Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

TRABAJO FINAL DE MÁSTER

Área: Deep Reinforcement Learning Applications

Explorando técnicas avanzadas de RL para conducción autónoma

Autor: Víctor Díaz Bustos

Tutor: Raúl Parada Medina

Profesor: Susana Acedo Nadal

Granada, 26 de mayo de 2025

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Explorando técnicas avanzadas de RL para conducción autónoma
Nombre del autor:	Víctor Díaz Bustos
Nombre del colaborador/a docente:	Raúl Parada Medina
Nombre del PRA:	Susana Acedo Nadal
Fecha de entrega (mm/aaaa):	05/2025
Titulación o programa:	Master Ciencia de Datos
Área del Trabajo Final:	Área 5
Idioma del trabajo:	Español
Palabras clave	Aprendizaje por Refuerzo Profundo, Conducción autónoma, Entornos Simulados

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a todas aquellas personas e instituciones que, de una forma u otra, han contribuido a la realización de este Trabajo Final de Máster.

En primer lugar, deseo agradecer profundamente a mi tutor, **Raúl Parada Medina**, por su invaluable guía, sus consejos y su constante disposición y eficacia a lo largo de todo el desarrollo de este proyecto.

Asimismo, extiendo mi gratitud a la **Universitat Oberta de Catalunya (UOC)** y al programa del Máster Universitario en Ciencia de Datos, por brindarme la oportunidad de profundizar mis conocimientos y por los recursos ofrecidos para mi formación.

Un agradecimiento muy especial a mi **familia, pareja y amigos**, por su incondicional apoyo, comprensión y ánimo constante, especialmente en los momentos de mayor dedicación y esfuerzo. Sin su aliento, este camino habría sido mucho más arduo.

A mis **amigos y compañeros**, por las conversaciones estimulantes, el apoyo moral y por recordarme la importancia de los descansos.

Finalmente, agradezco a la comunidad de desarrolladores de software de código abierto, en particular a los creadores y mantenedores de herramientas como SUMO, Python, PyTorch y CodeCarbon, cuyo trabajo ha sido indispensable para la ejecución de este proyecto.

Víctor Díaz Bustos

Granada, mayo de 2025

Resumen

Lograr una **conducción autónoma** completamente segura representa uno de los mayores desafíos tecnológicos, con el potencial de transformar tanto la industria automovilística como nuestra vida diaria. Un sistema de movilidad eficiente y sin accidentes redefiniría nuestra percepción del transporte, tanto en entornos urbanos como en trayectos de larga distancia. Los recientes avances en telecomunicaciones, sensores e **Inteligencia Artificial** acercan cada vez más esta visión a la realidad.

Este trabajo aborda el reto de conseguir una conducción fluida y precisa dentro de un entorno simulado de conducción autónoma. Para ello, se emplearán métodos de Aprendizaje por Refuerzo Profundo (DRL, **Deep Reinforcement Learning**), una disciplina que combina el Aprendizaje por Refuerzo (RL, **Reinforcement Learning**) con Redes Neuronales Profundas (DL, **Deep Learning**).

El simulador seleccionado para el desarrollo del proyecto es **SUMO**, que permite simular tráfico vehicular a gran escala y facilita la integración de algoritmos personalizados para la gestión de la movilidad.

Los resultados indican que el algoritmo de DRL **SAC** demostró las capacidades de aprendizaje más prometedoras en comparación con los algoritmo **DDQN** y **PPO**. Sin embargo, todos los agentes, incluido SAC, exhibieron inestabilidad y no lograron completar consistentemente episodios completos sin incidentes críticos, lo que subraya los desafíos persistentes en seguridad y robustez. PPO experimentó un colapso de su política, mientras que DDQN mostró un aprendizaje limitado e inestable. El estudio también incluye una **estimación de la huella de carbono** de los procesos de entrenamiento utilizando **CodeCarbon**. Este trabajo establece un marco de simulación funcional y proporciona información valiosa sobre el rendimiento comparativo y los desafíos de estos algoritmos de DRL, ofreciendo una base para futuras investigaciones centradas en la optimización de hiperparámetros, el refinamiento de la función de recompensa y la mejora de la representación del estado, particularmente para el algoritmo SAC.

Palabras clave: Conducción Autónoma, Aprendizaje por Refuerzo Profundo, SUMO, Entornos Simulados.

Abstract

Achieving completely safe **autonomous driving** represents one of the greatest technological challenges, with the potential to transform both the automotive industry and our daily lives. An efficient, accident-free mobility system would redefine our perception of transportation, both in urban environments and long-distance travel. Recent advances in telecommunications, sensors, and **Artificial Intelligence** are bringing this vision closer to reality.

This work addresses the challenge of achieving smooth and precise driving within a simulated autonomous driving environment. To this end, **Deep Reinforcement Learning** (DRL) methods will be employed — a discipline that combines **Reinforcement Learning** (RL) with **Deep Learning** (DL).

The simulator chosen for the project's development is **SUMO**, which allows large-scale vehicular traffic simulation and facilitates the integration of custom algorithms for mobility management.

The results indicate that the DRL algorithm **SAC** demonstrated the most promising learning capabilities compared to the **DDQN** and **PPO** algorithms. However, all agents, including SAC, exhibited instability and failed to consistently complete full episodes without critical incidents, highlighting persistent challenges in safety and robustness. PPO experienced a collapse of its policy, while DDQN showed limited and unstable learning. The study also includes a **carbon footprint estimation** of the training processes using **CodeCarbon**. This work establishes a functional simulation framework and provides valuable insights into the comparative performance and challenges of these DRL algorithms, offering a foundation for future research focused on hyperparameter optimization, reward function refinement, and improvement of state representation, particularly for the SAC algorithm.

Keywords: Autonomous Driving, Deep Reinforcement Learning, SUMO, Simulated Environments.

Índice general

Índice de figuras

Índice de cuadros

1. Introducción

1.1. Contexto y motivación

La **conducción autónoma** es uno de los campos de mayor impacto en la investigación de **sistemas inteligentes** de transporte. La simulación de escenarios urbanos permite evaluar, de manera segura y controlada, estrategias de conducción basadas en algoritmos de **aprendizaje por refuerzo**.

Existen **seis niveles de conducción autónoma**, que van desde el nivel 0 (sin ningún tipo de automatización) hasta el nivel 5, en el que el vehículo opera de manera totalmente autónoma en todas las condiciones sin intervención humana [5].

Actualmente, la mayoría de los vehículos comerciales disponen de sistemas de nivel 2, que ofrecen asistencia parcial en dirección, aceleración y frenado. Algunos prototipos y modelos experimentales ya han empezado a incorporar capacidades de nivel 3, permitiendo cierta automatización en entornos controlados, y en áreas georrestringidas se están probando vehículos de nivel 4. Empresas como **Waymo** y **Cruise** han desplegado servicios de robotaxi en áreas georestringidas (como algunas zonas de Phoenix o San Francisco, respectivamente) [6] [7]. En estos entornos controlados, los vehículos pueden operar de forma completamente autónoma sin intervención humana, siempre y cuando se mantengan dentro de los límites geográficos y de condiciones predefinidas. Sin embargo, alcanzar la plena autonomía (nivel 5) sigue siendo un reto tanto tecnológico como regulatorio.

Sobre estos niveles y las tecnologías de IA comentadas se hablará en detalle en el capítulo de estado del arte.

La elección de la temática para el trabajo fin de máster ha estado motivada por dos razones principales. Por un lado, DL y RL han sido dos de las asignaturas más interesantes cursadas durante el máster y este proyecto me da la oportunidad de ampliar mis conocimientos en estos campos. Por otra parte, actualmente trabajo en el sector de la automoción, por lo que adquirir experiencia en un campo con una proyección como la conducción autónoma puede beneficiarme laboralmente

El presente TFM propone el desarrollo de un entorno en SUMO que integra un vehículo controlable mediante la API TraCI, permitiendo:

- Registrar el estado del vehículo y su entorno (posición, velocidad, información de carril, detección de vehículos cercanos, etc.).
- Penalizar comportamientos no deseados (colisiones, teletransportación, maniobras forzadas) y recompensar conductas seguras.

- Reasignar rutas de forma dinámica cuando el vehículo alcanza el final del tramo, simulando condiciones reales de adaptación en la conducción.

Esta propuesta es relevante tanto desde el punto de vista científico como práctico, ya que aporta un marco de pruebas para algoritmos de RL en entornos complejos y contribuye al avance de la conducción autónoma).

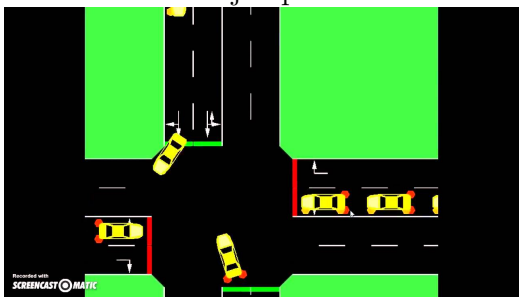
La elección de este TFM surge de mi fuerte interés en explorar el potencial del aprendizaje por refuerzo y del aprendizaje profundo, áreas en las que he profundizado a través de las asignaturas *"Deep Learning"* y *Reinforcement Learning"*. Aunque mi experiencia práctica en simulaciones de conducción autónoma es aún incipiente, estoy motivado por aplicar técnicas avanzadas de RL a problemas reales, especialmente en el ámbito del transporte urbano. Considero que la integración de simuladores como SUMO con métodos de aprendizaje por refuerzo abre la puerta a soluciones innovadoras que pueden mejorar significativamente la seguridad y eficiencia de los sistemas de movilidad, convirtiéndose en una contribución valiosa tanto a nivel académico como práctico.

1.2. Objetivos

Este trabajo final de máster se centra en el diseño, implementación y evaluación de un entorno de simulación para la conducción autónoma utilizando SUMO (Simulation of Urban MObility) y la interfaz TraCI. Se desarrolla un entorno en Python en el que un vehículo controlable es entrenado mediante algoritmos de aprendizaje por refuerzo (RL) para aprender a conducir de forma segura y eficiente.

El sistema penaliza eventos adversos, como colisiones, teletransportaciones y maniobras inválidas, y recompensa comportamientos deseables (por ejemplo, mantener velocidad adecuada, recorrer distancias y alcanzar el final de la ruta). Además, se implementa la reasignación dinámica de rutas, de modo que al llegar al final de un tramo, el vehículo se redirige hacia una nueva ruta aleatoria desde su posición actual, imitando situaciones reales en las que el conductor debe adaptarse al entorno.

Figura 1: Situación de ejemplo del simulador SUMO.



En la Figura 1 se aprecia una situación de ejemplo del simulador SUMO.

Para alcanzar este objetivo, haremos uso de métodos pertenecientes a dos de las ramas de la IA más prometedoras: el **Aprendizaje Profundo** (DL) y el **Aprendizaje por Refuerzo**(RL).

En DL se intenta, a través de un algoritmo automático jerárquico, emular la forma de aprender del cerebro humano. Los modelos de DL aprenden por sí mismos y sin reglas previamente establecidas mediante una fase de entrenamiento. Debido a esta similitud con el aprendizaje humano, las redes utilizadas en DL se denominan redes neuronales. Estas redes son capaces de obtener conceptos abstractos y complejos "mezclando" otros más simples a lo largo de sus capas.

Por otra parte, en RL un agente aprende a tomar decisiones correctas a través de prueba y error, interaccionando con el entorno. Dado un estado, el agente realizará una acción en base a recompensas generadas por dicho entorno. Esta acción dará lugar a un nuevo estado del agente. Escenificamos a continuación esta lógica con un ejemplo simple. El agente es el coche autónomo. Una recompensa podría ser negativa al llegar al borde de la carretera. Al recibir esta recompensa el coche realizará la acción de girar el volante para mantenerse en la carretera.

A lo largo de este trabajo, aplicaremos técnicas de **Aprendizaje por Refuerzo Profundo** (DRL). Estos modelos aúnan las cualidades de las técnicas expuestas anteriormente. Siguiendo la lógica del RL, en DRL la capacidad de aprendizaje del agente viene dada por una red neuronal profunda. De esta forma, el agente podrá tomar decisiones a partir de datos de entrada no estructurados como son las imágenes.

1.2.1. Objetivo principal

Desarrollar e implementar un entorno de simulación basado en SUMO que permita entrenar a un agente de aprendizaje por refuerzo para la conducción autónoma, comparando diferentes algoritmos de RL que buscan maximizar la seguridad y eficiencia en la circulación.

1.2.2. Objetivos secundarios

- Definir y desarrollar el entorno de simulación: Crear una interfaz en Python utilizando TraCI que permita controlar un vehículo manual en SUMO, extrayendo información del entorno y gestionando rutas dinámicas.
- Diseñar la función de recompensa: Establecer penalizaciones para colisiones, teletransportaciones y maniobras inválidas, y recompensas para la velocidad adecuada, distancia recorrida y llegada exitosa al destino.
- Implementar el cambio dinámico de rutas: Permitir que, al finalizar un tramo, el vehículo reciba una nueva ruta aleatoria desde su posición actual.

- Recopilar y procesar el estado del vehículo y su entorno: Obtener, en cada paso, información detallada del vehículo (posición, velocidad, ángulo, carril, etc.) y de los vehículos cercanos, para alimentar al algoritmo de RL.
- Evaluar y comparar distintos algoritmos de RL: Analizar el desempeño del agente en diferentes escenarios y ajustar la función de recompensa para mejorar la conducción.

1.3. Sostenibilidad, diversidad y desafíos ético/sociales

El proyecto se centra en el desarrollo de un entorno de simulación para la conducción autónoma utilizando técnicas de aprendizaje por refuerzo. A continuación se evalúan los posibles impactos en distintas dimensiones [18]:

Sostenibilidad Aunque el desarrollo se realiza en un entorno simulado y, en principio, no implica un consumo directo de recursos a gran escala, el enfoque del proyecto tiene un potencial impacto positivo en la movilidad urbana sostenible. La aplicación de algoritmos de RL en la conducción autónoma podría, en escenarios reales, contribuir a optimizar el flujo de tráfico, reducir los atascos y, por ende, disminuir el consumo de combustible y las emisiones contaminantes. Además, una mejor eficiencia en la conducción se alinea con algunos de los Objetivos de Desarrollo Sostenible (ODS), como el ODS 11 (Ciudades y comunidades sostenibles) y el ODS 9 (Industria, innovación e infraestructura). Se analizará en el desarrollo cómo la optimización de la conducción puede influir en la reducción de la huella ecológica del transporte.

Comportamiento ético y responsabilidad social Al tratarse de un entorno de simulación para la investigación en aprendizaje por refuerzo, el proyecto es en gran medida técnico. Sin embargo, su aplicación en la conducción autónoma tiene implicaciones éticas y sociales importantes. Por un lado, el desarrollo de sistemas de conducción más seguros y eficientes puede reducir el número de accidentes y mejorar la seguridad vial, lo que tiene un impacto positivo en la sociedad. Por otro lado, la integración de estas tecnologías en entornos reales exige cumplir con las normativas relativas a la privacidad, la seguridad de los datos y la responsabilidad profesional. En este sentido, el proyecto se desarrollará siguiendo los principios éticos de la profesión y se tendrá en cuenta la normativa vigente en materia de protección de datos, garantizando un tratamiento adecuado de la información personal si llegara a emplearse en futuros desarrollos o validaciones con datos reales.

Diversidad, género y derechos humanos El carácter técnico del proyecto implica que, en su fase de investigación y desarrollo, el impacto directo en términos de diversidad, género o derechos humanos es limitado. No obstante, es importante considerar que la aplicación

de tecnologías de conducción autónoma puede influir en la accesibilidad y en la seguridad de todos los usuarios de la vía, incluyendo personas con discapacidad o en situación de vulnerabilidad. Además, la incorporación de sistemas inteligentes en el transporte debe diseñarse de manera inclusiva, garantizando que la tecnología sea accesible y segura para la mayor diversidad de usuarios. Se analizará, en la fase de conclusiones, cómo estas tecnologías pueden contribuir a la igualdad y a la mejora de la calidad de vida en entornos urbanos.

