload machines. CNN Explainer [36] and GAN Lab [17] are web-based tools that teach the fundamentals of convolutional neural networks (CNNs) and generative adversarial networks (GANs), respectively. Both tools rely on the Tensorflow.js DL library [33] which provides a CPU-bound JavaScript API to interact with smaller DL models. NeuRL originally relied on Tensorflow.js, but the library's CPU-bound API became a significant performance bottleneck when interfaced with hundreds of agents. Therefore, we created a custom library that provided a GPU-bound interface between agents and neural networks.

To the best of our knowledge, available free resources for RL education necessitate coding skills and usually depend on DL libraries. OpenAI's Gymnasium [7] library is the most popular known resource for testing RL algorithms but contains limited visualization features and does not provide a no-code user interface. ML-Agents [16] is a popular RL resource that uses the Unity game engine to visualize trained agents. Unfortunately, ML-Agents requires users to install Unity and a suite of Python-based libraries. Additionally, coding knowledge is required to modify experimental setups within ML-Agents. Table 1 compares NeuRL's key features with Gymnasium and ML-Agents.

Table 1. Comparison of Key Features

Feature	NeuRL	ML-Agents	Gymnasium
No Installation Required	✓	✗	✗
No DL Library Dependencies	✓	✗	!
Compatible with Mobile Devices	✓	✗	!
Includes Interactive 3D Visualizations	✓	✓	!
No Coding Knowledge Required	✓	!	✗

Entries marked with an exclamation mark imply limited compatibility.

4 Motivation and Design

Despite the numerous potential applications of RL in society, the field is still considered to be in its infancy [35]. RL-related research is hindered by several key challenges, such as a lack of simulation environments, limited observability of environments as agents are being trained, and a lack of learning resources [10, 32]. Another significant challenge is that the current RL resources necessitate software installation beforehand, which has proven to be a frustrating obstacle for students [19]. Offering a free and accessible no-code web-based tool for RL education that doesn't require installation or coding knowledge and enables real-time observation of agents and networks addresses these needs. NeuRL is the first web-based RL visualization tool designed for education, with the potential to stimulate RL-related research.

4.1 Software Overview

When developing NeuRL, we prioritized software that offered scalability and accessibility. To achieve both objectives, NeuRL was designed as a single-page application (SPA) that can be accessed through any modern web browser. NeuRL is hosted as static HTML files and served free of charge using Netlify's cloud platform [39]. Because services such as Cloudflare [9] provide free unlimited bandwidth for static websites, hosting NeuRL will remain free, regardless of the number of users.

NeuRL's user interface (UI) was created using Typescript and React [11] and the interactive 3D WebGL environments were created using Three.js [5]. Blender [4] was used to design complex 3D objects and generate UV texture maps [25] to combine multiple material objects into a single material object.

4.2 Neural Network and RL Library

Because JavaScript is a single-threaded language, training multiple agents in parallel using neural networks would not be feasible in real time. Therefore, a custom neural network and RL library was written using GLSL [29] and compiled into eight separate compute shaders. Parameters in compute shaders are stored as 2D matrices of pixels, with each pixel holding four separate floating-point values. GLSL scripts enable the manipulation of individual pixels and are compiled onto GPUs, enabling parallelization without requiring the preinstallation of DL libraries, a major advantage of using NeuRL. Calculating the neural network's forward and backward passes and updating agent states are broken into eight sequential steps (Figure 2).

The 3D models depicted in NeuRL are created as meshes using Three.js. Meshes contain a geometry object that determines their vertex positions and a material object that determines their appearance. Both objects rely on separate GLSL shaders to compile changes in position and appearance onto GPUs. To interface meshes with NeuRL's RL library, we modified these geometry and material shaders to track specific parameters within our compute shader RL library and visually represent that information by updating the mesh's position or color. This processing pipeline takes place entirely on GPUs and is the fundamental reason why a custom neural network and RL library was created for NeuRL.

