UNIVERSIDAD NACIONAL DE INGENIERÍA
Facultad de Ingeniería Industrial y de Sistemas
Unidad de Posgrado
Maestría en Inteligencia Artificial

Trabajo Final

Artículo de Investigación

Aprendizaje por refuerzo profundo para CarRacing-v3 con Red Neuronal Q Profunda (DQN) y variantes

Curso: Aprendizaje por Refuerzo

Grupo 4:

Koc Góngora, Luis Enrique
Mancilla Antaya, Alex Felipe
Meléndez García, Herbert Antonio
Paitán Cano, Dennis Jack

Docente: María Tejada Begazo

14 de enero de 2026

Aprendizaje por refuerzo profundo para CarRacing-v3 con Red Neuronal Q Profunda (DQN) y variantes

Luis Koc, Alex Mancilla, Herbert Meléndez, Dennis Jack Paitán
Unidad de Posgrado, Universidad Nacional de Ingeniería (UNI)
luis.koc@gmail.com amancillaa@uni.pe
hamg.94@gmail.com dennis.paitan.c@uni.pe

***Resumen*—Se implementa y evalúa un agente de aprendizaje por refuerzo profundo para el entorno visual *CarRacing-v3* de Gymnasium, donde el agente aprende a conducir un vehículo usando únicamente observaciones rojo-verde-azul (RGB) de 96×96 píxeles del entorno de prueba. Se compara el algoritmo DQN con dos extensiones ampliamente adoptadas para estabilizar el aprendizaje y mejorar el desempeño: DQN Doble que reduce la sobreestimación) y DQN con arquitectura *dueling* para descomponer el valor del estado y la ventaja de la acción.**

Para adaptar CarRacing que requiere naturalmetne acciones continuas a DQN que solo soporta acciones discretas, se discretiza el control (timon, acelerador, freno) en 12 acciones fijas. El estado se preprocesa con conversión a escala de grises, normalización y apilamiento de 4 frames (frame stacking) para capturar dinámica temporal.

Los resultados se reportan mediante evaluación sin exploración ($\epsilon = 0$) en 5 episodios, y se presentan tablas y gráficos de entrenamiento/evaluación generados automáticamente por el proyecto. En las corridas evaluadas, Double DQN alcanzó el mayor retorno promedio a 1000 episodios, mientras que a 200 episodios los retornos fueron más variables, evidenciando sensibilidad a la etapa de entrenamiento y a la varianza del entorno.

***Palabras clave*—Aprendizaje por refuerzo, DQN, Double DQN, Dueling DQN, CarRacing-v3, Gymnasium, PyTorch.**

I. Introducción

El aprendizaje por refuerzo (RL) estudia agentes que aprenden políticas a partir de interacción con un entorno, maximizando una recompensa acumulada [1]. En problemas de control a partir de percepción visual, el uso de redes convolucionales para aproximar funciones de valor permitió escalar RL a entradas de alta dimensión [2].

En este trabajo se utiliza el entorno *CarRacing-v3* provisto por Gymnasium [3], [4]. Es un escenario de conducción con observaciones visuales y dinámica física (implementada sobre Box2D [5]) cuyo control nativo es continuo. Sobre este entorno se evalúa el impacto de mejoras conocidas sobre DQN —Double DQN [6] y Dueling DQN [7]— en comparación con el DQN base [2]. Finalmente, se prioriza la reproducibilidad mediante el uso de un repositorio público en *Github* .

Objetivos: (i) entrenar agentes DQN y variantes en CarRacing-v3; (ii) registrar métricas y gráficos de entrenamiento; (iii) evaluar los modelos sin exploración y comparar retornos con tablas/figuras reproducibles.

Este trabajo se desarrolla como trabajo final del curso de *Aprendizaje por Refuerzo*, como parte de la maestría en Inteligencia Artifical de la Universidad Nacional de Ingeniería (Perú).