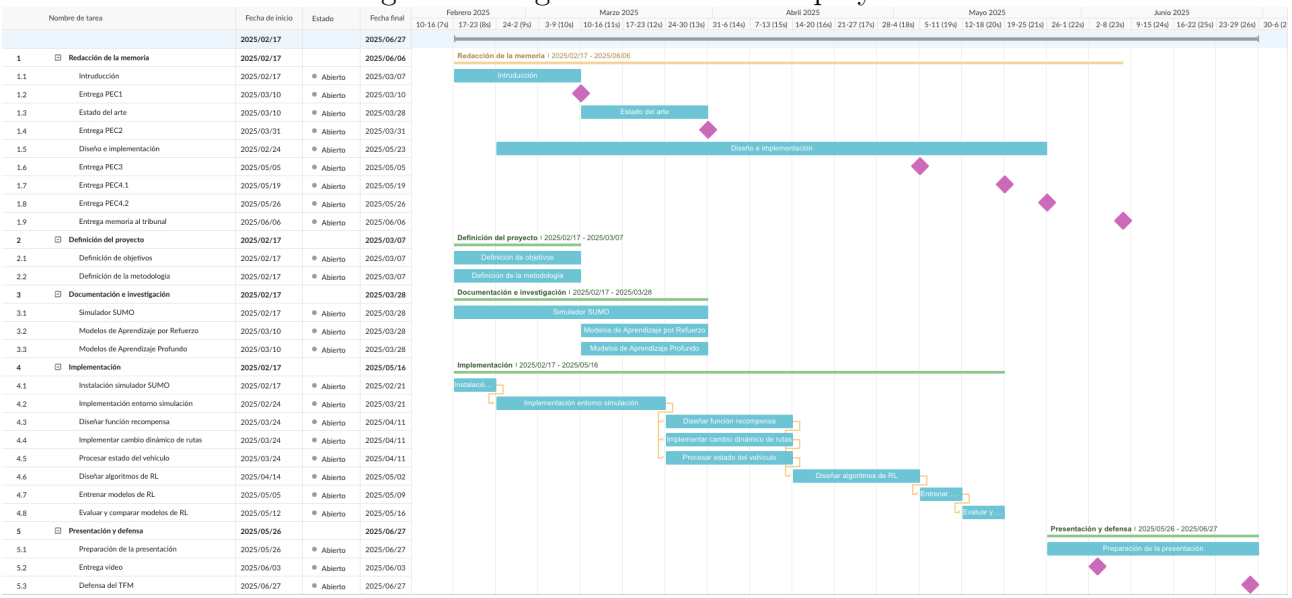
1.4. Enfoque y metodología

El TFM se desarrollará mediante las siguientes estrategias:

- **Investigación bibliográfica**: Revisión de literatura en aprendizaje por refuerzo, simulación de tráfico y conducción autónoma, apoyándose en artículos científicos, tesis y documentación oficial de SUMO y TraCI.
- **Desarrollo del entorno de simulación**:
  - **Implementación en Python**: Se desarrollará un entorno que utilice la API TraCI para controlar un vehículo en SUMO, extrayendo información del estado, aplicando acciones y evaluando recompensas.
  - **Gestión de rutas**: El entorno permitirá asignar rutas válidas y, cuando se alcance el final de la ruta, se forzará una maniobra (incluso si es inválida) para simular una situación de colisión o error, que se penalizará y reiniciará el episodio.
- **Entrenamiento de algoritmos de RL**: Se probarán distintos métodos como ***DDQN*** *(Dueling Deep Q-Network),* ***PPO*** *(Proximal Policy Optimization),* ***A3C*** *(Asynchronous Advantage Actor Critic)* o ***SAC*** *(Soft Actor-Critic)*, en los que se profundizará posteriormente en el apartado de *Implementacion*, para entrenar al agente en el entorno desarrollado.
- **Validación y análisis**: Se analizarán los resultados del entrenamiento, comparando métricas de seguridad y eficiencia, y se realizarán ajustes en la función de recompensa y en la configuración del entorno.

1.5. Planificación

Figura 2: Diagrama de Gantt del proyecto.



	Nombre de tarea	Fecha de inicio	Estado	Fecha final
		2025/02/17		2025/06/27
1	**Redacción de la memoria**	2025/02/17		2025/06/06
1.1	Intruducción	2025/02/17	Abierto	2025/03/07
1.2	Entrega PEC1	2025/03/10	Abierto	2025/03/10
1.3	Estado del arte	2025/03/10	Abierto	2025/03/28
1.4	Entrega PEC2	2025/03/31	Abierto	2025/03/31
1.5	Diseño e implementación	2025/02/24	Abierto	2025/05/23
1.6	Entrega PEC3	2025/05/05	Abierto	2025/05/05
1.7	Entrega PEC4.1	2025/05/19	Abierto	2025/05/19
1.8	Entrega PEC4.2	2025/05/26	Abierto	2025/05/26
1.9	Entrega memoria al tribunal	2025/06/06	Abierto	2025/06/06
2	**Definición del proyecto**	2025/02/17		2025/03/07
2.1	Definición de objetivos	2025/02/17	Abierto	2025/03/07
2.2	Definición de la metodología	2025/02/17	Abierto	2025/03/07
3	**Documentación e investigación**	2025/02/17		2025/03/28
3.1	Simulador SUMO	2025/02/17	Abierto	2025/03/28
3.2	Modelos de Aprendizaje por Refuerzo	2025/03/10	Abierto	2025/03/28
3.3	Modelos de Aprendizaje Profundo	2025/03/10	Abierto	2025/03/28
4	**Implementación**	2025/02/17		2025/05/16
4.1	Instalación simulador SUMO	2025/02/17	Abierto	2025/02/21
4.2	Implementación entorno simulación	2025/02/24	Abierto	2025/03/21
4.3	Diseñar función recompensa	2025/03/24	Abierto	2025/04/11
4.4	Implementar cambio dinámico de rutas	2025/03/24	Abierto	2025/04/11
4.5	Procesar estado del vehículo	2025/03/24	Abierto	2025/04/11
4.6	Diseñar algoritmos de RL	2025/04/14	Abierto	2025/05/02
4.7	Entrenar modelos de RL	2025/05/05	Abierto	2025/05/09
4.8	Evaluar y comparar modelos de RL	2025/05/12	Abierto	2025/05/16
5	**Presentación y defensa**	2025/05/26		2025/06/27
5.1	Preparación de la presentación	2025/05/26	Abierto	2025/06/27
5.2	Entrega video	2025/06/03	Abierto	2025/06/03
5.3	Defensa del TFM	2025/06/27	Abierto	2025/06/27

En la Figura 2 se representa el diagrama de Gantt del proyecto, donde se pueden distinguir las diferentes partes del trabajo y su planificación temporal, que comprende desde mediados de febrero hasta finales de junio de 2025. Este está compuesto por los siguientes items:

- **Grupos o Fases**: Son bloques grandes que engloban un conjunto de tareas relacionadas (por ejemplo, "Redacción de la memoria", "Implementación", etc.). Permiten organizar el proyecto en secciones o etapas lógicas.
- **Hitos**: Aparecen como puntos o diamantes que representan un suceso clave o entrega importante (por ejemplo, una fecha de revisión o la defensa del TFM). Normalmente no tienen duración, solo una fecha concreta.
- **Tareas**: Se muestran como barras horizontales que indican la duración (fecha de inicio y fin) de cada actividad. Suelen incluir una breve descripción de la acción a realizar (por ejemplo, "Diseñar función recompensa").
- **Estado**: Suele ser una columna adicional en la que se refleja el progreso (p.ej., .ªbierto", "En curso", "Hecho", "Cerrado"), lo que facilita el seguimiento del avance en cada tarea o fase.

Se puede observar como algunas de las tareas se realizan de forma paralela.
La descripción de las etapas es la siguiente:

- **Redacción de la memoria**: La memoria se irá redactando a lo largo de toda la duración del proyecto. Existiendo diferentes hitos que corresponden a los entregables exigidos para superar la asignatura.

- **Definición del proyecto**: Durante esta etapa quedan definidos los objetivos y la metodología a seguir durante el TFM.

- **Documentación e investigación**: Las tareas dentro de este apartado tienen como objetivo documentarse sobre todas las posibilidades del simulador SUMO, así como de la teoría de los modelos que se aplicarán en el proyecto.

- **Implementación**: Tras instalar el simulador, se comenzará a implementar el entorno y los modelos de DRL, así como su entrenamiento, evaluación y comparación. Una parte importante será la optimización de los parámetros para conseguir que los modelos se comporten de la mejor manera y sean lo más generales posible. El objetivo será conseguir una conducción autónoma correcta y fluida.

- **Presentación y defensa**: Una vez implementado el proyecto y redactada la memoria, se procede a preparar la presentación que se utilizará en la defensa del TFM. Con los hitos de la entrega del video y la defensa del proyecto se dará por concluido el trabajo fin de máster.

1.6. Resumen de los productos del proyecto

Los principales productos generados en este proyecto incluyen: un entorno de simulación configurable basado en SUMO y Python para la conducción autónoma; implementaciones de los algoritmos de Aprendizaje por Refuerzo Profundo DDQN, PPO y SAC adaptados a dicho entorno; el código fuente completo del proyecto disponible públicamente; y un análisis comparativo de los resultados de entrenamiento y evaluación de los agentes. Los detalles de cada uno de estos componentes se desarrollan en los capítulos subsiguientes.

1.7. Breve descripción de los demás capítulos del informe

El presente informe se estructura de la siguiente manera:

1. La Sección 1, **Introducción**, establece el contexto y la motivación del proyecto, define los objetivos principales y secundarios, discute la sostenibilidad y los desafíos ético-sociales relacionados, y presenta el enfoque metodológico y la planificación seguida.

2. La Sección 2, **Estado del arte**, revisa la literatura científica y técnica existente sobre conducción autónoma, Aprendizaje por Refuerzo Profundo (DRL) y el uso de simuladores como SUMO. Se identifican los avances relevantes y los vacíos en la investigación que este trabajo busca abordar.

3. La Sección 3, **Diseño e implementación del trabajo**, detalla la arquitectura del sistema desarrollado, incluyendo la instalación y configuración del entorno SUMO, la interfaz de programación TraCI, y la implementación específica de los algoritmos de DRL seleccionados: Dueling Double Deep Q-Network (DDQN), Asynchronous Advantage Actor-Critic (A3C), Proximal Policy Optimization (PPO) y Soft Actor-Critic (SAC).

4. La Sección 4, **Descripción del experimento**, describe la configuración experimental utilizada para entrenar y evaluar los agentes. Se detallan el entorno de simulación específico, los hiperparámetros principales de cada algoritmo, las métricas registradas para el seguimiento del rendimiento y la plataforma experimental sobre la cual se realizaron los cómputos. Además, se incluye una estimación de la huella de carbono asociada al proceso de entrenamiento de los modelos.

5. La Sección 5, **Resultados del entrenamiento**, presenta y analiza en profundidad las curvas de aprendizaje, la evolución de las métricas clave y el rendimiento general observado durante la fase de entrenamiento para cada uno de los algoritmos implementados (DDQN, PPO y SAC), discutiendo su convergencia y estabilidad.

6. La Sección 6, **Comparativa de rendimiento en evaluación**, ofrece un análisis comparativo del desempeño de los agentes entrenados sobre un conjunto de rutas de evaluación estandarizadas y fijas. Se presentan los resultados por episodio, una tabla resumen de métricas y se discuten las diferencias observadas en el rendimiento final de los algoritmos.

7. La Sección 7, **Análisis de problemas comunes y soluciones intentadas**, examina los desafíos recurrentes encontrados durante la fase de experimentación. Se enfoca en aspectos críticos como el diseño iterativo de la función de recompensa, la adecuación de la representación del estado, la estabilidad algorítmica y la interacción con el entorno SUMO, detallando las soluciones y ajustes implementados.

8. Finalmente, la Sección 8, **Conclusiones y trabajo futuro**, resume los principales hallazgos y contribuciones del proyecto, realiza una evaluación crítica del grado de consecución de los objetivos propuestos y discute las limitaciones encontradas. Con base en esto, se proponen diversas líneas de investigación y desarrollo futuras para continuar avanzando en este campo.

2. Estado del arte

2.1. Introducción al estado del arte

La conducción autónoma representa uno de los campos de investigación más activos y desafiantes en el ámbito de la **Inteligencia Artificial (IA)** y la **robótica**. Numerosos trabajos abordan esta problemática desde perspectivas diversas, incluyendo la **visión por computador**, la **fusión de sensores** y la **planificación de trayectorias**. En los últimos años, las técnicas de Aprendizaje por Refuerzo Profundo (**DRL**) han cobrado relevancia para entrenar agentes capaces de tomar decisiones en tiempo real [9, 11].

En esta sección, se revisan las investigaciones y soluciones más destacadas relacionadas con:

- La conducción autónoma en simuladores (especialmente SUMO).
- El uso de algoritmos de Aprendizaje por Refuerzo para optimizar la toma de decisiones.
- El estado de la investigación en cuanto a limitaciones, éxitos y retos actuales.

Este estudio bibliográfico constituye la base para la propuesta de este TFM, justificando las decisiones metodológicas y los objetivos definidos.

2.2. Relevancia y motivación de la problemática

El avance hacia una conducción autónoma **segura y eficiente** representa uno de los mayores retos tecnológicos y sociales de la actualidad. La conducción autónoma no sólo implica el dominio de la **física del vehículo**, sino también la capacidad de **interactuar** de forma segura en entornos urbanos complejos, donde la densidad del tráfico, la diversidad de actores (vehículos, peatones, ciclistas) y las condiciones cambiantes hacen que la toma de decisiones en tiempo real sea fundamental. En este contexto, la simulación de escenarios de tráfico realistas se vuelve indispensable para evaluar y perfeccionar las estrategias de control, ya que permite replicar situaciones de riesgo sin exponer a personas o bienes a peligros reales [13, 14].

Numerosos estudios han demostrado el potencial del Aprendizaje por Refuerzo Profundo (DRL) para entrenar agentes autónomos en entornos controlados. Sin embargo, gran parte de la literatura se ha enfocado en simuladores con escenarios relativamente simples o en circuitos cerrados, lo que limita la validez de los resultados cuando se pretende aplicar la solución a contextos urbanos reales. Por ejemplo, trabajos como los presentados por Dosovitskiy et al. [13] y Sallab et al. [14] han logrado avances notables en el entrenamiento de agentes para tareas de maniobra y toma de decisiones en entornos simplificados, pero estos enfoques suelen prescindir de la complejidad inherente a la gestión de flujos de tráfico a gran escala.

La integración de SUMO (Simulation of Urban MObility) con algoritmos de DRL se presenta como una solución innovadora y necesaria para cubrir este vacío. SUMO ofrece la capacidad de **modelar escenarios urbanos a gran escala**, lo que permite incorporar aspectos como la interacción entre múltiples vehículos, la variabilidad en las condiciones del tráfico y la influencia de infraestructuras urbanas (intersecciones, semáforos, carriles exclusivos, etc.). Esta capacidad de simulación realista es crucial para evaluar estrategias de control que optimicen no sólo la maniobra de un vehículo, sino también decisiones relacionadas con la elección de rutas, el control de velocidad y la interacción con el resto de actores del tráfico.

Además, al combinar estas dos áreas (simulación de tráfico y DRL) se abre la posibilidad de desarrollar soluciones que puedan adaptarse dinámicamente a situaciones imprevistas y, en última instancia, contribuir a reducir accidentes y mejorar la eficiencia del transporte urbano. Este enfoque también aborda aspectos éticos y de seguridad, ya que permite probar y validar los algoritmos en un entorno controlado antes de su eventual aplicación en entornos reales.

Por lo tanto, en este TFM se propone explorar y validar la integración de SUMO y técnicas de DRL para desarrollar un agente que no solo realice maniobras de conducción a nivel individual, sino que también tome decisiones complejas en un entorno urbano simulado. Este trabajo pretende contribuir al estado del arte proporcionando un marco de simulación más realista y escalable, que permita una mejor extrapolación de los resultados a escenarios del mundo real.

2.3. Trabajos previos y justificación

2.3.1. Uso de simuladores de tráfico

- **SUMO**: *SUMO (Simulation of Urban MObility)* es un simulador de tráfico de código abierto ampliamente reconocido por su capacidad para modelar escenarios urbanos complejos a gran escala. Diversos estudios han demostrado su flexibilidad para simular semáforos, rutas de autobuses, y variaciones en la densidad de tráfico [16]. Sin embargo, la mayoría de las implementaciones actuales se centran en la optimización de sistemas de control de tráfico (por ejemplo, el ajuste de ciclos de semáforos o la optimización de rutas) sin explorar en profundidad el control individual de vehículos mediante algoritmos de Aprendizaje por Refuerzo. Este TFM se diferencia en que propone integrar SUMO con algoritmos de DRL para entrenar agentes que tomen decisiones de conducción a nivel individual, interactuando con otros vehículos y adaptándose a condiciones dinámicas en entornos urbanos reales. Además, el uso de SUMO permite aprovechar una red vial detallada y realista, lo cual es fundamental para validar modelos de RL en escenarios complejos.