Python-based implementations to explore RL, such as PyTorch's deep Q-learning tutorial [26], often use a single agent's replay memory to train networks. The environment is simulated at a very high frequency, which reduces the time a user has to wait but makes it infeasible to observe agents as they learn. Ensuring users can visualize every step of the simulation was an important feature we wanted to provide, but it bound the simulation frequency to the user's screen refresh rate, which is normally 30-60 FPS. To reduce the time users have to wait, our model is trained using batches of 256 independent agents. This approach reduced average training time to less than a couple of minutes while allowing users to visualize and manually control every step of the simulation.

To validate the accuracy of our compute shader pipeline, a separate testing library was created that compares the parameters stored on each compute shader with a neural network implementation written in Python.

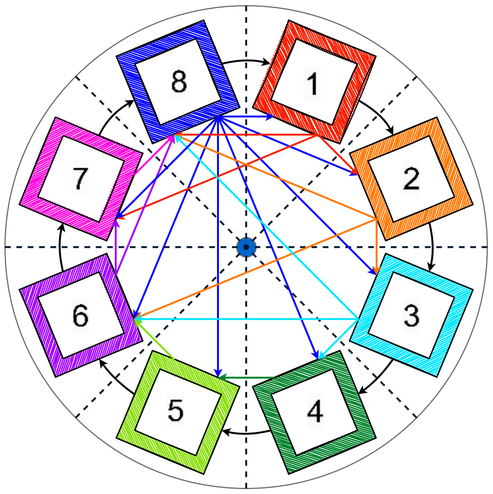


Figure 2: The neural network and RL library that is compiled onto eight separate compute shaders. The arrows in the center of the image depict dependencies between individual shaders.

4.2.1 Compute Shader 1. The first compute shader multiplies current agent states with the weight matrix that connects the input and hidden layers, as shown in Equation (1). In this equation, S_t represents the agent state matrix at time *t*, W_{hl} is the hidden layer weight matrix, and H_{net} is the matrix product.

$$H_{net} = S_t \cdot W_{hl} \tag{1}$$

4.2.2 Compute Shader 2. The second compute shader applies an activation function to each element of H_{net}, resulting in the hidden layer output matrix, H_{out}. It then multiplies H_{out} with the weight matrix connecting the hidden layer to the output layer, W_{ol}, to obtain O_{net}, the output layer sum, as shown in Equation (2). The ReLU activation function is used by default, but users can select a different activation function for experimentation.

$$O_{net} = H_{out} \cdot \mathrm{W_{ol}} = f_{hl}(H_{net}) \cdot \mathrm{W_{ol}} \tag{2}$$

4.2.3 Compute Shader 3. The third compute shader applies the output layer activation function to O_{net}, then uses the policy dictated by the selected RL algorithm to select an action. The selected action, At, is then used to derive the next potential state, as shown in Equation (3). *G* corresponds to the environmental procedure to update each agent's state, which is outlined within Gymnasium's source code [7]. An asterisk is used for this state because the agent may not assume that state if it is determined to be at a terminal step.

$$S^*_{t+1} = G(S_t, A_t) \tag{3}$$

4.2.4 Compute Shaders 4 and 5. The fourth and fifth compute shaders follow the same procedure outlined in compute shaders 1 and 2, except the potential next state, S^*_{t+1}, derived in compute shader 3, is used as the input to the neural network instead of St.

4.2.5 Compute Shader 6. The sixth compute shader uses the chain rule to derive the partial derivative of loss with respect to O_{net}, as shown in Equation (4). In this equation, $\frac{\partial E}{\partial O_{out}}$ is derived using the selected RL algorithm and the mean squared error loss function and $\frac{\partial O_{out}}{\partial O_{net}}$ is derived by calculating the derivative of the output layer activation function.

$$\frac{\partial E}{\partial O_{net}} = \frac{\partial E}{\partial O_{out}} \cdot \frac{\partial O_{out}}{\partial O_{net}} \tag{4}$$