II. Problema y motivación

En *CarRacing-v3*, el agente recibe únicamente observaciones visuales y debe tomar decisiones de control para maximizar el retorno acumulado mientras se desplaza por la pista [4]. La combinación de percepción de alta dimensión (imágenes), dinámica no lineal del simulador y exploración durante el aprendizaje lo convierte en un banco de prueba representativo para RL profundo [1], [2].

La motivación práctica es estudiar la estabilidad del entrenamiento con recursos limitados (ejecución en una unidad central de procesamiento (CPU) y entrenamiento sin renderizado) y comprender qué variante de DQN ofrece mejor compromiso entre complejidad y desempeño, manteniendo una implementación controlada.

III. Metodología

III-A. Entorno

Se utiliza Gymnasium [3] con el entorno *CarRacing-v3* [4]. Gymnasium es una librería/framework de código abierto en Python para aprendizaje por refuerzo que proporciona una API estándar para interactuar con entornos y para describir formalmente los espacios de observaciones y acciones. Además, incluye utilidades (*wrappers*) para transformar observaciones/recompensas y facilitar la experimentación reproducible. La interacción agente–entorno sigue la especificación oficial: la observación es una imagen RGB de 96×96 píxeles y la acción corresponde, en nuestro caso, a un control continuo de conducción: (timón, acelerador, freno) (en inglés: *steer, gas, brake*).

La dinámica de la simulación se implementa sobre Box2D [5], un motor de física 2D usado para simular colisiones y movimiento.

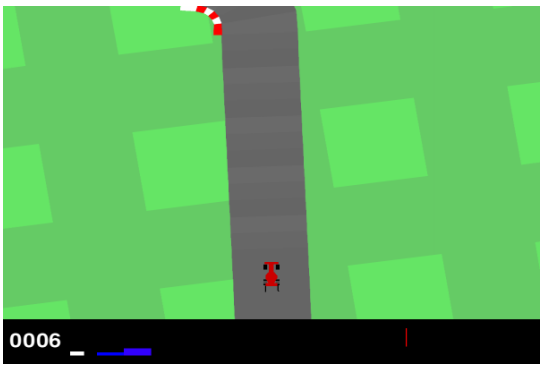

Figura 1. Vista del entorno de simulación CarRacing-v3: observación RGB de 96 × 96 píxeles desde perspectiva cenital, mostrando el vehículo (centro) navegando por la pista generada proceduralmente.

III-B. Discretización del espacio de acciones

DQN asume un espacio de acciones discreto [2]. Por ello, se define un mapeo desde el control continuo (steer, gas, brake) hacia un conjunto finito de 12 acciones. Cada acción corresponde a una combinación de giro $\in \{-1, 0, 1\}$, aceleración $\in \{0, 1\}$ y freno $\in \{0, 0{,}2\}$. Este esquema busca cubrir maniobras básicas (recto, giros, acelerar y frenar) respetando el formato de control y las convenciones descritas para el entorno [4].

III-C. Preprocesamiento y estado

Se aplica un preprocesamiento de imágenes para reducir variabilidad y costo computacional: cada frame se convierte a escala de grises y se reescala a $[0, 1]$ usando OpenCV [8]. Para incorporar información temporal sin recurrir a modelos recurrentes, se apilan los 4 frames más recientes (*frame stacking*), una estrategia habitual en DQN para entradas visuales [2]. El estado resultante tiene forma $(4, 96, 96)$.

III-D. Modelos: DQN, Double DQN y Dueling DQN

Se implementa un aproximador $Q_\theta(s, a)$ con una red neuronal convolucional (CNN, por sus siglas en inglés: *Convolutional Neural Network*) sencilla (2 capas convolucionales + un perceptrón multicapa (MLP, por sus siglas en inglés: *Multi-Layer Perceptron*)). El entrenamiento usa *experience replay* [9] y política ϵ-greedy [1].