- **Otros simuladores**: Simuladores como *CARLA* se centran en la simulación de la física

del vehículo y en la percepción visual, proporcionando un entorno rico para el procesamiento de imágenes y la detección de objetos [13]. Estos entornos son ideales para entrenar algoritmos basados en visión, pero suelen limitarse a escenarios con una menor escala de tráfico o en entornos controlados (p. ej., circuitos cerrados). Además, la alta demanda de recursos computacionales para renderizar gráficos realistas hace que su uso en estudios a gran escala sea más complejo. En contraste, SUMO se orienta hacia la simulación de flujos de tráfico a nivel macro, permitiendo evaluar el comportamiento de múltiples agentes y la gestión global de rutas, lo que resulta esencial para abordar la conducción autónoma en entornos urbanos densos sin requerir hardware de última generación.

2.3.2. Algoritmos de Aprendizaje por Refuerzo

- **Aprendizaje por Refuerzo Clásico**: Técnicas tradicionales como **Q-Learning** o **SARSA** han sido aplicadas en entornos con **estados y acciones discretos**, en los que la complejidad del entorno se reduce significativamente. Aunque estos métodos son útiles para ilustrar conceptos básicos y se han empleado en escenarios de baja dimensión, su **escalabilidad es limitada** y se vuelven inadecuados para entornos con altos grados de incertidumbre o dimensiones continuas, como es el caso de la conducción autónoma.

- **Aprendizaje por Refuerzo Profundo (DRL)**: Los **métodos de DR**L, que combinan Aprendizaje por Refuerzo con Redes Neuronales Profundas, han permitido superar las limitaciones de los algoritmos clásicos. Técnicas como **Dueling Deep Q-Network (DDQN)**, **Proximal Policy Optimization (PPO)**, **Soft Actor-Critic (SAC)** y **Asynchronous Advantage Actor Critic (A3C)** han demostrado un rendimiento sobresaliente en tareas complejas de toma de decisiones [14, 9]. Estos algoritmos permiten la **aproximación de funciones de valor y políticas en espacios de alta dimensión**, lo que los hace aptos para entornos de conducción donde las variables (como posición, velocidad, estados de tráfico, etc.) son continuas y altamente interrelacionadas. No obstante, la mayoría de los estudios en DRL se centran en entornos simplificados o en simuladores con escasa interacción de tráfico, lo que subraya la necesidad de explorar su aplicación en entornos más realistas y complejos, como el propuesto en este TFM.

2.3.3. Retos actuales

- **Escalabilidad**: La conducción autónoma no depende únicamente de la maniobra de un solo vehículo, sino de la interacción con numerosos agentes en un entorno urbano. La escalabilidad se vuelve crítica al aumentar la densidad de tráfico, ya que el modelo debe procesar y reaccionar ante múltiples estímulos de forma simultánea. Este reto es aún más

evidente cuando se intenta entrenar algoritmos de DRL en entornos complejos, donde el espacio de estados y acciones se expande considerablemente.

- **Generalización**: Uno de los mayores desafíos en la aplicación de algoritmos de DRL es la capacidad del modelo para generalizar a situaciones que no fueron contempladas durante el entrenamiento. Muchos modelos entrenados en escenarios reducidos fallan al enfrentarse a condiciones reales y variables, lo que hace imprescindible diseñar entornos de simulación que capturen la complejidad del tráfico urbano. La generalización es clave para asegurar que el agente pueda operar de forma segura y eficiente en diferentes contextos y condiciones imprevistas.

- **Validación en entornos realistas**: Aunque existen numerosos simuladores, la mayoría se centra en escenarios controlados o simplificados. Es fundamental contar con un simulador que reproduzca la complejidad del tráfico urbano, incluyendo semáforos, transporte público, múltiples carriles y diversas condiciones de tráfico. Esta validación realista no solo facilita el entrenamiento de modelos robustos, sino que también permite evaluar el impacto de los algoritmos en la seguridad vial y en la eficiencia del transporte, aspectos críticos para la futura implementación de vehículos autónomos en entornos reales.

2.4. Vacíos detectados

A partir de la literatura revisada, se observa que:

- **Pocos trabajos combinan SUMO y DRL** para abordar la conducción autónoma a gran escala, centrándose principalmente en la optimización de semáforos o enrutamientos.

- Falta de estudios que evalúen el comportamiento de un agente autónomo en **escenarios urbanos completos**, donde el flujo de tráfico se gestione de forma realista.

2.5. Metodología de búsqueda y fuentes consultadas

Para el desarrollo de este estado del arte se han seguido las **pautas metodológicas de Oates** [15], utilizando principalmente las técnicas de ***"document-based research"*** y ***"survey"***:

- Se han empleado **palabras clave** como *"SUMO autonomous driving", "reinforcement learning traffic management", "deep reinforcement learning conduction urbana"* en Google Scholar y Web of Science.

- Se han consultado **repositorios de la UOC y bibliotecas virtuales de otras universidades** para acceder a artículos y libros especializados.

- Se ha utilizado el gestor de bibliografía de **Latex** para organizar las referencias y extraer citas de forma consistente.

2.6. Trabajos previos

Existen varios artículos que abordan la temática de conducción autónoma a través de aprendizaje por refuerzo profundo, pero uno de los que más información aporta sobre este tema es *"Deep Reinforcement Learning for Autonomous Driving: A Survey"* [9]. El artículo científico está firmado por **B. Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani y Patrick Pérez**. En él se explican con detalle las diferentes aproximaciones que se han hecho en los últimos años en este campo.

Para la realización de este proyecto se ha tomado como inspiración el trabajo fin de máster *"Aplicación de técnicas de aprendizaje por refuerzo profundo para conducción autónoma en el simulador CARLA"* [2], firmado por **Pablo López Ladrón de Guevara**. Este TFM explora la aplicación de técnicas de aprendizaje por refuerzo profundo (DRL) para la conducción autónoma en el simulador CARLA. Se implementan y comparan dos arquitecturas de red Dueling DQN: una que procesa directamente imágenes de cámaras de segmentación semántica junto con datos de conducción (CNN + MLP), y otra que utiliza un Variational Autoencoder (VAE) para reducir la dimensionalidad de las imágenes antes de la entrada. Los resultados muestran que la arquitectura CNN + MLP entrena más rápido y logra una mayor precisión en la ruta de entrenamiento específica, con menor huella de carbono. Sin embargo, ambos modelos presentan dificultades para generalizar el comportamiento de conducción a rutas nuevas y no entrenadas, sugiriendo la necesidad de mejoras en la capacidad de generalización y explorando otras arquitecturas DRL, como las basadas en políticas o con espacios de acción continuos, para futuras investigaciones.

Otro artículo que investiga una línea muy similar a la seguida en este proyecto es *"Deep Reinforcement Learning Based Control for Autonomous Vehicles in CARLA"* [10], firmado por **Óscar Pérez-Gil, Rafael Barea, Elena López-Guillén, Luis M. Bergasa, Carlos Gómez-Huélamo, Rodrigo Gutiérrez y Alejandro Díaz-Díaz**. Esta investigación será utilizada durante el proyecto para hacer una comparativa entre resultados y tomarla de base para buscar nuevos algoritmos.

2.7. Conclusiones del estado del arte

La revisión bibliográfica confirma la **importancia de la simulación** como herramienta para acelerar la investigación en conducción autónoma. Además, se evidencia la necesidad de **integrar algoritmos de Aprendizaje por Refuerzo Profundo** con entornos de tráfico a gran escala, un área que no está suficientemente explorada y que presenta un alto potencial de contribución. Por tanto, el presente TFM se centrará en:

1. **Desarrollar** un entorno de simulación en SUMO para la conducción autónoma.
2. **Implementar** algoritmos de DRL que controlen un vehículo individual, pero interactuando con múltiples vehículos y rutas variables.
3. **Evaluar** la robustez del agente ante distintas densidades de tráfico y condiciones de la red vial.

Así, se espera contribuir a la línea de investigación sobre conducción autónoma en escenarios urbanos complejos, ofreciendo una base sólida para futuros trabajos que deseen optimizar el tráfico de manera global.

3. Diseño e implementación del trabajo

En este capítulo se detalla el proceso de diseño e implementación del entorno de simulación para la conducción autónoma basado en SUMO y la integración de algoritmos de Aprendizaje por Refuerzo Profundo (DRL). El objetivo es desarrollar un sistema que permita entrenar a un agente de RL para que aprenda a conducir de forma segura y eficiente en un entorno urbano realista, gestionando tanto maniobras de conducción individuales como decisiones a nivel de enrutamiento. Se describirá la arquitectura del sistema, los componentes clave (como la interfaz con SUMO mediante TraCI, la función de recompensa, y el mecanismo de reasignación dinámica de rutas) y la metodología seguida para la implementación y validación del entorno. Este diseño constituye la base para experimentar con distintos algoritmos de DRL y evaluar su rendimiento en escenarios complejos.

3.1. Instalación del entorno SUMO

SUMO es un simulador de tráfico de código abierto que sigue una arquitectura Cliente-Servidor. Para instalar el entorno SUMO se debe descargar la versión correspondiente desde el repositorio oficial, en este caso, la 1.20.0, lo que garantiza contar con una versión estable y probada. En este proyecto se ha optado por instalar SUMO en la máquina local, de modo que tanto el servidor como el cliente se ejecuten en el mismo equipo, facilitando la integración mediante la API **TraCI**.

La instalación se realiza de la siguiente manera:

1. **Descarga e instalación del servidor:** Se accede a la página oficial de SUMO [16] y se descarga el instalador correspondiente a la versión recomendada. Una vez descargado, se ejecuta el instalador y se sigue el proceso habitual de instalación en el sistema operativo.

2. **Configuración del entorno:** Una vez instalado el servidor, se verifica que SUMO se ejecuta correctamente abriendo SUMO-GUI. En este proyecto se utilizará el cliente en Python, por lo que se recomienda crear un entorno virtual (por ejemplo, con `venv` o `conda`) utilizando Python 3.12, que garantice la compatibilidad con las funciones de la API TraCI.

3. **Integración de TraCI:** La API TraCI, que permite la comunicación en tiempo real entre el simulador y el código Python, se encuentra incluida en el paquete de herramientas de SUMO. Se debe añadir la ruta al directorio `tools` de SUMO en la variable de entorno `PYTHONPATH` o incluirla directamente en el script de Python. De este modo, se podrá importar el módulo `traci` y establecer la conexión entre el cliente y el servidor SUMO.

4. **Verificación de la instalación:** Se recomienda ejecutar un script sencillo que inicie SUMO en modo GUI y realice algunos pasos de simulación para confirmar que la comunicación a través de TraCI funciona correctamente. Las especificaciones hardware del sistema utilizado en este proyecto (procesador *Apple M2* y 16GB de RAM) cumplen con los requisitos mínimos para ejecutar SUMO de forma fluida [16].

Estas características, junto con la amplia documentación y comunidad de usuarios de SUMO, garantizan que el simulador pueda integrarse eficazmente en el proyecto, permitiendo una simulación de tráfico a gran escala y el control personalizado del vehículo a través de algoritmos de aprendizaje por refuerzo.

3.2. Arquitectura del sistema

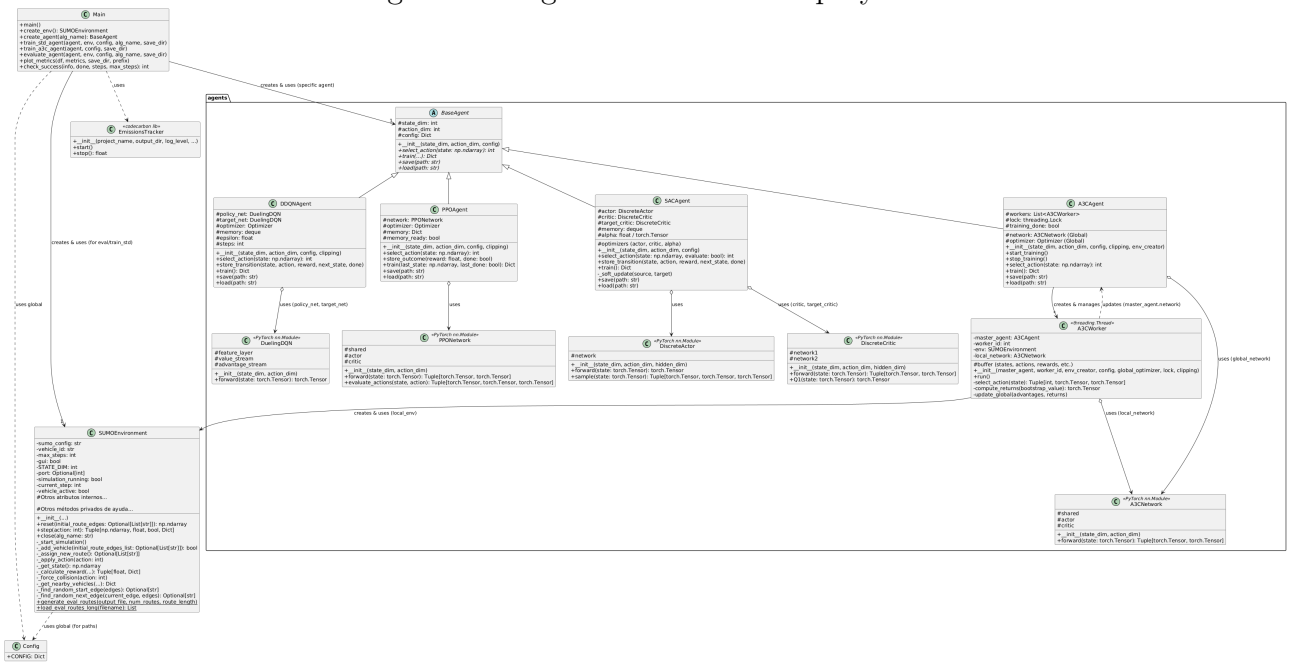
El sistema se compone de los siguientes módulos principales:

- **Entorno SUMO (sumo_environment.py)**: Interfaz creada para manejar la interacción entre el agente de RL y el simulador SUMO. Responsable de:
  1. Iniciar y controlar la simulación SUMO.
  2. Añadir y gestionar el vehículo controlado por el agente.
  3. Obtener el estado del entorno (información del vehículo, entorno, etc.).
  4. Aplicar las acciones del agente al vehículo.
  5. Calcular la recompensa.
  6. Gestionar el fin de los episodios.
  7. Visualización (delegada a SUMO-GUI).
- **Agente de RL (<algorithm>_agent.py)**: Implementa el algoritmo de aprendizaje por refuerzo profundo correspondiente (DDQN, PPO, A3C y SAC). Responsable de:
  1. Recibir el estado del entorno.
  2. Seleccionar una acción basada en su política.
  3. Almacenar experiencias (transiciones).
  4. Aprender de las experiencias (actualizar su política y/o función de valor).
- **Script Principal (main.py)**: Orquesta el proceso de entrenamiento y/o evaluación. Responsable de:
  1. Crear instancias del entorno y del agente.

2. Ejecutar el bucle principal de entrenamiento (episodios).
3. Guardar/cargar el modelo del agente.
4. Visualizar el progreso del entrenamiento.

3.2.1. Diagrama de clases

Figura 3: Diagrama de clases del proyecto



La Figura 3 ilustra la arquitectura modular del proyecto, destacando las clases principales y sus interacciones. El fichero `main.py` actúa como un orquestador central, encargado de crear instancias y gestionar el flujo de entrenamiento y evaluación de los agentes. Este interactúa con una instancia de `SUMOEnvironment`, que encapsula toda la lógica de comunicación con el simulador SUMO (vía TraCI), la gestión del vehículo, la extracción del estado del entorno y el cálculo de recompensas. `SUMOEnvironment` también incluye métodos estáticos para generar y cargar rutas de evaluación estandarizadas. El fichero `config.py` proporciona acceso global a los hiperparámetros y configuraciones del proyecto. Para el seguimiento del impacto ambiental, `main` utiliza la librería externa `EmissionsTracker` de CodeCarbon.

Los diferentes algoritmos de Aprendizaje por Refuerzo (`DDQNAgent`, `PPOAgent`, `SACAgent`, `A3CAgent`) heredan de una clase abstracta `BaseAgent`, que define la interfaz común para seleccionar acciones, entrenar y guardar/cargar modelos. Cada agente específico implementa su propia lógica interna y utiliza sus correspondientes redes neuronales (ej., `DuelingDQN`, `PPONetwork`, `DiscreteActor`/`DiscreteCritic`, `A3CNetwork`), todas ellas basadas en `torch.nn.Module`. La

arquitectura de `A3CAgent` es particular, ya que gestiona múltiples `A3CWorker` (hilos) que interactúan de forma independiente con sus propias instancias de `SUMOEnvironment` y actualizan una red neuronal global compartida.