4.2.6 Compute Shader 7. The seventh compute shader derives the partial derivative of error with respect to H_{net}. The chain rule is again used, as shown in Equation (5). In this equation, $\frac{\partial O_{net}}{\partial H_{out}}$ is equal to the output layer weight matrix. $\frac{\partial H_{out}}{\partial H_{net}}$ is equal to the derivative of the hidden layer activation function.

$$\frac{\partial E}{\partial H_{net}} = \frac{\partial E}{\partial O_{net}} \cdot \frac{\partial O_{net}}{\partial H_{out}} \cdot \frac{\partial H_{out}}{\partial H_{net}} \tag{5}$$

4.2.7 Compute Shader 8. The eighth and final compute shader updates both the neural network weights and current agent states. Agent states are updated by copying the potential next states stored in the third compute texture if the agent's episode hasn't ended. Otherwise, the agent's state is reset.

Equation (6) depicts the algorithm to update the weights connected to the output layer. In this equation, $\frac{\partial O_{net}}{\partial W_{ol}}$ is equal to H_{out}, and η is the learning rate.

$$W^+_{ol} = W_{ol} - \eta \frac{\partial E}{\partial W_{ol}} = W_{ol} - \eta \frac{\partial E}{\partial O_{net}} \cdot \frac{\partial O_{net}}{\partial W_{ol}} \tag{6}$$

Equation (7) depicts the algorithm to update the weights connected to the hidden layer. In this equation, $\frac{\partial H_{net}}{\partial W_{hl}}$ is equal to each agent's current state, S_t.

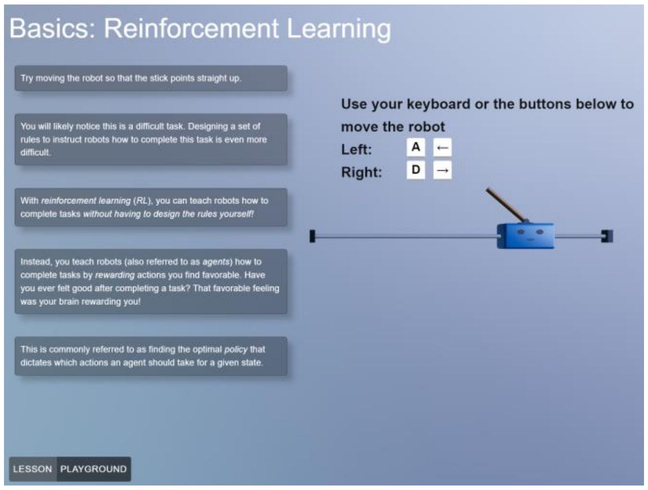
$$W^+_{hl} = W_{hl} - \eta \frac{\partial E}{\partial W_{hl}} = W_{hl} - \eta \frac{\partial E}{\partial H_{net}} \cdot \frac{\partial H_{net}}{\partial W_{hl}} \tag{7}$$

4.3 Interactive Lesson

When users navigate to NeuRL's URL, they are first provided with an interactive lesson that covers RL fundamentals and introduces Q-learning [37], the RL algorithm used in a seminal paper that demonstrated the potential of integrating RL with DL [24]. The first section, shown in Figure 3, provides a high-level overview of RL and allows students to manually control the cart agent to test their own skill at balancing a pole. Subsequent sections introduce: (1) states, actions, and rewards, (2) the Bellman equation, and (3) deep Q-learning. An interactive range slider was provided in the Bellman equation section that enabled students to tweak the gamma hyperparameter and observe how that impacts the agent's long-term predicted value. Our intent was to gamify the lesson, which has been shown to affect students' drive and learning positively [1].

4.4 Playground Environment

The playground page within NeuRL enables users to experiment with multiple RL algorithms and hyperparameters. A control panel at the top allows users to navigate between three separate scenes to visualize: (1) the agents, (2) the neural network that acts as each agent's brain, or (3) a 3D scatterplot depicting the neural network's output at varying input states.



Basics: Reinforcement Learning

Try moving the robot so that the stick points straight up.

You will likely notice this is a difficult task. Designing a set of rules to instruct robots how to complete this task is even more difficult.