DQN: propuesto por Mnih et al. [2], aprende $Q(s, a)$ con una red profunda y estabiliza el entrenamiento mediante *experience replay* y una red objetivo Q_{θ^-}.
Double DQN: propuesto por van Hasselt et al. [6], reduce la sobreestimación al desacoplar selección (red online) y evaluación (red objetivo) de la acción en el *target*.
Dueling DQN: propuesto por Wang et al. [7], usa una arquitectura que separa valor del estado $V(s)$ y ventaja $A(s, a)$ para estimar $Q(s, a)$ con mayor eficiencia.
Las variantes se activan con banderas de configuración (Double/Dueling) y pueden combinarse (Dueling Double DQN).

III-E. Entrenamiento

El optimizador usado es Adam, propuesto por Kingma y Ba (del acrónimo *Adaptive Moment Estimation*) [10], que adapta la tasa de aprendizaje por parámetro usando estimaciones del primer y segundo momento del gradiente; suele converger rápido y ser estable, aunque puede ser sensible a la tasa de aprendizaje y no siempre generaliza tan bien como SGD en algunos problemas [11], [12]. Se entrena por episodios, con red objetivo actualizada periódicamente (cada 5 episodios) y guardado de checkpoints. Para acelerar entrenamiento, se aplica *frame skipping* [2] con `SKIP_FRAMES=2`.

IV. Arquitectura del proyecto

La Figura 2 resume la organización del proyecto y el flujo de ejecución. El punto de entrada es `main.py`, que orquesta (i) la construcción del entorno (Gymnasium/CarRacing-v3) y su preprocesamiento; (ii) la inicialización del agente DQN y sus variantes (Double/Dueling); (iii) el ciclo de entrenamiento (recolección de experiencia, *replay buffer* y actualizaciones de la red); y (iv) la evaluación y el registro de métricas.

La lógica principal se encapsula en el paquete `src/`: `entorno.py` concentra la interacción con el entorno y el modo de observación; `preprocesamiento.py` implementa la preparación del estado (transformaciones de imagen y *frame stacking*); `agente.py` define el modelo y la política de acción; `entrenamiento.py` gestiona el bucle de entrenamiento/evaluación; y `configuracion.py` centraliza hiperparámetros y banderas. Los artefactos de salida (modelos, métricas y gráficas) se guardan en `resultados/` y se complementan con scripts en `tests/` para pruebas y evaluación.

V. Diseño experimental

V-A. Configuración

Se entrenan agentes por 1000 episodios (cuando aplica), registrando por episodio: reward acumulado, ϵ, tamaño del buffer y loss promedio. Para la evaluación, se fija $\epsilon = 0$ (sin exploración) y se ejecutan 5 episodios por checkpoint. Los retornos por episodio se exportan a un archivo de valores separados por comas (CSV, por sus siglas en inglés: *Comma-Separated Values*) y se generan gráficos automáticamente. El repositorio y el cuaderno de resultados documentan el flujo de entrenamiento y evaluación (Jupyter [13]).

V-B. Métrica

La métrica principal es el retorno total por episodio (reward acumulado), reportado como media y desviación estándar poblacional ($ddof$=0) sobre 5 episodios. Este reporte sigue el resumen implementado en el cuaderno final.