3.3. Dueling Deep Q-Network (DDQN)

Para comprender a fondo DDQN es fundamental analizar los algoritmos base en los que se sustenta, principalmente los métodos basados en Q-learning. A continuación se detallan los algoritmos clave:

3.3.1. Q-Network

En primer lugar cabe aclarar que en el aprendizaje por refuerzo existen dos tipos de algoritmos:

- **off-policy**: Un algoritmo *off-policy* es aquel que aprende una política óptima diferente de la política utilizada para generar los datos. Es decir, la política que se está evaluando y mejorando (política objetivo) no es necesariamente la misma que la que se usa para explorar el entorno (política de comportamiento). Esto permite reutilizar experiencias pasadas y explorar con políticas distintas a la óptima, lo cual puede mejorar la eficiencia del aprendizaje.
- **on-policy**: Un algoritmo *on-policy* es aquel que actualiza su política en función de la misma política con la que interactúan en el entorno.

Q-learning es un algoritmo de aprendizaje por refuerzo **off-policy** que busca aprender la **función de valor-acción** $Q(s, a)$. Esta función representa la **recompensa acumulada esperada** al tomar la acción a en el estado s y seguir la política óptima a partir de ese momento. La actualización se realiza utilizando la **ecuación de Bellman** [19]:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{1}$$

donde α es la **tasa de aprendizaje** y γ el **factor de descuento**. Debido a que Q-learning almacena los Q-valores en una tabla, su uso se vuelve impracticable en problemas con espacios de estados muy grandes o continuos.

3.3.2. Deep Q-Network (DQN)

DQN extiende el Q-learning utilizando redes neuronales profundas para aproximar la función $Q(s, a)$. En lugar de usar una tabla, se entrena una red que, dada la representación del estado, produce los Q-valores para cada acción posible. Entre sus técnicas clave se encuentran:

- **Experience Replay**: Se almacena un historial de experiencias (s, a, r, s') en un buffer y se entrena la red con muestras aleatorias, lo que rompe la correlación entre experiencias consecutivas y estabiliza el aprendizaje.
- **Red Objetivo (Target Network)**: Se utiliza una copia de la red que se actualiza periódicamente para calcular los valores objetivo, reduciendo la inestabilidad de las actualizaciones.

La **función de pérdida** se define como el error cuadrático medio entre el Q-valor estimado y el objetivo obtenido mediante la ecuación de Bellman. Función de pérdida [20]:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(y - Q(s, a; \theta))^2\right] \tag{2}$$

Donde $\mathbb{E}_{(s,a,r,s')\sim D}$ es el **valor esperado** para una muestra aleatoria de las experiencias almacenadas en el **buffer de experiencias** D, $Q(s, a; \theta)$ es el **valor Q estimado por la red neuronal**, $Q(s', a'; \theta^-)$ es el **valor Q estimado por la red objetivo**, γ es el **factor de descuento**, y es el **valor objetivo para la actualización de la red**, calculado como la **recompensa inmediata** r más el **valor descontado de las futuras recompensas** $(\gamma \max_{a'} Q(s', a'; \theta^-))$, y θ y θ^- son los **parámetros de la red neuronal actual y la red objetivo**, respectivamente.

3.3.3. Dueling Deep Q-Network (DDQN)

En el enfoque clásico de DQN, se utiliza una única red neuronal para aproximar la función de valor–acción $Q(s, a)$, la cual estima la recompensa acumulada esperada de ejecutar la acción a en el estado s y seguir la política actual. Sin embargo, en muchos entornos resulta **redundante** evaluar de forma detallada la contribución de cada acción cuando, en muchos estados, el valor intrínseco del estado domina la dinámica de recompensa. El enfoque Dueling DQN propone separar este problema en dos componentes:

- **Valor del Estado** $V(s)$: Estima la bondad intrínseca del estado sin considerar las acciones.
- **Ventaja** $A(s, a)$: Mide la contribución adicional de cada acción respecto al valor promedio del estado.

La salida de la red se combina mediante la fórmula:

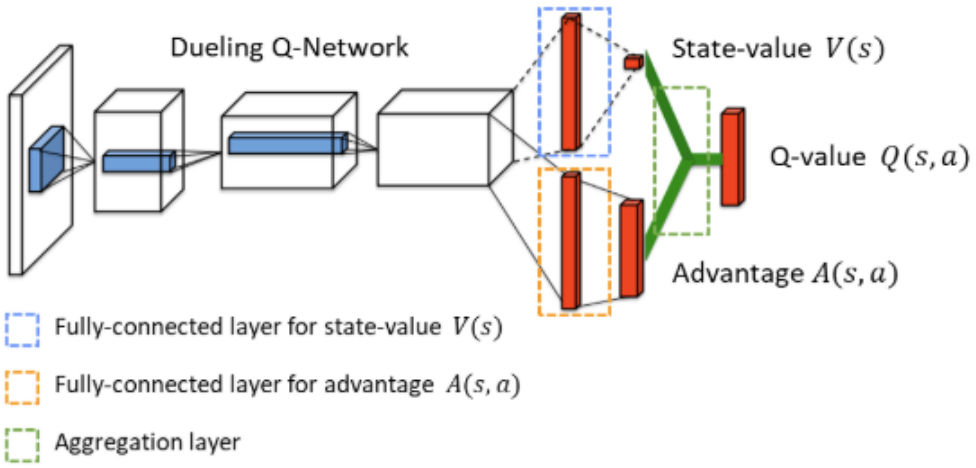
$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|A|}\sum_{a'} A(s, a')\right) \tag{3}$$

Esta separación permite que la red se concentre en aprender con mayor precisión qué estados son valiosos, lo que puede ser fundamental en entornos donde la elección de la acción tiene un impacto marginal en el retorno a largo plazo.

La red neuronal de Dueling DQN se compone de un **bloque de extracción de características compartido** (por ejemplo, capas convolucionales en entornos visuales) que luego se divide en dos "ramas" independientes (Figura 4):

- **Stream de Valor**: Esta rama termina en una neurona única que estima $V(s;\theta,\beta)$.
- **Stream de Ventaja**: Esta rama calcula un vector $A(s,a;\theta,\alpha)$ para cada acción a.

Figura 4: Arquitectura de DDQN.



La combinación de ambas ramas se realiza mediante una **función de agregación** diseñada para asegurar la **identificabilidad**, es decir, que la descomposición en V y A sea única. Una formulación común es:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|} \sum_{a'} A(s,a';\theta,\alpha) \right) \tag{4}$$

Donde $Q(s,a;\theta,\alpha,\beta)$ es la **función de acción-valor Q** que estima el valor de tomar una acción a en el estado s, θ son los **parámetros** de la red neuronal que se encargan de **modelar** tanto la función de valor de estado como la de ventaja, α son los **parámetros** de la red neuronal asociados con la **función de ventaja**, β son los **parámetros** de la red neuronal asociados con la **función de valor del estado**, $V(s;\theta,\beta)$ es la **función de valor del estado** $V(s)$, que estima la calidad de un estado s sin considerar una acción específica, $A(s,a;\theta,\alpha)$ es la **función de ventaja** $A(s,a)$, que representa la diferencia entre el valor de la acción a y el valor promedio de todas las posibles acciones a' en el estado s y $\frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)$ es el **valor promedio**

de las ventajas de todas las acciones posibles a' en el estado s, donde $|A|$ es el número total de acciones posibles. Este término ayuda a **centrar la función de ventaja alrededor de 0**, evitando que el valor de $A(s, a)$ sea arbitrario debido a la escala.

La sustracción del promedio de las ventajas garantiza que la estimación de $V(s)$ represente de forma aislada el valor basal del estado, mientras que $A(s, a)$ refleja únicamente el efecto diferencial de cada acción [21].

3.3.4. Proceso de entrenamiento

El entrenamiento de Dueling DQN sigue, en esencia, la metodología de Deep Q-Learning, haciendo uso de técnicas como:

- **Experience Replay**: Se almacena la experiencia en forma de tuplas (s, a, r, s') para romper la correlación temporal y estabilizar la actualización de los pesos.
- **Red Objetivo (Target Network)**: Se utiliza una red objetivo cuyas actualizaciones son periódicas para proporcionar un objetivo de aprendizaje más estable y reducir la oscilación en las estimaciones de Q.
- **Minimización de la Función de Pérdida**: La función de pérdida se define como el error cuadrático medio entre el valor $Q(s, a)$ predicho y el objetivo de Bellman, es decir:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{5}$$

La estructura dueling se integra en el cómputo de $Q(s, a)$ como se describió anteriormente, permitiendo que el gradiente se propague de manera diferenciada en las ramas de V y A.

3.3.5. Pseudocódigo

El algoritmo 1 muestra el pseudocódigo de DDQN.

3.3.6. Ventajas y consideraciones

Entre las ventajas fundamentales de la arquitectura Dueling DQN destacan:

- **Estabilidad en la Estimación del Valor**: Al separar $V(s)$ de $A(s, a)$, el algoritmo puede aprender de manera más robusta qué estados son intrínsecamente buenos, independientemente de las acciones disponibles.

Algorithm 1 Dueling Deep Q-Network (DDQN)

Entrada: γ (descuento), α (tasa de aprendizaje), τ (actualización de la red objetivo), buffer_size (tamaño del buffer de experiencias)

Inicializar red $Q(\theta)$ y red objetivo $Q'(\theta' = \theta)$
Inicializar buffer de experiencias D
for cada episodio **do**
    $estado \leftarrow entorno.reset()$
    for cada paso t **do**
        $accion \leftarrow \varepsilon$-greedy$(Q(estado), \varepsilon)$
        $nuevo\_estado, recompensa, done \leftarrow entorno.step(accion)$
        Almacenar $(estado, accion, recompensa, nuevo\_estado, done)$ en D
        ▷ Muestrear mini-batch de D
        $batch \leftarrow sample(D, batch\_size)$
        ▷ Calcular Q-target usando red objetivo y descomposición Dueling
        $Q\_target \leftarrow recompensa + \gamma \cdot \text{máx}(Q'(nuevo\_estado)) \cdot (1 - done)$
        ▷ Calcular pérdida y actualizar θ
        $perdida \leftarrow MSE(Q(estado)[accion], Q\_target)$
        descenso_gradiente(θ, $perdida$)
        ▷ Actualizar red objetivo
        $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
        $estado \leftarrow nuevo\_estado$
        if done **then**
            break
        end if
    end for
end for

- **Mejora en la Eficiencia del Aprendizaje**: En situaciones en las que la elección de acción tiene un impacto marginal, la red puede concentrar recursos en estimar con precisión el valor del estado, reduciendo la varianza en la actualización de Q.

- **Identificabilidad**: La estrategia de sustracción (ya sea el promedio o, en algunas variantes, el máximo) permite superar el problema de no poder distinguir entre V y A únicamente a partir de Q.

Es importante notar que, en algunas implementaciones, la técnica Dueling se combina con la estrategia de Double DQN para mitigar la sobreestimación de Q, obteniéndose variantes conocidas como Dueling Double DQN o D3QN. Esta combinación utiliza la red actual para seleccionar la acción óptima y la red objetivo para evaluarla, lo que reduce el sesgo en la estimación [22].

3.4. Asynchronous Advantage Actor Critic (A3C)

Otro de los algoritmos a implementar y comparar es **A3C**. Para comprender a fondo A3C es fundamental analizar los algoritmos base en los que se sustenta, principalmente los métodos de gradiente de política y las arquitecturas actor-crítico. A continuación se detallan los algoritmos clave:

3.4.1. REINFORCE (Policy Gradient Monte Carlo)

REINFORCE es uno de los algoritmos pioneros en el campo de los **métodos de gradiente de política**, propuesto por Williams en 1992. Sus características principales son:

- **Basado en Monte Carlo**: Utiliza episodios completos para estimar el retorno acumulado $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ desde un estado s_t y así calcular el gradiente de la política.

- **Actualización directa de la política**: Se actualiza la política mediante la regla $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi(a_t \mid s_t; \theta) G_t$, donde α es la **tasa de aprendizaje**.

- **Alta varianza**: Debido a que utiliza retornos completos de episodios, las estimaciones pueden ser ruidosas y poco eficientes, lo que dificulta la convergencia en entornos complejos.

Este algoritmo sienta las bases conceptuales para el aprendizaje basado en gradientes de política, pero su alta varianza llevó al desarrollo de métodos que integraran estimadores de valor para estabilizar el aprendizaje [23].

3.4.2. Métodos Actor-Crítico

Los métodos actor-crítico combinan dos componentes fundamentales:

- **Actor**: Responsable de parametrizar la **política** $\pi(a \mid s; \theta)$ y determinar qué acción tomar en cada estado.
- **Crítico**: Encargado de estimar la **función de valor** $V(s; \theta_v)$ (o, en algunas variantes, la **función acción-valor** $Q(s, a)$) que sirve como referencia para **evaluar** las decisiones del actor.

Principales ventajas:

- **Reducción de varianza**: Al utilizar un estimador de valor (el crítico) para aproximar el retorno esperado, se **reduce la varianza** en la actualización del gradiente en comparación con REINFORCE.
- **Aprendizaje basado en TD (Temporal Difference)**: El crítico se entrena mediante **métodos de diferencia temporal**, lo que permite realizar actualizaciones de forma **incremental** sin necesidad de esperar a la finalización completa del episodio.

La interacción entre actor y crítico se traduce en una actualización que pondera el gradiente de la política con el error de valoración, de forma similar a:

$$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi(a_t \mid s_t; \theta) \delta_t, \tag{6}$$

donde $\delta_t = r_t + \gamma V(s_{t+1}; \theta_v) - V(s_t; \theta_v)$ es el **error de TD**.

Este marco permite que el actor se oriente a tomar acciones que produzcan retornos mayores, mientras que el crítico mejora progresivamente su estimación del valor de los estados [24].

3.4.3. Advantage Actor-Critic (A2C)

El **Advantage Actor-Critic (A2C)** es una evolución natural de los métodos actor-crítico que introduce el concepto de **ventaja** para mejorar la **eficiencia del aprendizaje**:

La ventaja se calcula como:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t), \tag{7}$$

o de forma práctica mediante una **aproximación n-pasos**:

$$A(s_t, a_t) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t). \tag{8}$$

Este término mide lo "mejor" que es una acción en comparación con el promedio esperado en el estado.

La **política** se actualiza utilizando el **gradiente ponderado por la ventaja**, lo que ayuda a **centrar el aprendizaje** en aquellas acciones que superan la **expectativa**:

$$\nabla_\theta \log \pi(a_t \mid s_t; \theta) A(s_t, a_t). \tag{9}$$

En A2C la actualización se realiza de manera **sincrónica**, es decir, se recopilan las experiencias de múltiples muestras o mini-batches y se actualizan los parámetros de forma coordinada.

La introducción del estimador de ventaja ayuda a **reducir la varianza** sin incurrir en un **sesgo** excesivo, lo cual es esencial para entornos con alta complejidad o donde los retornos pueden variar considerablemente [25].

3.4.4. Asynchronous Advantage Actor Critic (A3C)

El algoritmo **Asynchronous Advantage Actor-Critic (A3C)** se puede ver como una extensión de A2C, donde se abordan algunas limitaciones prácticas:

- **Asincronía**: En lugar de sincronizar todas las actualizaciones (como en A2C), A3C utiliza múltiples workers que interactúan de forma independiente con sus propias copias del entorno. Cada *worker* actualiza de forma asíncrona una red global, lo que acelera la **convergencia** y reduce la **correlación** entre muestras.