With *reinforcement learning* (*RL*), you can teach robots how to complete tasks *without having to design the rules yourself!*

Instead, you teach robots (also referred to as *agents*) how to complete tasks by *rewarding* actions you find favorable. Have you ever felt good after completing a task? That favorable feeling was your brain rewarding you!

This is commonly referred to as finding the optimal *policy* that dictates which actions an agent should take for a given state.

Use your keyboard or the buttons below to move the robot

Left: A ←

Right: D →

Figure 3: First page of the lesson provided by NeuRL. Students can control the agent's movement to see if they can balance the pole manually. Students can also scroll down to see other concepts outlined in the lesson.

The icon located at the top right of the playground page opens a settings panel, displayed in Figure 4, that can be used to configure the simulation. The *Environment* section of the panel can be used to switch between the cart pole, pendulum, and mountain car classic control RL environments provided by Gymnasium [7]. The *RL Algorithm* section enables students to manipulate hyperparameters related to the RL algorithm, such as the gamma parameter, and provides a dropdown to switch between the following RL algorithms: Q-learning, SARSA, and A2C [3, 23, 24]. The *Network* section can be used to manipulate hyperparameters related to the neural network, such as the learning rate and the number of nodes in the hidden layer.

The runtime control panel displayed at the bottom of the playground page enables pausing the simulation, stepping through the simulation frame by frame, or resetting the simulation. The panel also provides a dropdown to change the FPS of the simulation and a toggleable checkbox that can freeze the neural network by preventing weights from updating.

The scene portraying the agents is displayed in Figure 4. The two plots displayed behind the agents illustrate the average value predicted by the neural network across all agents and the average episode length. The color of each agent is set based on whether the agent successfully balanced their pole for 200 frames.

The neural network scene is displayed in Figure 1 and depicts the current status of the neural network acting as each agent's brain. The weights, represented by the cylinders connected to the rectangular boxes, will update at a speed dictated by the error perceived by the agents and the selected learning rate. The 3D scatterplot scene enables sampling network outputs at different input states. The scatterplot enables observing which actions are considered to be most valuable depending on the state. Improving agent policy interpretability is one of the key challenges preventing many RL-based approaches from being applied in practice [12].

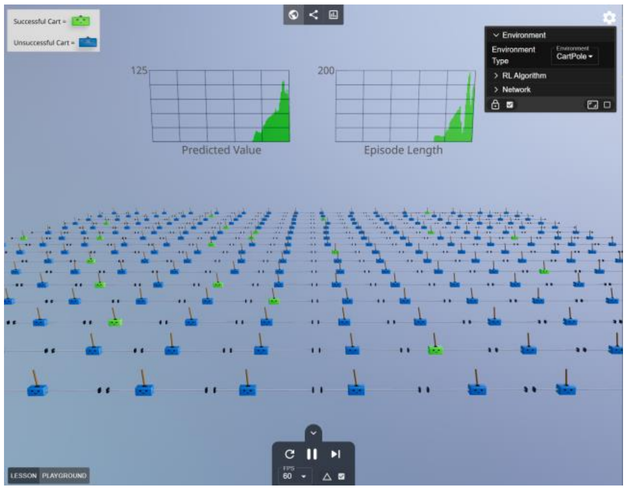


Figure 4: Agent scene included in the NeuRL's playground page. The scene portrays 256 independent agents who are learning to balance a pole.

5 Evaluation of NeuRL

5.1 Study Context

We used NeuRL to supplement a 50-minute classroom lecture covering RL fundamentals in a Machine Learning Engineering (MLE) course offered at a large public university in the United States. All students in the course were consented under IRB Protocol #ET00022492. Students were taught RL concepts using NeuRL in a guest lecture led by the first author. To gamify the lesson, students were asked to: (1) control the cart agent and see if they could balance the pole, (2) modify the gamma parameter and see if they can get the value predicted by the Bellman equation [2] to plateau at 10 instead of 100, and (3) adjust the gamma and neural network learning rate hyperparameters and compete to see who can successfully train the agents with the lowest gamma value.