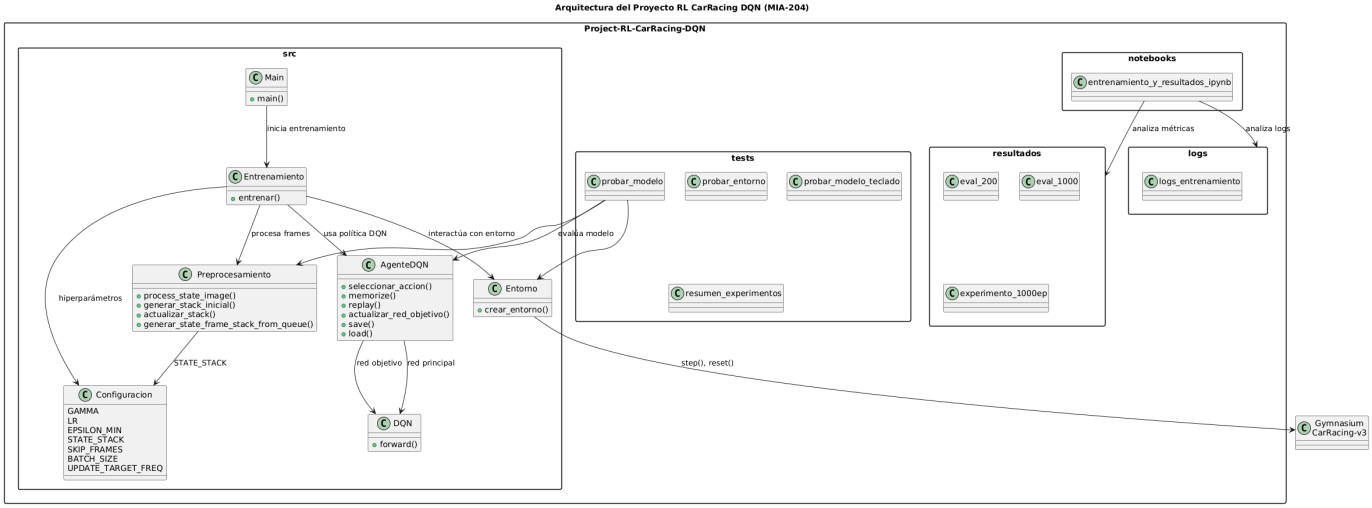

Figura 2. Arquitectura del proyecto y flujo de entrenamiento/evaluación del agente DQN en CarRacing-v3.

Tabla I
Evaluación a 200 episodios (5 episodios, $\epsilon = 0$)

Algoritmo	Mean	Std	Min	Max	Full
DQN	238.85	207.06	67.24	646.67	4
Double DQN	108.76	81.52	30.74	265.52	5
Dueling Double DQN	262.85	174.93	46.63	463.14	4

Tabla II
Evaluación a 1000 episodios (5 episodios, $\epsilon = 0$)

Algoritmo	Mean	Std	Min	Max	Full
DQN	418.58	209.95	252.73	825.09	5
Double DQN	567.72	257.19	278.29	883.33	5
Dueling Double DQN	462.46	222.57	189.76	793.47	4

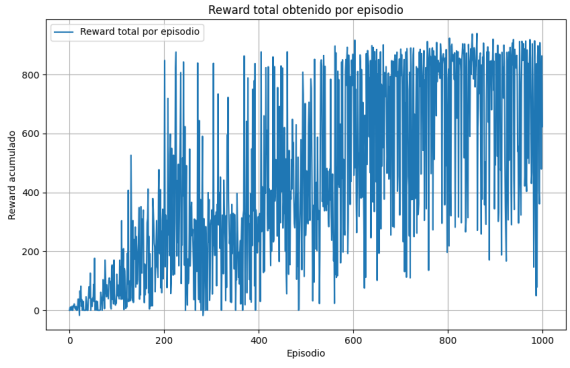

Figura 3. Entrenamiento: reward por episodio (Double DQN).

VI. Resultados

VI-A. Resumen cuantitativo

La Tabla I y la Tabla II presentan los retornos de evaluación (5 episodios, $\epsilon = 0$) para distintos algoritmos. Además de media y dispersión, se indica cuántos episodios alcanzaron 1000 frames (columna *Full*), ya que el entorno puede finalizar antes por condiciones de término o truncamiento [4].

En conjunto, los resultados sugieren que el desempeño mejora con el entrenamiento y que Double DQN tiende a ser más competitivo en este entorno cuando se entrena por más episodios: a 1000 episodios obtiene el mayor retorno promedio (567.72) y también el mayor máximo (883.33), lo que es consistente con su objetivo de reducir la sobreestimación al construir los *targets*. En contraste, a 200 episodios las medias son más inestables y la dispersión es alta (p.ej., DQN: Std=207.06), lo que indica que el ranking entre métodos aún no se consolida en etapas tempranas y que la varianza entre episodios es significativa [1]. La columna *Full* muestra que, aun sin exploración ($\epsilon = 0$), algunos episodios terminan antes de 1000 frames (truncamiento/término), por lo que el retorno observado refleja tanto la calidad de la política como la frecuencia de episodios completos.