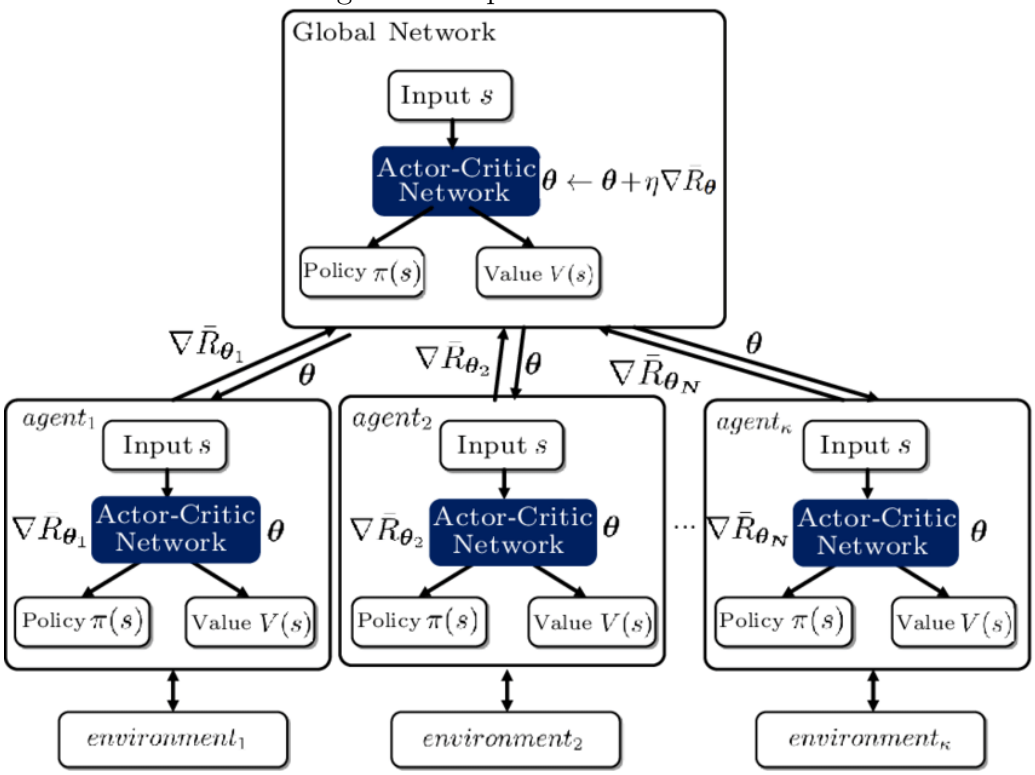
- **Exploración diversificada**: Gracias a la ejecución paralela en diferentes entornos o instancias, el algoritmo explora de manera más amplia el **espacio de estados**, lo que mejora la **generalización** y la **robustez** del aprendizaje.

- **Buen aprovechamiento de arquitecturas paralelas**: La arquitectura asíncrona permite aprovechar de forma más eficiente la capacidad de múltiples núcleos de CPU o combinaciones CPU/GPU, lo que resulta en una aceleración sustancial del proceso de **entrenamiento**.

A3C es un algoritmo de aprendizaje por refuerzo profundo que representa un avance significativo en la combinación de **estrategias de política (actor)** y **evaluación de estados (crítico)** con la ventaja adicional de la paralelización asíncrona. Propuesto inicialmente por DeepMind en 2016, A3C surge para mitigar limitaciones inherentes a métodos anteriores, como la **correlación en las muestras** y la **dependencia de replay buffers** en métodos basados en Q-learning. La idea central es **explotar múltiples agentes** (o "workers") que exploran de

forma independiente el entorno, acumulando gradientes que se sincronizan de manera asíncrona con una red global (Figura 5). Esto no solo **acelera el aprendizaje**, también **mejora la robustez** al diversificar las experiencias recogidas en entornos potencialmente heterogéneos.

Figura 5: Arquitectura de A3C.



A3C se enmarca dentro de los métodos actor-crítico, donde:

- **Actor**: Es responsable de tomar decisiones. Define una **política estocástica** $\pi(a \mid s; \theta)$ que mapea estados s a distribuciones sobre acciones a.
- **Crítico**: Estima la **función de valor** $V(s; \theta_v)$ para evaluar la calidad del estado actual y servir de base para calcular la ventaja.

La ventaja se define como:

$$A(s_t, a_t; \theta, \theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v) \tag{10}$$

Donde $A(s_t, a_t; \theta, \theta_v)$ es la **función de ventaja**, que mide la diferencia entre el valor real de tomar una acción a_t en un estado s_t y el valor esperado de ese estado y θ y θ_v son los **parámetros de las redes neuronales actor y crítica**, respectivamente. γ es el **factor de descuento**, que controla el peso que tiene el futuro en la toma de decisiones, un valor cercano a

1 indica que se valoran mucho las recompensas futuras, mientras que un valor cercano a **0** indica que solo se valoran las recompensas inmediatas. r_{t+i} es la **recompensa recibida** en el paso de tiempo $t+i$, por lo que $\sum_{i=0}^{k-1} \gamma^i r_{t+i}$ calcula la **recompensa acumulada** durante los siguientes k pasos, descontada por el factor γ. $V(s_{t+k}; \theta_v)$ es el **valor del estado** s_{t+k} predicho por la **red crítica**, es decir, una estimación de la recompensa futura que se puede esperar a partir de ese estado y $V(s_t; \theta_v$ es el **valor estimado del estado** s_t según la **red crítica**, representa lo que el agente espera recibir en términos de recompensa desde ese estado, sin importar la acción tomada.

El concepto de **ventaja** permite **reducir la varianza en la estimación de gradientes** al comparar el retorno observado con una estimación de valor de referencia, lo que guía de forma más precisa la actualización de la política. Este término ayuda a identificar qué tan buena fue una acción respecto a la expectativa general en ese estado [26].

3.4.5. Proceso de entrenamiento

Una de las innovaciones más relevantes es el uso de múltiples agentes que interactúan en paralelo con copias independientes del entorno. Cada agente (worker) ejecuta su propio proceso y mantiene una versión local de los parámetros θ' y θ'_v, sincronizándose periódicamente con los parámetros globales θ y θ_v. Este diseño asíncrono permite:

- **Descorrelación de muestras**: Al operar en diferentes entornos o diferentes partes del mismo entorno, se reduce la correlación entre las muestras de experiencia.
- **Mejor utilización de recursos computacionales**: Se aprovechan múltiples núcleos de CPU (e incluso combinaciones híbridas CPU/GPU) para acelerar el proceso de entrenamiento.

El proceso de cada worker se resume en los siguientes pasos:

1. Inicializar parámetros locales con los globales.
2. Recoger una secuencia de transiciones (hasta alcanzar $t_{\max}$ o un estado terminal).
3. Calcular el retorno acumulado R utilizando un esquema n-pasos.
4. Acumular gradientes para la política y para el crítico:
   - Para el actor: $\nabla_{\theta'} \log \pi(a_t \mid s_t; \theta') \, (R - V(s_t; \theta'_v))$
   - Para el crítico: Gradiente del error cuadrático $\nabla_{\theta'_v} \, (R - V(s_t; \theta'_v))^2$
5. Actualizar de forma asíncrona los parámetros globales con los gradientes acumulados.

Este enfoque se asemeja a una versión paralelizada del descenso de gradiente estocástico, donde cada worker contribuye de forma independiente a la optimización global.

3.4.6. Pseudocódigo

El algoritmo 2 muestra el pseudocódigo de A3C.

Algorithm 2 Asynchronous Advantage Actor-Critic (A3C)

Entrada: γ (descuento), α_{actor}, α_{critic} (tasas de aprendizaje), t_{max} (pasos por worker)

Inicializar red global Actor (θ) y Crítico (θ_v)
Crear N workers (hilos paralelos)
for cada worker en paralelo **do**
    while no converja **do**
        $estado \leftarrow entorno.reset()$
        buffer_estados, buffer_acciones, buffer_recompensas $\leftarrow$ [], [], []
        for $t = 1$ hasta t_{max} **do**
            $accion \sim \pi(accion|estado; \theta)$
            $nuevo\_estado, recompensa, done \leftarrow entorno.step(accion)$
            Almacenar $(estado, accion, recompensa)$
            $estado \leftarrow nuevo\_estado$
            if done **then**
                break
            end if
        end for
        ▷ Calcular ventajas (n-pasos) y valores objetivo
        $R \leftarrow 0$ si *done* sino $V(nuevo\_estado; \theta_v)$
        buffer_ventajas $\leftarrow$ []
        for $i = t - 1$ hasta 0 **do**
            $R \leftarrow$ buffer_recompensas[i] $+\gamma \cdot R$
            $ventaja \leftarrow R - V($buffer_estados[i]$; \theta_v)$
            Insertar $ventaja$ en buffer_ventajas al inicio
        end for
        ▷ Actualizar red global (Actor y Crítico)
        $grad_{actor} \leftarrow \nabla_\theta \log \pi(acciones|estados) \cdot ventajas$
        $grad_{critic} \leftarrow \nabla_{\theta_v}(V(estados) - R)^2$
        actualizar_red_global$(\theta, \theta_v, grad_{actor}, grad_{critic})$
    end while
end for

3.4.7. Ventajas y consideraciones

El algoritmo Asynchronous Advantage Actor-Critic (A3C) presenta diversas ventajas y consideraciones que impactan su desempeño y aplicabilidad en problemas de aprendizaje por refuerzo.

Ventajas de A3C:

- **Paralelismo asíncrono:** A3C emplea múltiples agentes que operan en paralelo y de manera asíncrona, lo que permite una exploración más amplia y diversa del espacio de estados. Este enfoque reduce la correlación entre las muestras y mejora la eficiencia del aprendizaje.
- **Reducción de la varianza en las estimaciones:** Al integrar las funciones de actor y crítico, A3C utiliza la estimación de ventaja para guiar las actualizaciones de la política, lo que disminuye la varianza en las estimaciones y conduce a un aprendizaje más estable.
- **Buen aprovechamiento de arquitecturas paralelas:** La implementación asíncrona permite que A3C aproveche de manera más efectiva los recursos computacionales, especialmente en arquitecturas con múltiples núcleos de procesamiento, acelerando el proceso de aprendizaje.

Consideraciones y Desventajas:

- **Complejidad en la implementación:** La coordinación entre múltiples agentes y la gestión de sus actualizaciones asíncronas pueden complicar la implementación y el mantenimiento del sistema.
- **Consumo de recursos:** El funcionamiento simultáneo de múltiples agentes puede requerir una cantidad significativa de recursos computacionales, lo que podría ser un desafío en entornos con recursos limitados.
- **Estabilidad en entornos no estacionarios:** En entornos donde las dinámicas cambian con el tiempo, la política aprendida puede volverse inestable debido a la naturaleza asíncrona de las actualizaciones.

En resumen, A3C ofrece mejoras significativas en términos de eficiencia y estabilidad en el aprendizaje por refuerzo al combinar técnicas de actor-crítico con paralelismo asíncrono. Sin embargo, es crucial evaluar las necesidades específicas del problema y los recursos disponibles antes de su implementación.

3.5. Proximal Policy Optimization (PPO)

El **Proximal Policy Optimization (PPO)** es un algoritmo de aprendizaje por refuerzo que surge como respuesta a las limitaciones de métodos anteriores basados en **gradiente de política**, como el **REINFORCE** y, en particular, el **Trust Region Policy Optimization (TRPO, Subsubsección 3.5.1)** [27]. La necesidad de equilibrar la **estabilidad** de las actualizaciones de política con la **simplicidad computacional** llevó a la formulación de PPO,

propuesto por Schulman et al. en 2017 [28]. Mientras que TRPO introducía restricciones fuertes (mediante una **divergencia Kullback-Leibler**) para garantizar mejoras monótonas, su implementación resultaba compleja y costosa en términos computacionales. PPO simplifica este enfoque mediante mecanismos de **clipping** que, de forma implícita, limitan la **magnitud** de las actualizaciones.

3.5.1. Trust Region Policy Optimization (TRPO)

El algoritmo **Trust Region Policy Optimization (TRPO)** introdujo la idea de **limitar el cambio de la política** entre iteraciones, garantizando que la nueva política no se alejara demasiado de la anterior. Esto se formaliza al imponer una restricción basada en la **divergencia de Kullback-Leibler (KL)**:

$$\max_{\theta} \mathbb{E}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A(s,a)\right] \quad \text{sujeto a} \quad D_{\text{KL}}(\pi_{\theta_{\text{old}}} \parallel \pi_\theta) \leq \delta, \tag{11}$$

donde $A(s,a)$ es la **función de ventaja** y δ un parámetro que controla la "**proximidad**" entre políticas.

Aunque este enfoque ofrece garantías teóricas de **mejora monótona**, su resolución implica un problema de **optimización cuadrática** restringido, aumentando la **complejidad** de implementación y la **carga computacional** [27].

3.5.2. Proximal Policy Optimization (PPO)

El algoritmo **Proximal Policy Optimization (PPO)** se fundamenta en la utilización de un objetivo **surrogate** que compara la **nueva política** π_θ con la **política anterior** $\pi_{\theta_{\text{old}}}$ mediante el **ratio de importancia** [28]:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{12}$$

Este ratio mide **cuánto ha cambiado la probabilidad de seleccionar la acción** a_t en el estado s_t tras la actualización.

La innovación clave de PPO es la introducción de un mecanismo de *clipping* en la **función de pérdida**, definida como:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{13}$$

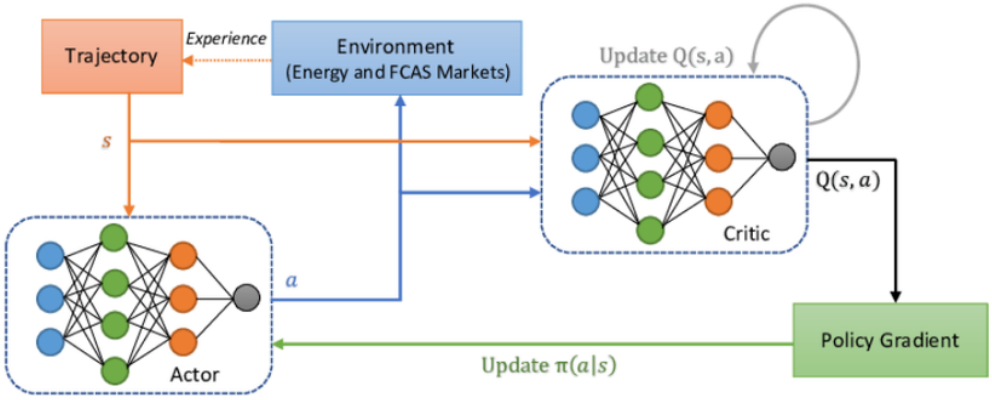
Donde $\hat{A}_t$ es el **estimador de la ventaja** y ϵ es un **hiperparámetro** pequeño (típicamente entre 0.1 y 0.2).

Este objetivo actúa de la siguiente forma:

- Si $r_t(\theta)$ se mantiene dentro del intervalo $[1-\epsilon, 1+\epsilon]$, la actualización se realiza de manera **libre**, maximizando directamente $r_t(\theta)\hat{A}_t$.
- Si $r_t(\theta)$ excede este rango, el **clipping** evita que la contribución de esa muestra empuje la actualización de forma demasiado agresiva. Esto **limita el riesgo de grandes desviaciones en la política** y promueve actualizaciones más **conservadoras y estables**.

La Figura 6 muestra la arquitectura del algoritmo PPO.

Figura 6: Arquitectura de PPO.



3.5.3. Proceso de entrenamiento

El proceso de entrenamiento de PPO se basa en dos fases principales que se repiten de forma iterativa:

1. **Recolección de Datos (Rollout)**:
   - El agente interactúa con el entorno usando su política actual para recolectar trayectorias (secuencias de estados, acciones, recompensas y, a veces, información adicional como valores estimados).
   - Estas trayectorias se pueden recopilar en **paralelo** mediante entornos **vectorizados**, lo que aumenta la **eficiencia** al obtener muestras de varios episodios simultáneamente.
2. **Actualización de la Política y del Valor**:
   - **Cálculo de ventajas y retornos**: Se utiliza una técnica como la **Estimación de Ventaja Generalizada (GAE)** para calcular, a partir de las recompensas y las **predicciones del valor**, tanto el **retorno** (suma descontada de recompensas) como la **ventaja** (la diferencia entre el retorno observado y el valor estimado).

- **Optimización del objetivo PPO**: Se define una función objetivo **surrogate** que compara la probabilidad de haber tomado una acción bajo la nueva política frente a la antigua. Mediante un mecanismo de **clipping** se evita que la actualización se aleje demasiado de la política previa.

  La actualización se realiza con varios pasos de descenso de gradiente (usualmente mini-batches y múltiples épocas) sobre este objetivo, lo que permite aprovechar mejor los datos recolectados.

- **Actualización del crítico**: Se entrena la red de valor **minimizando la diferencia** (por ejemplo, el error cuadrático medio) entre la estimación del valor y el retorno calculado, lo que ayuda a que la ventaja se calcule con **mayor precisión** en futuras iteraciones.

En resumen, PPO alterna entre recopilar experiencia en el entorno y actualizar tanto la política (usando la función de pérdida recortada) como el valor (mediante regresión sobre los retornos), lo que garantiza actualizaciones estables y eficientes en la búsqueda de la política óptima.

3.5.4. Pseudocódigo

El algoritmo 3 muestra el pseudocódigo de PPO.