After the lesson, students were provided with a link and a QR code that opened the survey in their web browser. Qualtrics [27] was used to administer the survey. The survey included questions that evaluated NeuRL's usability [2] and measured self-reported learning outcomes [8]. Those who participated and consented to research received 1% extra credit for their participation. Those who did not participate were given an alternate assignment carrying equal weight and requiring equal effort.

5.2 Participants

Out of the 111 students enrolled in the MLE course, 95 students completed our survey (Response Rate: 86%). The median age of respondents was 18-24. Demographic information for the students who completed the survey is presented in Table 2.

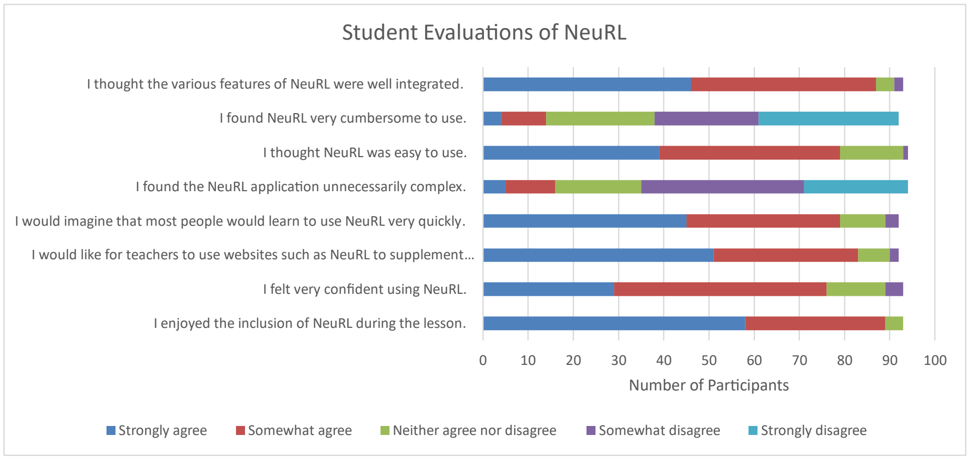


Figure 5: Evaluations of NeuRL provided by 95 undergraduate and graduate students.

Table 2. Student Demographics

	N	%
Gender		
Male	71	75
Female	23	24
No Response	1	1
Major		
Computer Science	88	93
Computer Engineering	6	6
Other	1	1
Degree Program		
Undergraduate	32	34
Graduate	63	66

5.3 Results

Student evaluations of NeuRL's usability are displayed in Figure 5. Overall, students embraced the use of NeuRL in the lesson (Mean: 4.6 / 5, STD: 0.57, using a 5-point Likert scale with Strongly agree coded as five and Strongly disagree coded as one) and found it easy to use (Mean: 4.2 / 5, STD: 0.74). Additionally, students felt that NeuRL improved their understanding of RL. For the latter, students self-reported an increase in their understanding of RL after using NeuRL ($\text{Mean}_{\text{post}}$: 3.2/5, STD_{post} 0.83) compared to their understanding before (Mean_{pre}: 2.3/5, STD_{pre} 0.93). This difference was statistically significant when conducting the Wilcoxon signed-rank test (Z: 98, N: 95, p: < 0.001).

In open-ended response questions, students reported that NeuRL strengthened their understanding of advanced ML concepts such as the Bellman Equation and Q-learning. For instance, a student reported, "*NeuRL was instrumental in deepening [their] understanding of several advanced concepts in neural networks and machine learning, particularly in areas that initially posed significant challenges.*" Another student reported that they "*learned about the Q-learning algorithm [through NeuRL], which [they] did not understand prior to interacting with NeuRL. Although the equation [was] slightly confusing, the explanations and model [gave] it more clarity*". Students also praised the system's interactivity and visualizations. For instance, students complimented the tool, stating that the "*visualizations [in NeuRL] were helpful in clarifying the math behind the concepts*" and NeuRL "*helped [them] visualize how the different layers and criteria interact with each other.*"