VI-B. Gráficos de entrenamiento

En los gráficos de entrenamiento se observa la dinámica típica del aprendizaje: el retorno por episodio tiende a aumentar a medida que el agente consolida una política más efectiva, aunque con oscilaciones debidas a exploración y variabilidad del entorno. En paralelo, ϵ decae (menor exploración) y el *replay buffer* crece hasta estabilizarse, lo que incrementa la reutilización de experiencias; el *loss* suele ser ruidoso y no necesariamente decreciente de forma monótona, por lo que se interpreta como señal de estabilidad de actualización más que como métrica directa de desempeño. Las Figuras 3–5 ilustran la evolución del retorno por episodio durante el entrenamiento. Estas visualizaciones se obtienen a partir de los registros generados por el proyecto y se construyen con Matplotlib [14].

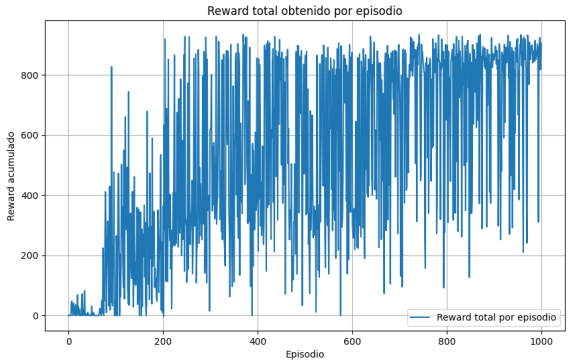

Figura 4. Entrenamiento: reward por episodio (DQN).

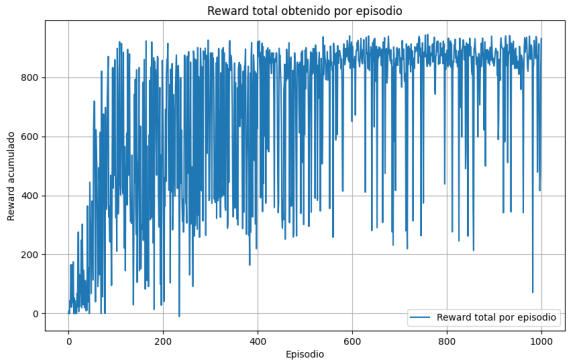

Figura 5. Entrenamiento: reward por episodio (Dueling Double DQN).

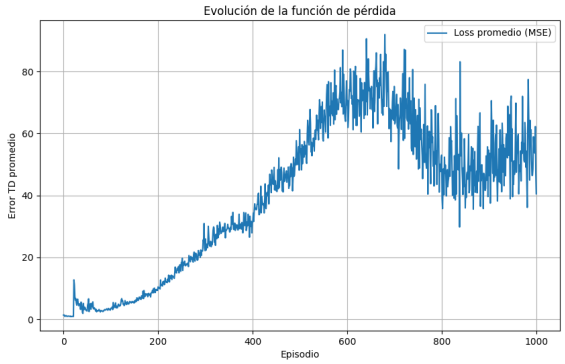

Figura 6. Entrenamiento (Double DQN): loss promedio por episodio.

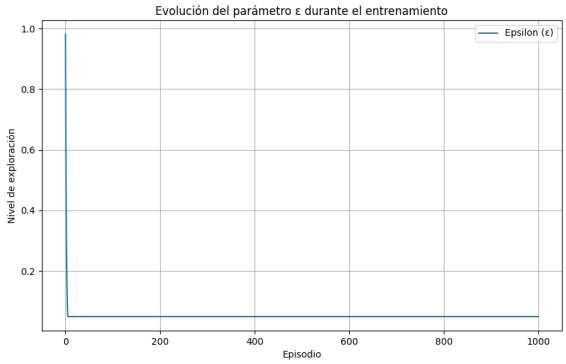

Figura 7. Entrenamiento (Double DQN): decaimiento de ϵ (exploración).