3.5.5. Ventajas y consideraciones

La creación de PPO respondió a varias necesidades críticas en la investigación en aprendizaje por refuerzo:

- **Estabilidad en las actualizaciones**: Se buscaba evitar cambios bruscos en la política que pudieran llevar a comportamientos inestables o a la pérdida de la convergencia, sin recurrir a costosas optimizaciones.

- **Simplicidad y facilidad de implementación**: Al reemplazar la restricción explícita (como la divergencia KL en TRPO) por un método de clipping, PPO permite una implementación más directa y menos dependiente de soluciones numéricas complejas.

- **Eficiencia en el uso de datos**: Al garantizar que las actualizaciones sean moderadas, se logra una mayor eficiencia en el aprovechamiento de las muestras recolectadas, mejorando la tasa de convergencia en entornos complejos.

Algorithm 3 Proximal Policy Optimization (PPO)

Entrada: γ (descuento), ϵ (clip_range), K (épocas de actualización)

Inicializar política $\pi(\theta)$ y Crítico $V(\theta_v)$
Inicializar buffer de trayectorias D
for cada iteración **do**
    $estado \leftarrow entorno.reset()$
    while no se llene D **do** ▷ Fase de recolección de datos
        $accion \sim \pi(accion|estado;\theta)$
        $nuevo\_estado, recompensa, done \leftarrow entorno.step(accion)$
        Almacenar $(estado, accion, recompensa, V(estado))$ en D
        $estado \leftarrow nuevo\_estado$
    end while
    ▷ Calcular ventajas y retornos (GAE)
    calcular_ventajas(D, γ, λ)
    for $epoca = 1$ hasta K **do** ▷ Fase de optimización
        for cada $batch$ en D **do**
            $ratio \leftarrow \frac{\pi(accion|estado;\theta)}{\pi(accion|estado;\theta_{old})}$
            $ventaja\_normalizada \leftarrow \frac{ventaja - \mu_{ventaja}}{\sigma_{ventaja}}$
            ▷ Pérdida del Actor (con clipping)
            $perdida_{actor} \leftarrow mean(-\min(ratio \cdot ventaja\_normalizada, \text{clip}(ratio, 1-\epsilon, 1+\epsilon) \cdot ventaja\_normalizada))$
            ▷ Pérdida del Crítico
            $perdida_{critic} \leftarrow MSE(V(estado), retorno)$
            ▷ Actualizar θ y θ_v
            descenso_gradiente$(\theta, \theta_v, perdida_{actor} + perdida_{critic})$
        end for
    end for
end for

- **Robustez en aplicaciones reales**: La estabilidad y simplicidad de PPO lo han convertido en uno de los algoritmos preferidos para tareas de control, robótica y juegos, donde la eficiencia y la robustez son cruciales.

Además del enfoque basado en clipping, existen variantes de PPO que incorporan directamente una penalización por divergencia KL en la función objetivo. Aunque ambas variantes persiguen la meta de mantener actualizaciones "proximales", la versión con clipping es la más utilizada por su simplicidad y eficacia empírica.

3.6. Soft Actor-Critic (SAC)

El algoritmo **Soft Actor-Critic (SAC)** es un método de aprendizaje por refuerzo que se inscribe dentro del paradigma de la **maximización de entropía**, combinando ideas de los enfoques **actor-crítico** y métodos **off-policy**.

A diferencia de los métodos clásicos que únicamente maximizan la recompensa acumulada, SAC incorpora un término de **entropía** en el objetivo. La **función de recompensa** modificada es:

$$J(\pi) = \mathbb{E}\left[\sum_t \left(r(s_t, a_t) + \alpha \cdot H(\pi(\cdot|s_t))\right)\right] \tag{14}$$

Donde $H(\pi(\cdot|s))$ representa la **entropía de la política** en el estado s y α es un parámetro que regula la importancia de la entropía.

Este enfoque favorece **políticas estocásticas**, promoviendo una **exploración más robusta** y evitando la **convergencia prematura** a soluciones subóptimas.

SAC surge como evolución de métodos basados en actor-crítico, particularmente aquellos diseñados para entornos continuos. Se puede trazar su línea evolutiva a partir de:

- **Deep Deterministic Policy Gradient (DDPG)**: Un método **off-policy** para control continuo, que sin embargo es propenso a **problemas de estabilidad** y **exploración limitada** debido a su naturaleza **determinista** [29].

- **Soft Q-Learning**: Introdujo el concepto de **maximización de entropía** en el aprendizaje por refuerzo, inspirando la incorporación de la entropía en la función de valor [30].

- **Actor-Crítico Estocástico**: Donde la **política** se parametriza de manera **probabilística**, permitiendo la incorporación natural del término entropía.

El enfoque SAC combina estas ideas: utiliza la estructura **actor-crítico**, pero con **políticas estocásticas** y una formulación que incorpora la **maximización de la entropía**, logrando mejorar tanto la **estabilidad** del entrenamiento como la **calidad** de la exploración [31, 32].

3.6.1. Proceso de entrenamiento

El entrenamiento se basa en dos componentes principales

1. **Crítico (Q-function)**: SAC aprende una aproximación de la función de valor acción-estado, **minimizando el residuo de Bellman** modificado:

$$L(\theta) = \mathbb{E}(s, a, s') \sim D \left[\left(Q\theta(s, a) - \left(r(s, a) + \gamma \cdot \left(\min_i \hat{Q}\theta_i(s', a') - \alpha \cdot \log \pi\phi(a'|s') \right) \right) \right)^2 \right] \tag{15}$$

   Donde $r(s, a)$ es la **recompensa inmediata**, γ el **factor de descuento**, α el **coeficiente de entropía**, y π_ϕ la **política parametrizada** por ϕ.

   Se utiliza la técnica de **Double Q-Learning** para **reducir el sesgo positivo** y se emplean **redes objetivo** para **estabilizar** las actualizaciones.

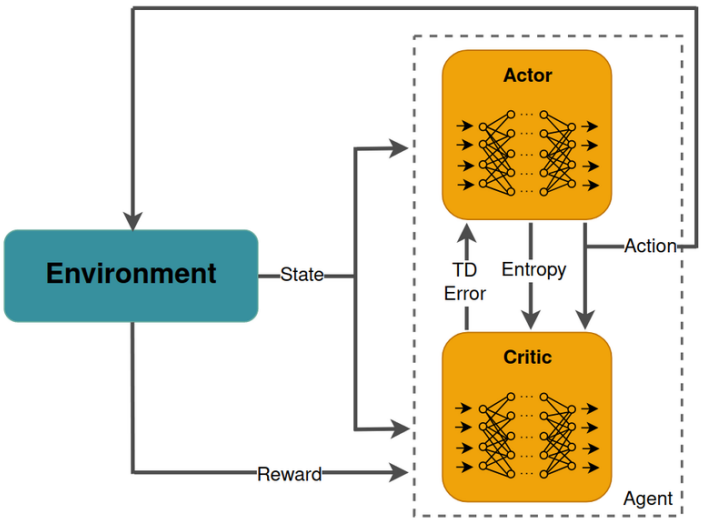
2. **Actor (Política)**: La política se actualiza **minimizando la divergencia Kullback-Leibler (KL)** entre la **política actual** y una **distribución suavizada derivada de la función** Q, lo que ajusta la política hacia acciones que **maximizan** tanto la **recompensa inmediata** como la **entropía**. La reparametrización, mediante una transformación de una **variable latente** (por ejemplo, una gaussiana), facilita la estimación eficiente de gradientes.

Una innovación clave en SAC es la capacidad de **ajustar automáticamente el coeficiente de entropía** α, equilibrando de forma dinámica la **exploración** y la **explotación** según la **señal de recompensa**, lo cual mejora la **robustez** del algoritmo frente a la selección de hiperparámetros.

Al ser un método **off-policy**, SAC reutiliza muestras almacenadas en un **buffer de reproducción**, lo que incrementa la **eficiencia** de la muestra y permite aplicar técnicas de aprendizaje profundo con redes neuronales para **aproximar tanto la política como la función** Q.

La Figura 7 muestra la arquitectura del algoritmo SAC.

Figura 7: Arquitectura de SAC.



3.6.2. Pseudocódigo

El algoritmo 4 muestra el pseudocódigo de SAC.

Algorithm 4 Soft Q-Learning with Entropy Regularization

Entrada: γ (descuento), α (coef. entropía), τ (suavizado de redes objetivo)

Inicializar redes $Q_1(\theta_1)$, $Q_2(\theta_2)$, política $\pi(\phi)$
Inicializar redes objetivo $Q'_1(\theta'_1)$, $Q'_2(\theta'_2)$
Inicializar buffer de experiencias D
for cada episodio **do**
    $estado \leftarrow entorno.reset()$
    for cada paso t **do** ▷ Exploración con política estocástica
        $accion \sim \pi(\phi) + ruido$
        $nuevo\_estado, recompensa, done \leftarrow entorno.step(accion)$
        Almacenar $(estado, accion, recompensa, nuevo\_estado, done)$ en D
        $batch \leftarrow sample(D, batch\_size)$
        ▷ Actualizar redes Q
        $Q\_target \leftarrow recompensa + \gamma \cdot (min(Q'_1, Q'_2) - \alpha \cdot \log \pi(nuevo\_estado))$
        $perdida_{Q1} \leftarrow MSE(Q_1(estado, accion), Q_{target})$
        $perdida_{Q2} \leftarrow MSE(Q_2(estado, accion), Q_{target})$
        descenso_gradiente$(\theta_1, \theta_2, perdida_{Q1} + perdida_{Q2})$
        ▷ Actualizar política (maximizar entropía)
        $acciones\_nuevas \sim \pi(\phi)$
        $perdida_\pi \leftarrow (\alpha \cdot \log \pi(acciones\_nuevas) - min(Q_1, Q_2)).mean()$
        descenso_gradiente$(\phi, perdida_\pi)$
        ▷ Actualizar redes objetivo y α (opcional)
        actualizar_redes_objetivo$(\theta'_1, \theta_1, \tau)$
        actualizar_redes_objetivo$(\theta'_2, \theta_2, \tau)$
        ajustar α
    end for
end for

3.6.3. Ventajas y consideraciones

Propuesto por Haarnoja et al. en 2018 [31, 32], SAC fue diseñado para abordar las limitaciones de algoritmos anteriores en entornos de control continuo. Los principales motivos fueron:

- **Estabilidad en el entrenamiento**: La incorporación del término de **entropía reduce la varianza** en las actualizaciones y mejora la **estabilidad** del entrenamiento en comparación con métodos deterministas como DDPG.
- **Exploración eficiente**: La maximización de la entropía incentiva la **exploración sistemática del espacio de acciones**, fundamental en entornos con altos grados de incertidumbre o con múltiples óptimos locales.
- **Robustez a la elección de hiperparámetros**: El **ajuste automático** del parámetro α permite que el algoritmo se adapte a diferentes escenarios sin requerir un costoso proceso de afinación manual, facilitando su aplicación en una amplia variedad de tareas.

SAC ha tenido un impacto significativo en el campo del aprendizaje por refuerzo, tanto en aplicaciones de robótica como en simulaciones complejas, al demostrar que un marco basado en la máxima entropía puede superar los desafíos tradicionales de la exploración y la estabilidad.

4. Descripción del experimento

En este capítulo se detalla el proceso experimental llevado a cabo para evaluar el rendimiento de los diferentes algoritmos de Aprendizaje por Refuerzo Profundo (DRL) aplicados al problema de conducción autónoma en el entorno simulado desarrollado. Se describe la configuración del entorno, los algoritmos implementados, los hiperparámetros utilizados, las métricas registradas y la plataforma sobre la que se realizaron los experimentos.

4.1. Entorno de simulación

Para asegurar la reproducibilidad y permitir una comparación justa entre los algoritmos, se estableció una configuración experimental común en la medida de lo posible. Además, se ha subido el código del proyecto a un repositorio de GitHub [1].

- **Simulador**: Se ha utilizado SUMO (Simulation of Urban MObility) versión *1.20.0*. SUMO es un simulador de tráfico microscópico, multimodal y de código abierto, ampliamente utilizado en investigación.

- **Escenario**: Los experimentos se realizaron sobre un escenario urbano denominado "*scene*" basado en la ciudad de Granada (Figura 8), cuya red (osm.net.xml.gz) y elementos visuales (osm.poly.xml.gz) fueron generados o adaptados (posiblemente a partir de datos de OpenStreetMap, como sugiere el prefijo "osm"). Este escenario incluye múltiples intersecciones, algunas de ellas reguladas por semáforos, y permite la simulación de tráfico en un entorno urbano representativo aunque de complejidad controlada.

- **Vehículo controlado**: El agente de RL controla un único vehículo con ID *"manualVehicle"* y tipo *"veh_passenger"*. Se asume que este tipo pertenece a la clase vehicular *"passenger"* para las comprobaciones de permisos en la red.

- **Dinámica de rutas**: Se ha implementado un sistema de rutas dinámicas cortas. Al inicio de cada episodio, se asigna al vehículo un *edge* de partida aleatorio (válido para su tipo) y un *edge* de destino aleatorio conectado al de partida. Una vez que el vehículo alcanza el final del *edge* de destino actual, se le asigna automáticamente un nuevo *edge* de destino aleatorio partiendo del *edge* actual, permitiendo episodios potencialmente largos y continuos dentro de la red.

- **Condiciones de episodio**: Cada episodio finaliza si ocurre alguna de las siguientes condiciones:

  - Se alcanza el número máximo de pasos ($max\_steps = 2000$).

- El vehículo sufre una colisión (detectada por SUMO o forzada por una acción inválida, como realizar un giro cuando no hay intersección).
- El vehículo es teleportado por SUMO (generalmente indica un bloqueo irrecuperable).
- El vehículo alcanza el final de su segmento de ruta actual, pero no se puede asignar un nuevo *edge* de destino válido (ej., callejón sin salida o error).

- **Espacio de estados**: Se ha definido un vector de características de **59 dimensiones** ($5+3+4+40+4+3=59$) como entrada para las redes neuronales (más detalles en la Subsubsección 4.1.1):
  - **Información del vehículo** (5 componentes): Posición (x, y normalizada), velocidad (escalar normalizada), ángulo (componentes sin/cos).
  - **Información de carril/*edge*** (3 componentes): Índice del carril actual (normalizado), número total de carriles en el *edge* (normalizado), límite de velocidad del carril (normalizado).
  - **Información de intersección** (4 componentes): Indicador booleano de si está en una intersección, indicadores booleanos de posibles maniobras (recto, izquierda, derecha) desde el carril actual.
  - **Vehículos cercanos** (8*5=40 componentes): Información de los 8 vehículos más cercanos dentro de un radio de 50 metros. Para cada vehículo cercano: posición relativa (x', y' normalizadas respecto al ego), velocidad relativa (vx', vy' normalizadas), distancia (normalizada). Se utiliza padding con valores específicos ([0, 0, 0, 0, 2.0]) si se detectan menos de 8 vehículos.
  - **Semáforos** (4 componentes): Estado del próximo semáforo (codificación *one-hot* [1] [Verde, Ámbar, Rojo]) y distancia normalizada a él.
  - **Distancia y posición relativa al objetivo** (3 componentes): Para proporcionar al agente una previsión sobre el tramo inmediatamente posterior, se incluye información normalizada sobre el **siguiente *edge*** en la ruta actual asignada al vehículo. Estas características son: posición relativa normalizada (coordenada x") del punto final de este siguiente *edge*, posición relativa normalizada (coordenada y") del punto final de este siguiente *edge* y longitud normalizada de este siguiente *edge*.

[1] La codificación *one-hot* es una técnica utilizada para representar variables categóricas (como los colores de un semáforo) como vectores binarios. Se crea un vector con tantos elementos como categorías existan. Solo el elemento correspondiente a la categoría activa toma el valor 1 ('hot'), mientras que todos los demás son 0. Para las categorías [Verde, Ámbar, Rojo], Verde sería [1, 0, 0], Ámbar [0, 1, 0] y Rojo [0, 0, 1]. Esto permite a los algoritmos procesar información categórica numéricamente sin asumir un orden inherente entre las categorías.

- **Espacio de acciones**: Se ha definido un espacio de acciones discreto con **8 acciones** posibles:

  0: Mantener acción anterior.

  1: Acelerar.

  2: Frenar/reversa.

  3: Cambiar a carril izquierdo.

  4: Cambiar a carril derecho.

  5: Girar a la izquierda en la próxima intersección.