6 Conclusion

To the best of our knowledge, NeuRL is the first example of a web-based application that enables users to explore RL and observe both neural networks and agent behaviors in real time. The results from our study demonstrate that students overwhelmingly embraced NeuRL, indicating the need for further research to evaluate its effectiveness for AI education. We plan to conduct more empirical experiments to assess the efficacy of NeuRL compared to popular RL libraries and traditional learning methods, such as passive in-person lectures or videos. We aim to expand NeuRL into a comprehensive AI education platform and collaborate with other institutions to integrate NeuRL into their curriculum. We intend for NeuRL to showcase the feasibility and value of providing interactive web-based tools for AI education and inspire other research groups to develop similar tools that promote active learning.

Acknowledgments

We would like to thank Dr. Vincent Bindschaedler for allowing us to teach a lecture during their class, and Dr. Subhash Nerella for providing valuable feedback when designing NeuRL. We gratefully acknowledge the support of the National Science Foundation through the NSF CAREER Award 1750192, which funded this work.

References

[1] Ashraf Alam. 2022. A Digital Game based Learning Approach for Effective Curriculum Transaction for Teaching-Learning of Artificial Intelligence and Machine Learning. In *2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, April 2022. 69–74. https://doi.org/10.1109/ICSCDS53736.2022.9760932

[2] Aaron Bangor, Philip T Kortum, and James T Miller. 2008. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction* 24, 6 (2008), 574–594.

[3] Andrew G Barto. 2021. Reinforcement learning: An introduction by Richards' Sutton. *SIAM Rev* 6, 2 (2021), 423.

[4] Blender Foundation. 2024. Blender: Open Source 3D creation.

[5] Ricardo Cabello. 2022. three.js. Retrieved March 20, 2022 from https://github.com/mrdoob/three.js

[6] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems* 30, (2017).

[7] Gymnasium Contributors. 2024. Gymnasium: A toolkit for developing and comparing reinforcement learning algorithms. Retrieved from https://gymnasium.github.io/

[8] Lindsay K Crawford, Kimberly Arellano Carmona, and Rewanshi Kumar. 2024. Examining the Impact of Project-Based Learning on Students' Self-Reported and Actual Learning Outcomes. *Pedagogy in Health Promotion* (2024), 23733799241234065.

[9] Dave Dykstra, Brian Bockelman, Jakob Blomer, and Laurence Field. 2019. The Open High Throughput Computing Content Delivery Network. In *EPJ Web of Conferences*, 2019. EDP Sciences, 04023.

[10] Bisni Fahad Mon, Asma Wasfi, Mohammad Hayajneh, Ahmad Slim, and Najah Abu Ali. 2023. Reinforcement Learning in Education: A Literature Review. In *Informatics*, 2023. MDPI, 74.

[11] Artemij Fedosejev. 2015. *React.js Essentials*. Packt Publishing Ltd.

[12] Claire Glanois, Paul Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. 2024. A survey on interpretable reinforcement learning. *Machine Learning* (2024), 1–44.

[13] Hock-Ann Goh, Chin-Kuan Ho, and Fazly Salleh Abas. 2023. Front-end deep learning web apps development and deployment: a review. *Applied intelligence* 53, 12 (2023), 15923–15945.

[14] Daniel Graupe. 2013. *Principles of artificial neural networks*. World Scientific.

[15] Ronghuai Huang, D Liu, A Tlili, S Knyazeva, TW Chang, X Zhang, D Burgos, M Jemni, M Zhang, R Zhuang, and others. 2020. Guidance on open educational practices during school closures: Utilizing OER under COVID-19 pandemic in line with UNESCO OER recommendation. *Beijing: Smart Learning Institute of Beijing Normal University* (2020).

[16] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and others. 2018. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627* (2018).

[17] Minsuk Kahng, Nikhil Thorat, Duen Horng Chau, Fernanda B Viégas, and Martin Wattenberg. 2018. Gan lab: Understanding complex deep generative models using interactive visual experimentation. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 310–320.