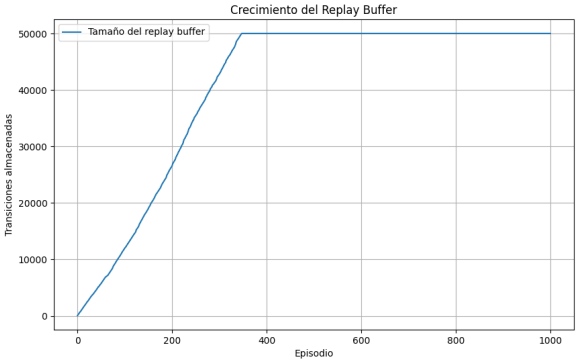

Figura 8. Entrenamiento (Double DQN): tamaño del replay buffer.

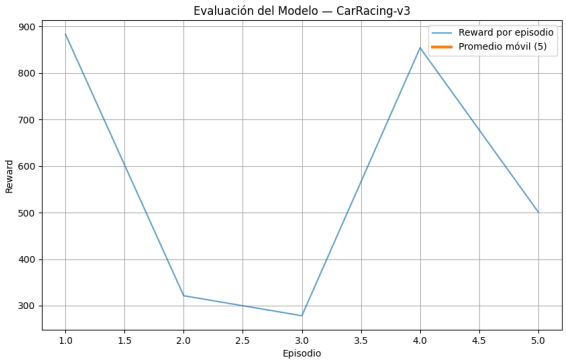

Figura 9. Evaluación (1000): Double DQN.

VI-C. Gráficos de evaluación

Los gráficos de evaluación muestran, para cada checkpoint, el retorno obtenido sin exploración ($\epsilon = 0$), por lo que reflejan directamente la calidad de la política aprendida en ese punto del entrenamiento. La dispersión entre episodios y la variación entre checkpoints evidencian (i) la varianza propia del entorno y (ii) que el desempeño no crece de forma estrictamente monótona, sino por tramos, con posibles retrocesos. Al comparar métodos, estas curvas ayudan a identificar tendencias (p.ej., retornos consistentemente más altos o más estables), pero deben leerse con cautela debido al tamaño muestral reducido. Las Figuras 9–11 muestran los retornos obtenidos en evaluación (5 episodios, $\epsilon = 0$) para distintos checkpoints. Dado el tamaño muestral reducido, se interpretan como un apoyo descriptivo y no como evidencia concluyente [1].

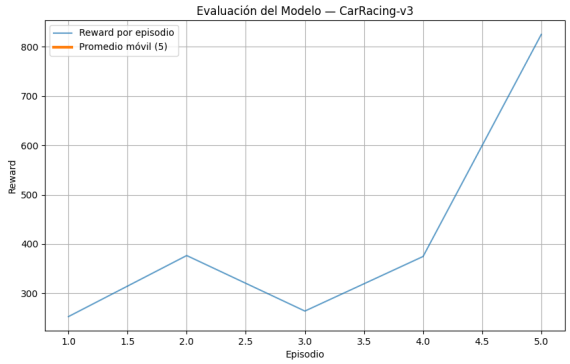

Figura 10. Evaluación (1000): DQN.

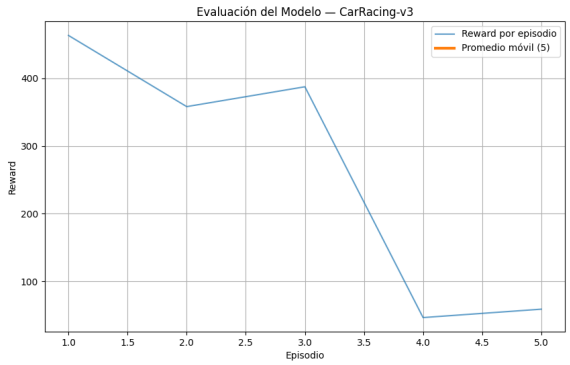

Figura 11. Evaluación (200): Dueling Double DQN.