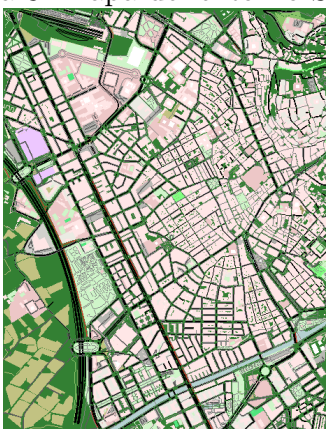
  6: Girar a la derecha en la próxima intersección.

  7: Seguir recto en la próxima intersección.

- **Función de recompensa**: Se ha diseñado una función de recompensa compuesta para guiar el aprendizaje, cuyos componentes y pesos se definen en DEFAULT_REWARD_WEIGHTS dentro de *sumo_environment.py*. Los componentes clave incluyen:

  - **Penalizaciones**: Colisión, teleportación, parada de emergencia (fuera de intersección y no por semáforo), parada larga en intersección, fallo de cambio de carril, error de ruta, ir en dirección contraria, pasar semáforo en rojo.
  - **Recompensas**: Velocidad, progreso hacia el final del *edge* actual (*approaching_goal*), alcanzar la meta del segmento (*goal_reached*), detenerse correctamente en semáforo rojo, mantener carril. Se han realizado ajustes iterativos a estos pesos durante la fase de experimentación (ver Subsección 7.1).

Figura 8: Mapa del entorno SUMO.



4.1.1. Sistema de percepción del agente (estado del entorno)

Para que el agente de Aprendizaje por Refuerzo pueda tomar decisiones informadas sobre cómo conducir, es crucial que reciba una representación adecuada de su entorno y de su propio estado. En este proyecto, en lugar de simular sensores físicos complejos como cámaras o LiDAR (lo cual añadiría una capa significativa de complejidad en el procesamiento de datos), se ha optado por un enfoque basado en la extracción directa de información relevante del simulador SUMO a través de su API **TraCI**. Este método simula un sistema de "sensores ideales" o "información perfecta" (dentro de ciertos límites) que proporcionan datos procesados directamente al agente.

La función SUMOEnvironment._get_state() es la responsable de recopilar y estructurar esta información en cada paso de tiempo. El estado resultante es un vector numérico de 59 dimensiones (presentadas en el apartado anterior), diseñado para capturar los aspectos más críticos para la tarea de conducción. Este vector se puede conceptualizar como la salida combinada de varios "sensores simulados":

1. **Propiocepción del vehículo (sensores internos)**:
   - **Posición absoluta**: Coordenadas (x, y) del vehículo en el mapa, normalizadas utilizando *MAX_MAP_COORD_X* y *MAX_MAP_COORD_Y*.
   - **Velocidad**: Velocidad escalar actual del vehículo, normalizada por *MAX_EXPECTED_SPEED*.
   - **Orientación (ángulo)**: Representada como componentes seno y coseno del ángulo de orientación del vehículo para evitar discontinuidades, sin normalización explícita ya que sin/cos están inherentemente en el rango [-1, 1].
2. **Percepción del entorno inmediato (sensores de carril/*edge*)**:
   - **Información del carril actual**: Índice del carril actual (normalizado por el número total de carriles en el *edge*) y número total de carriles en el *edge* actual (normalizado por un valor máximo esperado de carriles).
   - **Límite de velocidad del carril**: Límite de velocidad del carril/*edge* actual, normalizado por *MAX_EXPECTED_SPEED*.
   - **Estado de intersección**: Un indicador booleano que señala si el vehículo se encuentra actualmente dentro de una intersección.
3. **Percepción de maniobras posibles (planificación a corto plazo)**:

- **Enlaces de salida del carril actual**: Tres indicadores booleanos que representan si es posible seguir recto, girar a la izquierda o girar a la derecha desde el carril actual en la próxima intersección.

4. **Detección de Otros Vehículos (Sensor de Proximidad/Radar Simulado)**:
   - Se simula la detección de los **8 vehículos más cercanos** dentro de un radio de detection_radius (50 metros). La selección del número de vehículos se ha basado en un equilibrio entre relevancia local, proximidad y complejidad del espacio de estados.
   - Para cada vehículo detectado, se incluye:
     - Posición relativa (x', y') normalizada respecto al vehículo ego y rotada a su marco de referencia.
     - Velocidad absoluta normalizada del otro vehículo.
     - Velocidad relativa escalar normalizada respecto al vehículo ego.
     - Distancia normalizada al vehículo ego.
   - Si se detectan menos de 8 vehículos, se utilizan valores **placeholder** [2] para completar el vector de estado y mantener una dimensión fija. Este enfoque, aunque simplificado, permite al agente tener conciencia de los actores más relevantes en su proximidad.

5. **Percepción de señales de tráfico (sensor de semáforos)**:
   - **Estado del próximo semáforo**: Codificado en formato one-hot ([Verde, Ámbar, Rojo]) si hay un semáforo relevante en la ruta del vehículo.
   - **Distancia al próximo semáforo**: Distancia normalizada al semáforo detectado.

6. Información de ruta/objetivo (planificación a medio plazo):
   - **Características del siguiente *edge* de la ruta**: Para proporcionar al agente una previsión más allá del *edge* actual, se incluyen:
     - Posición relativa normalizada (x", y") del punto final del siguiente edge en la ruta actual del vehículo, transformada al marco de referencia del vehículo ego.
     - Longitud normalizada del siguiente *edge* en la ruta.
   - Esta información ayuda al agente a anticipar la geometría y la escala del próximo segmento de su trayectoria asignada.

[2] Un **placeholder** es un elemento temporal o simbólico que se utiliza para reservar espacio o representar datos que aún no están disponibles o definidos. Es común en programación y diseño de interfaces, como en formularios web, donde un placeholder puede mostrar texto indicativo dentro de un campo de entrada para guiar al usuario sobre qué información introducir.

La normalización de la mayoría de estos valores se realiza para mantener las entradas a la red neuronal en un rango similar, lo que generalmente ayuda a la estabilidad y eficiencia del entrenamiento. La selección y estructuración de estas 59 características busca un equilibrio entre proporcionar suficiente información para una toma de decisiones compleja y mantener un espacio de estados manejable para los algoritmos de DRL.

4.2. Algoritmos evaluados

Se han implementado y evaluado los siguientes algoritmos DRL, buscando comparar enfoques basados en valor y basados en política/actor-crítico:

- **Dueling Double Deep Q-Network (DDQN)**: Una mejora sobre DQN que utiliza redes separadas para estimar el valor del estado (V) y la ventaja de cada acción (A), y desacopla la selección de la acción de la evaluación de su valor para reducir la sobreestimación (implementado en ddqn_agent.py).
- **Asynchronous Advantage Actor-Critic (A3C)**: Un algoritmo actor-crítico basado en política que utiliza múltiples workers (hilos) para explorar el entorno en paralelo y actualizar una red global de forma asíncrona (implementado en a3c_agent.py).
- **Proximal Policy Optimization (PPO)**: Un algoritmo actor-crítico basado en política on-policy que optimiza un objetivo sustituto (surrogate objective) utilizando un recorte (clipping) para limitar los cambios en la política entre actualizaciones, mejorando la estabilidad (implementado en ppo_agent.py).
- **Soft Actor-Critic (SAC)**: Un algoritmo actor-crítico basado en política off-policy que busca maximizar tanto la recompensa acumulada como la entropía de la política, fomentando una mejor exploración y robustez. Se implementó una versión adaptada para el espacio de acciones discreto, con ajuste automático opcional del coeficiente de entropía alfa (implementado en sac_agent.py).

4.3. Hiperparámetros principales

Los hiperparámetros clave para cada algoritmo se definieron centralizadamente en el archivo config.py. La tabla resume los valores utilizados en la mayoría de los experimentos finales, aunque algunos (notablemente para PPO) fueron ajustados tras observar resultados iniciales (ver Sección 7). Se habilitó el recorte de gradientes ($clipping = True$) para los algoritmos que lo soportaban en la implementación (DDQN, PPO, A3C).

Cuadro 1: Hiperparámetros principales utilizados para cada algoritmo.

Hiperparámetro	DDQN	A3C	PPO	SAC
gamma (γ)	0.99	0.99	0.99	0.99
lr / actor_lr	0.0005	0.0001	0.0001	0.0003
critic_lr	N/A	N/A	N/A	0.0003
epsilon_decay	0.999	N/A	N/A	N/A
epsilon_min	0.05	N/A	N/A	N/A
target_update (pasos)	1000	N/A	N/A	N/A
tau (actualización suave)	N/A	N/A	N/A	0.005
batch_size	64	N/A (rollout)	32	256
memory_size / buffer_size	50000	N/A	N/A (on-policy)	1000000
gae_lambda (λ)	N/A	0.95	0.95	N/A
clip_epsilon (ϵ)	N/A	N/A	0.2	N/A
vf_coef (pérdida valor)	N/A	0.5	0.5	N/A
entropy_coef (pérdida entropía)	N/A	0.01	0.05	N/A
alpha (entropía SAC)	N/A	N/A	N/A	0.2 (inicial, auto)
learn_alpha (SAC)	N/A	N/A	N/A	True
alpha_lr (SAC)	N/A	N/A	N/A	0.0003
n_steps (rollout)	N/A	20	64	N/A
n_epochs (opt PPO)	N/A	N/A	3	N/A
n_workers (A3C)	N/A	4	N/A	N/A
max_grad_norm (clipping)	1.0	0.5	0.5	N/A

Nota: Los N/A indican que ese parámetro no existe para dicho algoritmo y por lo tanto no procede asignarle valor.

4.4. Métricas registradas

Para evaluar y comparar el rendimiento y el proceso de aprendizaje, se han registrado las siguientes métricas:

- **Durante el entrenamiento**:
  - Número de episodio.
  - Recompensa total del episodio.
  - Número de pasos del episodio.
  - Número total de pasos acumulados.
  - Recompensa promedio móvil.
  - Número de metas alcanzadas en el episodio.
  - Metas alcanzadas promedio móvil.

  - Pérdidas específicas del algoritmo (promediadas por episodio): loss (DDQN), policy_loss, value_loss, entropy_loss (PPO), actor_loss, critic_loss, alpha_loss (SAC), valor de epsilon (DDQN), valor de alpha (SAC).

- **Durante la evaluación**:
  - Número de episodio de evaluación.
  - Recompensa total del episodio.
  - Número de pasos del episodio.
  - Número de metas alcanzadas en el episodio.
  - Número de colisiones (forzadas o detectadas).
  - Número de teleportaciones.
  - Número de paradas de emergencia (o paradas largas en intersección).

Todos los datos de métricas se han guardaro en archivos *.csv* y se han generado gráficos *.png* para visualización.

4.5. Plataforma experimental

Los experimentos se han llevado a cabo principalmente en un ordenador portátil personal con las siguientes especificaciones aproximadas:

- **CPU**: Apple M2 (Arquitectura ARM)
- **RAM**: 16 GB
- **GPU**: GPU integrada del Apple M2 (Aunque PyTorch puede usarla vía MPS, el cuello de botella principal suele ser SUMO en CPU).
- **Sistema operativo**: macOS
- **Software clave**: Python 3.12, PyTorch, SUMO , TraCi, NumPy, Pandas, Matplotlib, Seaborn.

5. Resultados del entrenamiento

En este apartado se presentan y analizan los resultados obtenidos durante la fase de entrenamiento para cada uno de los algoritmos implementados: DDQN, A3C, PPO y SAC. El análisis se centra en las métricas clave registradas (recompensa, pasos, metas alcanzadas, pérdidas) para evaluar la convergencia, estabilidad y eficacia del aprendizaje de cada método en el entorno SUMO configurado.

5.1. Dueling Deep Q-Network (DDQN)

Se ha entrenado el agente DDQN durante 10000 episodios. Los hiperparámetros utilizados se detallan en la Subsección 4.3 y config.py.

5.1.1. Curvas de aprendizaje

Figura 9: Pasos por episodio - entrenamiento (DDQN).

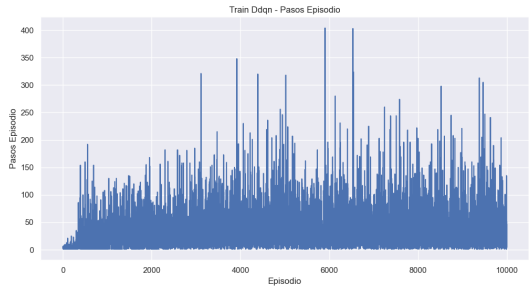


Figura 10: Pasos promedio móvil - entrenamiento (DDQN).

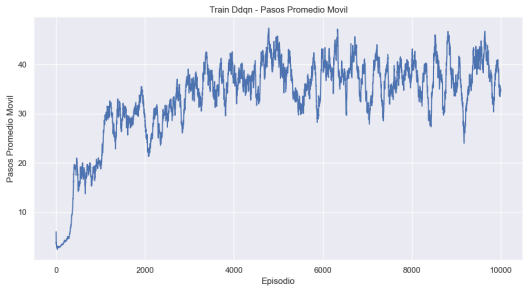


Figura 11: Recompensa por episodio - entrenamiento (DDQN).

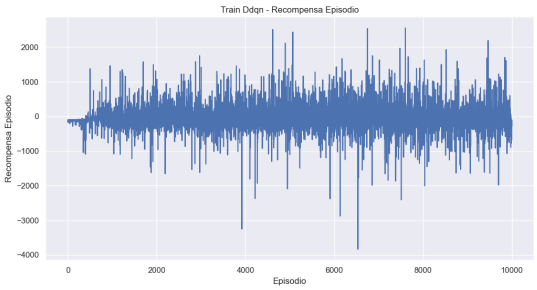


Figura 12: Recompensa promedio móvil - entrenamiento (DDQN).

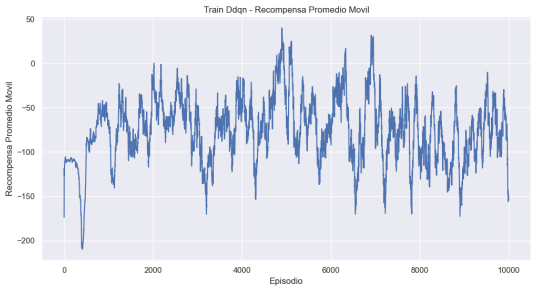


Figura 13: Metas alcanzadas por episodio - entrenamiento (DDQN).

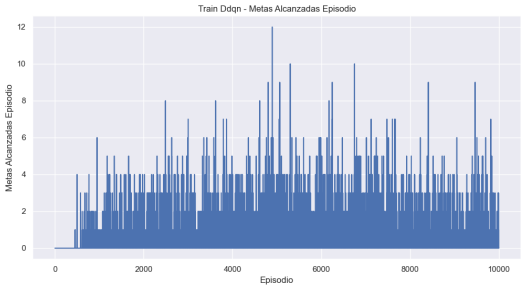
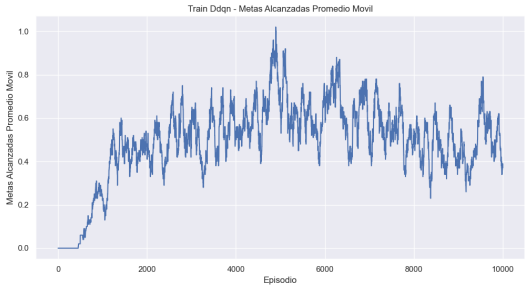


Figura 14: Metas alcanzadas promedio móvil - entrenamiento (DDQN).



5.1.2. Análisis de convergencia y rendimiento

- **Recompensa:** La Figura 12 muestra una clara mejora inicial. La recompensa promedio móvil comienza en valores muy negativos (por debajo de -150) y aumenta significativamente durante los primeros 1000-1500 episodios, alcanzando un máximo local cercano a -25. Sin embargo, después de este punto, el rendimiento no mejora de forma sostenida, sino que entra en una fase de alta oscilación, variando entre aproximadamente -25 y -125. Esto indica que el agente aprende a evitar los peores resultados iniciales, pero lucha por encontrar una política consistentemente buena y estable. La recompensa por episodio (Figura 11) es muy ruidosa, pero confirma la aparición de episodios con recompensas positivas altas (> 500), aunque mezclados con muchos episodios de recompensa negativa.

- **Pasos por episodio:** La Figura 10 y la Figura 9 corroboran el aprendizaje inicial. La duración promedio de los episodios aumenta notablemente desde valores muy bajos hasta estabilizarse alrededor de los 40-50 pasos promedios, con picos en episodios individuales que alcanzan los 400 pasos. Esto demuestra que el agente aprende a sobrevivir más tiempo, lo cual es un requisito previo para alcanzar metas. No obstante, la persistencia de episodios muy cortos (<25 pasos) indica que los fallos prematuros (colisiones, errores) siguen siendo comunes a lo largo de todo el entrenamiento.