[18] Kenneth R. Koedinger, Jihee Kim, Julianna Zhuxin Jia, Elizabeth A. McLaughlin, and Norman L. Bier. 2015. Learning is Not a Spectator Sport: Doing is Better than Watching for Learning from a MOOC. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale* (*L@S '15*), 2015. Association for Computing Machinery, New York, NY, USA, 111–120. https://doi.org/10.1145/2724660.2724681

[19] Sean Kross and Philip J Guo. 2019. Practitioners teaching data science in industry and academia: Expectations, workflows, and challenges. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019. 1–14.

[20] Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. 2022. Exploration in deep reinforcement learning: A survey. *Information Fusion* 85, (2022), 1–22.

[21] Lin Li, Robert S Keyser, and Raven Pierson. 2021. No-cost learning material: Perspectives from industrial and systems engineering students. *Journal of Higher Education Theory and Practice* 21, 10 (2021).

[22] Lívia S Marques, Christiane Gresse von Wangenheim, and Jean CR Hauck. 2020. Teaching machine learning in school: A systematic mapping of the state of the art. *Informatics in Education* 19, 2 (2020), 283–321.

[23] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 2016. PMLR, 1928–1937.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[25] Tony Mullen. 2011. *Mastering blender*. John Wiley & Sons.

[26] Adam Paszke. Google Colab. Retrieved May 12, 2024 from https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/9da0471a9eeb2351a488cd4b44fc6bbf/reinforcement_q_learning.ipynb

[27] Qualtrics LLC. 2024. Qualtrics.

[28] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems* 36, (2024).

[29] Randi J. Rost, Bill Licea-Kane, Dan Ginsburg, John Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen. 2009. *OpenGL Shading Language*. Pearson Education.

[30] Konstantinos I Roumeliotis and Nikolaos D Tselikas. 2023. Chatgpt and open-ai models: A preliminary review. *Future Internet* 15, 6 (2023), 192.

[31] Sandra Schaffert. 2010. Strategic integration of open educational resources in higher education: objectives, case studies, and the impact of Web 2.0 on universities. *Changing cultures in higher education: Moving ahead to future learning* (2010), 119–131.

[32] Adish Singla, Anna N. Rafferty, Goran Radanovic, and Neil T. Heffernan. 2021. Reinforcement Learning for Education: Opportunities and Challenges. https://doi.org/10.48550/arXiv.2107.08828

[33] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Charles Nicholson, Nick Kreeger, Ping Yu, Shanqing Cai, Eric Nielsen, David Soegel, Stan Bileschi, and others. 2019. Tensorflow.js: Machine learning for the web and beyond. *Proceedings of Machine Learning and Systems* 1, (2019), 309–321.

[34] Iro Voulgari, Marvin Zammit, Elias Stouraitis, Antonios Liapis, and Georgios Yannakakis. 2021. Learn to Machine Learn: Designing a Game Based Approach for Teaching Machine Learning to Primary and Secondary Education Students. In *Proceedings of the 20th Annual ACM Interaction Design and Children Conference* (*IDC '21*), 2021. Association for Computing Machinery, New York, NY, USA, 593–598. https://doi.org/10.1145/3459990.3465176

[35] Hao-nan Wang, Ning Liu, Yi-yun Zhang, Da-wei Feng, Feng Huang, Dong-sheng Li, and Yi-ming Zhang. 2020. Deep reinforcement learning: a survey. *Frontiers of Information Technology & Electronic Engineering* 21, 12 (2020), 1726–1744.

[36] Zijie J Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng, and Duen Horng Polo Chau. 2020. CNN explainer: learning convolutional neural networks with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 1396–1406.

[37] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, (1992), 279–292.

[38] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. 2023. A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development. *IEEE/CAA Journal of Automatica Sinica* 10, 5 (2023), 1122–1136. https://doi.org/10.1109/JAS.2023.123618

[39] Scale & Ship Faster with a Composable Web Architecture | Netlify. Retrieved May 3, 2024 from https://www.netlify.com/