VII. Discusión crítica

Con los resultados disponibles, el mejor desempeño global lo obtiene Double DQN a 1000 episodios (Tabla II), ya que alcanza el mayor retorno promedio y máximo. Esto es coherente con su mejora principal: desacoplar selección y evaluación de acciones en el *target* para mitigar la sobreestimación y estabilizar el aprendizaje [6]. A 200 episodios (Tabla I) el ranking es menos concluyente y la dispersión sigue siendo alta, lo que sugiere que ese régimen corresponde a una etapa temprana donde la política aún está en formación y la varianza por episodio domina [1].

El ejercicio muestra que 200 episodios es claramente insuficiente para comparar variantes con confianza en este entorno, mientras que 1000 episodios ofrece una señal más estable pero sigue siendo un presupuesto moderado para RL visual, donde el desempeño puede depender fuertemente de la semilla, de los hiperparámetros y del tiempo total de interacción [2], [1].

Limitaciones principales: (i) evaluación con pocos episodios por checkpoint ($n = 5$) y sin múltiples semillas, lo que limita inferencias estadísticas; (ii) discretización del control continuo, que impone una política restringida (12 acciones) y puede afectar el techo de desempeño; (iii) ausencia de una búsqueda sistemática de hiperparámetros y análisis de sensibilidad; y (iv) recursos de cómputo acotados (CPU), que condicionan la longitud del entrenamiento y el número de corridas.

VIII. Especificaciones y restricciones de implementación

Restricciones: se desactiva el renderizado interactivo para acelerar el entrenamiento (se usa el modo `rgb_array` para obtener observaciones); solo CPU como configuración típica; uso de frame skipping y mini-batches para controlar costo computacional.
Lenguaje y librerías: Python [15], Gymnasium [3], PyTorch [16], NumPy [17], Matplotlib [14], OpenCV [8], pandas [18], ImageIO [19] y pygame [20].

IX. Reproducibilidad

El repositorio del proyecto es: https://github.com/alexmancila/Proyect-RL-CarRacing-DQN/tree/main.
Ver Fig. 2 para una vista general de los módulos y del flujo de ejecución.
La configuración base (por defecto) se define en `src/configuracion.py`. Para replicar los experimentos: (i) instalar dependencias desde `requirements.txt`; (ii) ejecutar entrenamiento con `python main.py`; (iii) correr los scripts de evaluación en `tests/`. El cuaderno final en Jupyter documenta el flujo de ejecución y la ubicación de los artefactos de salida (modelos, métricas y gráficos) [13].

X. Conclusiones y trabajo futuro

Se cumplieron los objetivos planteados: se entrenaron agentes DQN y variantes en *CarRacing-v3*, se registraron métricas y gráficos de entrenamiento, y se evaluaron los modelos sin exploración ($\epsilon = 0$) con reportes reproducibles (tablas y figuras generadas por el proyecto). El logro principal es la implementación integrada y evaluable (DQN, Double y Dueling, incluyendo su combinación) sobre un entorno visual con control originalmente continuo, junto con una canalización de preprocesamiento y registro de resultados que permite comparar configuraciones de forma consistente.

Con la evidencia recopilada, Double DQN muestra el mejor desempeño promedio a 1000 episodios, en línea con su diseño para reducir sobreestimación en el *target* [6]. No obstante, estos hallazgos deben interpretarse como descriptivos: la evaluación usa pocos episodios ($n = 5$) y no incluye múltiples semillas, y además la discretización del control puede limitar el techo de desempeño y afectar comparaciones.

Como trabajo futuro: (i) aumentar el número de episodios de evaluación y usar múltiples semillas para reducir varianza [1]; (ii) realizar búsqueda y análisis de sensibilidad de hiperparámetros; (iii) incorporar mejoras de muestreo como *prioritized experience replay* [21] y probar arquitecturas más expresivas; y (iv) explorar métodos de control continuo para aprovechar la acción nativa del entorno (p.ej., DDPG [22] o SAC [23]).

XI. Participación y contribuciones

Koc Góngora, Luis Enrique: revisión del pipeline RL y código, análisis de resultados, documentación técnica, revisión bibliográfica, elaboración del informe.

Mancilla Antaya, Alex Felipe: integración y ajustes del pipeline, implementación de variantes (Double/Dueling), codificación e información técnica, repositorio Github, soporte en evaluación y automatización, revisión bibliográfica, revisión del informe y presentación.

Meléndez García, Herbert Antonio: revisión del pipeline RL y código, revisión bibliográfica, revisión del informe, elaboración de presentación y consolidación de resultados/figuras.

Paitán Cano, Dennis Jack: revisión del pipeline RL y código, revisión bibliográfica, revisión del informe, elaboración de presentación y organización de resultados.

XII. Transparencia (uso de herramientas)

Se utilizó Python [15] para el desarrollo y experimentación, y PyTorch [16] como framework principal para implementar y entrenar los modelos de aprendizaje por refuerzo profundo. El análisis y reporte de resultados se realizó con Jupyter Notebook [13], y la elaboración del informe se realizó en LaTeX [24]. Para apoyo en redacción y consistencia del documento se empleó asistencia de Inteligencia Artificial (IA) generativa como herramienta de productividad. Las decisiones técnicas, experimentos y resultados reportados provienen del código y documentación del repositorio.

Referencias

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[2] 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, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[3] Farama Foundation, "Gymnasium: A standard api for reinforcement learning environments," Software, accessed: 2026-01-14. [Online]. Available: https://gymnasium.farama.org/

[4] ——, "Gymnasium carracing-v3 environment documentation," Online documentation, accessed: 2026-01-14. [Online]. Available: https://gymnasium.farama.org/environments/box2d/car_racing/

[5] E. Catto, "Box2d: A 2d physics engine for games," Software, accessed: 2026-01-14. [Online]. Available: https://box2d.org/

[6] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10295

[7] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016. [Online]. Available: http://proceedings.mlr.press/v48/wangf16.html

[8] G. Bradski, "The opencv library," in *Dr. Dobb's Journal of Software Tools*, 2000. [Online]. Available: https://opencv.org/

[9] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," in *Machine Learning: Proceedings of the Eighth International Workshop*. Morgan Kaufmann, 1992.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations (ICLR)*, 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[11] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," *arXiv preprint arXiv:1705.08292*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1705.08292

[12] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019, appeared in ICLR 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1904.09237

[13] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks—a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, 2016, pp. 87–90. [Online]. Available: https://doi.org/10.3233/978-1-61499-649-1-87

[14] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. [Online]. Available: https://doi.org/10.1109/MCSE.2007.55

[15] Python Software Foundation, "Python language reference, version 3.x," Software, accessed: 2026-01-14. [Online]. Available: https://www.python.org/

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, project site: https://pytorch.org/ (accessed: 2026-01-14). [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[17] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[18] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference (SciPy)*, 2010, pp. 56–61. [Online]. Available: https://doi.org/10.25080/Majora-92bf1922-00a

[19] A. Silvester and contributors, "Imageio: Python library for reading and writing image data," Software, accessed: 2026-01-14. [Online]. Available: https://imageio.readthedocs.io/

[20] pygame contributors, "pygame," Software, accessed: 2026-01-14. [Online]. Available: https://www.pygame.org/

[21] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations (ICLR)*, 2016. [Online]. Available: https://arxiv.org/abs/1511.05952

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2016. [Online]. Available: https://arxiv.org/abs/1509.02971

[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, 2018. [Online]. Available: http://proceedings.mlr.press/v80/haarnoja18b.html

[24] The LaTeX Project, "LaTeX – a document preparation system," Software/Documentation, accessed: 2026-01-14. [Online]. Available: https://www.latex-project.org/