- **Metas alcanzadas:** La Figura 14 es la evidencia más fuerte del aprendizaje. La media móvil de metas alcanzadas por episodio pasa de cero a un valor promedio que oscila entre 0.3 y 0.8, e incluso picos cercanos a 1.0 en el promedio móvil. El gráfico de metas por episodio individual (Figura 13) indica picos donde se alcanzan múltiples metas (hasta 10-12). Esto demuestra que el agente aprende el valor de la recompensa `goal_reached` y desarrolla estrategias para completar segmentos de ruta. Sin embargo, la media móvil general, aunque positiva, sugiere que no logra completar metas consistentemente en la

mayoría de los episodios.

5.1.3. Conclusión

El agente DDQN muestra una capacidad de aprendizaje significativa en este entorno, superando el comportamiento inicial aleatorio y aprendiendo a completar segmentos de ruta y a sobrevivir durante más pasos en promedio. Logra alcanzar múltiples metas en episodios individuales. Sin embargo, sufre de una considerable inestabilidad en el rendimiento a lo largo de los 10,000 episodios, sin converger a una política robusta y consistentemente buena. La alta variabilidad en la recompensa y la persistencia de episodios cortos, incluso después de miles de episodios de entrenamiento, sugieren problemas continuos con la exploración eficiente o la estabilidad de la estimación de valor en ciertas situaciones del entorno simulado.

5.2. Asynchronous Advantage Actor Critic (A3C)

La implementación del algoritmo A3C, si bien teóricamente atractiva por su potencial de paralelismo y eficiencia de datos, ha presenado una serie de **obstáculos técnicos significativos e irresolubles** dentro de la configuración experimental de este proyecto, impidiendo la obtención de resultados de entrenamiento concluyentes. Estos desafíos están centrados principalmente en la **gestión concurrente de múltiples simulaciones SUMO** y sus correspondientes **conexiones TraCI desde hilos (threading) de Python**.

5.2.1. Descripción de los desafíos técnicos

Problema principal de Thread-Safety en traci.start()

La raíz del problema reside en que la función estándar *traci.start(sumo_cmd, port=...)*, utilizada para lanzar un proceso SUMO externo y establecer una conexión TraCI con él, no es intrínsecamente segura para hilos (thread-safe) cuando se invoca concurrentemente desde múltiples hilos para iniciar instancias separadas de SUMO. La librería TraCI en Python parece mantener internamente un estado global (posiblemente relacionado con un pool de conexiones o la referencia a la conexión activa). Cuando múltiples hilos llaman a *traci.start()* casi simultáneamente, incluso especificando puertos únicos, se producen condiciones de carrera al intentar modificar este estado global compartido. Un hilo puede sobrescribir el estado interno establecido por otro hilo antes de que este último complete su secuencia de inicialización o envíe su primer comando.

Manifestación del error

Este conflicto interno se manifestó consistentemente como un *'traci.exceptions.FatalTraCIError: Not connected.'* que ocurría después de que la función

traci.connect (llamada internamente por *traci.start* o explícitamente en el enfoque con subprocess) reportara una conexión aparentemente exitosa en un puerto específico. El error ocurría típicamente en la primera llamada a *traci.simulationStep()* dentro del método *reset()* del entorno del worker. Esto sugiere que, aunque la conexión TCP a nivel de socket se establecía brevemente, el estado interno de la librería TraCI se corrompía por las llamadas concurrentes de otros hilos, haciendo que la librería perdiera la referencia a la conexión correcta o la considerara inválida al intentar usarla inmediatamente después.

Errores de inicio del proceso SUMO

En las fases iniciales de depuración, también se observaron errores *'RuntimeError: No se pudo iniciar la simulación...'* acompañados de *'Connection refused'*. Aunque se solucionaron problemas como puertos duplicados y rutas relativas, la persistencia de estos errores en los workers A3C sugiere que el lanzamiento concurrente de procesos sumo desde hilos también podría estar causando conflictos a nivel del sistema operativo (agotamiento temporal de recursos, bloqueos) o errores internos silenciosos en SUMO que provocan su cierre prematuro antes de que TraCI pueda conectar de forma estable.

Intentos de mitigación y sus limitaciones

- **Puertos únicos**: Asignar un puerto TCP distinto a cada instancia de SUMO/TraCI (ej., 8813, 8815, 8817...) fue un paso necesario para evitar conflictos directos de *binding* [3], pero no resolvió el problema del estado global compartido dentro de la librería TraCI.

- **Rutas absolutas**: Utilizar rutas absolutas para el archivo .sumocfg descartó problemas de directorio de trabajo relativo en los hilos, pero no afectó la condición de carrera interna de TraCI.

- **Retardos entre workers**: Introducir pausas entre el inicio de cada worker redujo la simultaneidad del lanzamiento de procesos sumo, pero no eliminó la posibilidad de que las llamadas a *traci.connect* (o las operaciones internas de *traci.start*) aún ocurrieran de forma concurrente y conflictiva una vez que los hilos estaban activos.

- **subprocess.Popen + traci.connect**: Se ha intentado desacoplar el inicio del proceso SUMO (usando *subprocess*) de la conexión TraCI (usando *traci.connect*). Aunque esto

[3]En programación, "**binding**"(vinculación o enlace, en español) se refiere al proceso de **asociar un identificador** (como el nombre de una variable o función) **con un valor, objeto, dirección de memoria o implementación específica**. Esta asociación permite que, cuando usas un identificador en código, el sistema sepa a qué çosareal te estás refiriendo. Por ejemplo, en 'x = 5', se vincula el nombre 'x' al valor entero '5'; al llamar a una función, se vincula el nombre de la función con el bloque de código que la ejecuta; y en el contexto de redes, "binding.[a] un puerto significa asociar un socket específico de tu programa a una dirección IP y número de puerto concretos para poder escuchar o enviar datos a través de esa combinación [33].

evitaba usar la parte problemática de *traci.start*, el error *'FatalTraCIError: Not connected.'* persistió, indicando que *traci.connect* en sí misma (o las operaciones subsiguientes inmediatas como *simulationStep*) también sufren de problemas de thread-safety o estado global cuando múltiples hilos la usan para gestionar conexiones distintas. La serialización de *traci.connect* mediante un *threading.Lock* tampoco resolvió completamente el problema, sugiriendo que la interferencia podría ocurrir en otras funciones de TraCI que acceden al estado global.

5.2.2. Resultados

Debido a estos problemas técnicos irresolubles dentro del marco del proyecto, no ha sido posible completar una ejecución de entrenamiento estable y significativa para el agente A3C. Por lo tanto, no se dispone de curvas de aprendizaje ni métricas de rendimiento comparables a las de otros algoritmos.

5.2.3. Conclusión

La dificultad encontrada subraya los retos de implementar algoritmos DRL paralelos que interactúan con simuladores externos complejos como SUMO usando únicamente el módulo threading de Python. La gestión del estado global de TraCI y el lanzamiento de procesos externos parecen ser los principales cuellos de botella. Alternativas como el uso del módulo multiprocessing (que proporciona aislamiento de memoria) o librerías especializadas en RL paralelo (como Ray RLlib, que abstraen gran parte de la gestión de procesos y comunicación) serían líneas de investigación futuras más prometedoras para implementar A3C o algoritmos similares en este contexto [34, 35].

5.3. Proximal Policy Optimization (PPO)

Se ha entrenado el agente PPO durante 5000 episodios. Se realizaron ajustes significativos en los hiperparámetros después de observar una divergencia en los experimentos iniciales, reduciendo n_steps a 64, n_epochs a 3, lr a 0.0001 y aumentando entropy_coef a 0.05 (ver config.py).

5.3.1. Curvas de aprendizaje

Figura 15: Pasos por episodio - entrenamiento (PPO).

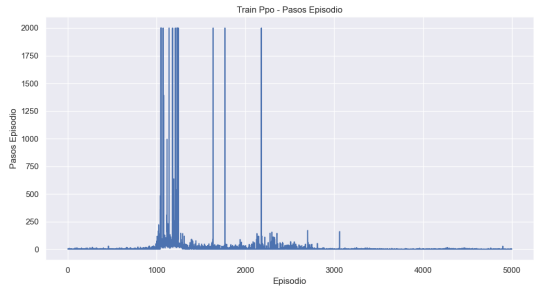


Figura 16: Pasos promedio móvil - entrenamiento (PPO).

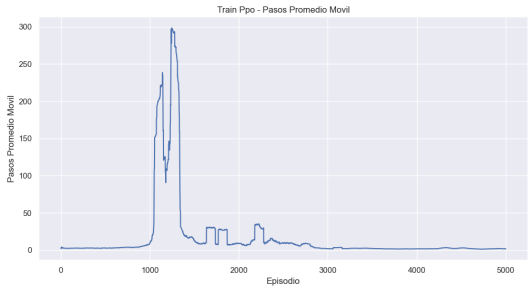


Figura 17: Recompensa por episodio - entrenamiento (PPO).

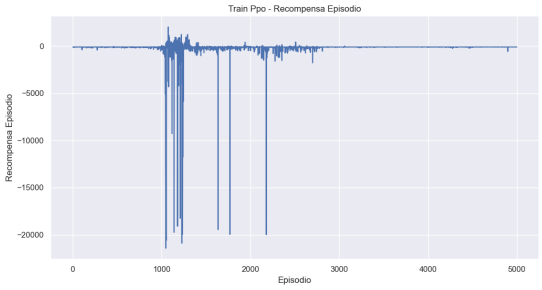


Figura 18: Recompensa promedio móvil - entrenamiento (PPO).

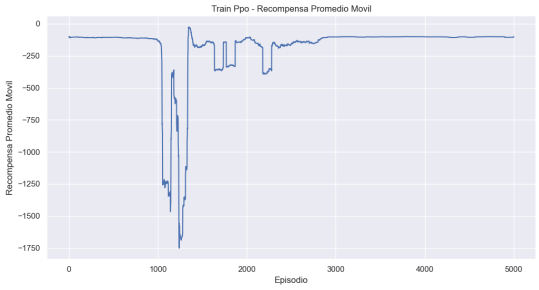


Figura 19: Metas alcanzadas por episodio - entrenamiento (PPO).

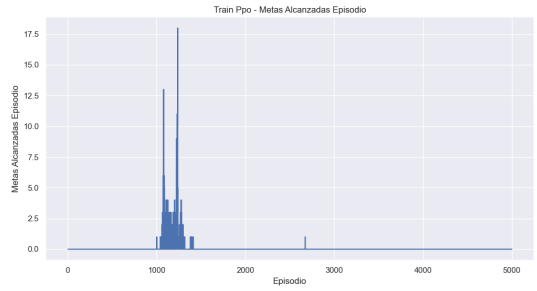


Figura 20: Metas alcanzadas promedio móvil - entrenamiento (PPO).

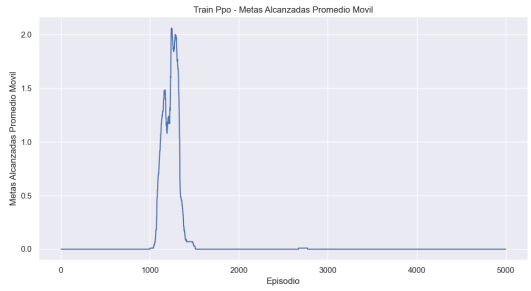


Figura 21: Pérdida de valor (Value Loss) - entrenamiento (PPO).

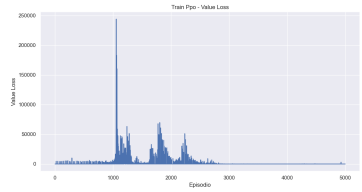


Figura 22: Pérdida de política (Policy Loss) - entrenamiento (PPO).

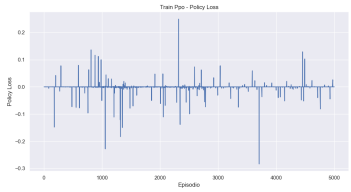
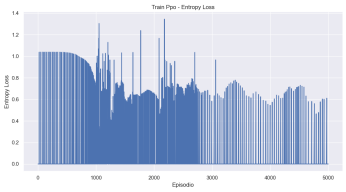


Figura 23: Pérdida de entropia (Entropy Loss) - entrenamiento (PPO).



5.3.2. Análisis de convergencia y rendimiento

- **Colapso inicial**: La Figura 18 muestra un comportamiento dramático. Tras un inicio cercano a -100, la recompensa promedio se desploma a valores extremadamente bajos (< -2500) entre los episodios 1000 y 1500 aproximadamente. Esto coincide con una explosión en la Value Loss (Figura 21) y un aumento repentino en la duración de los episodios (Figura 15), donde el agente parece quedarse atascado durante los 2000 pasos máximos. Este comportamiento sugiere una divergencia o colapso de la **función de valor y/o la política** durante esa fase.

- **Recuperación y estancamiento**: Después del colapso, la recompensa promedio se recupera rápidamente y se estabiliza cerca de -5, un valor significativamente mejor que el inicial. Sin embargo, no muestra una tendencia ascendente continua después del episodio 1800. Se queda estancada en ese nivel ligeramente negativo.

- **Metas alcanzadas**: La Figura 20 es reveladora. Muestra un breve período (coincidiendo con la caida de la recompensa) donde el agente aprende a alcanzar algunas metas (promedio móvil $> 0{,}6$). En estos episodios el agente alcanza algunas metas, pero luego se queda estancado acumulando una recompensa muy negativa en el episodio pese a haber alcanzado algún objetivo. Posteriormente vuelve a caer abruptamente a cero y permanece allí durante el resto del entrenamiento. Esto indica que la política aprendida durante la fase de recuperación no era robusta y colapsó a una estrategia que ya no busca ni alcanza las metas.

- **Pasos y pérdidas**: Los **pasos por episodio** (Figura 16) reflejan este colapso: después del bloque de episodios largos (1000-1700), vuelven a ser consistentemente bajos (< 20 pasos), indicando fallos rápidos. La **pérdida de valor** (Figura 21), tras explotar, se estabiliza cerca de cero, lo que podría indicar que la red de valor aprendió a predecir un valor bajo y constante, o que las ventajas son muy pequeñas debido a los episodios cortos.

La **pérdida de política** (Figura 22) es muy ruidosa, con picos positivos y negativos, sin mostrar una tendencia clara hacia la minimización (recordemos que **PPO minimiza -L_CLIP**), lo que sugiere actualizaciones erráticas. La **pérdida de entropía** (Figura 23) empieza alta (exploración inicial), luego cae drásticamente a cero durante la fase de colapso/recuperación (indicando que la política se volvió muy determinista/segura de sí misma, probablemente de forma prematura y errónea) y, aunque intenta recuperarse ligeramente después, permanece en niveles bajos, lo que es consistente con una política estancada y poco exploratoria. El ajuste del entropy_coef a 0.05 no parece haber sido suficiente para mantener la exploración activa a largo plazo.

5.3.3. Conclusión

A pesar de los ajustes de hiperparámetros, PPO no ha logrado aprender una política útil y estable para este problema. Ha experimentado una divergencia severa inicial, seguida de una recuperación parcial pero finalmente ha colapsado a una política ineficaz que no logra los objetivos. Las causas podrían ser una combinación de: **sensibilidad residual a hiperparámetros**, la **escala de recompensas** que aún causan inestabilidad numérica, o la propia **dificultad del problema de exploración** para un algoritmo **on-policy** como PPO en este entorno con fallos frecuentes.

5.4. Soft Actor-Critic (SAC)

Se ha entrenado el agente SAC (versión discreta) durante 3000 episodios y se ha utilizado un ajuste automático de alfa.

La versión discreta de SAC adapta el algoritmo original, diseñado para espacios de acción continuos, al caso en el que el agente elige entre un conjunto finito de acciones. En este contexto, la política aprende una distribución categórica sobre las acciones posibles en lugar de una distribución continua.

5.4.1. Curvas de aprendizaje

Figura 24: Pasos por episodio - entrenamiento (SAC).

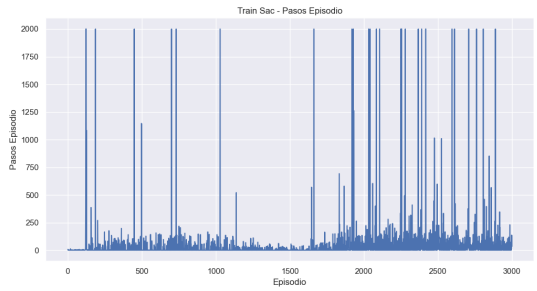


Figura 25: Pasos promedio móvil - entrenamiento (SAC).

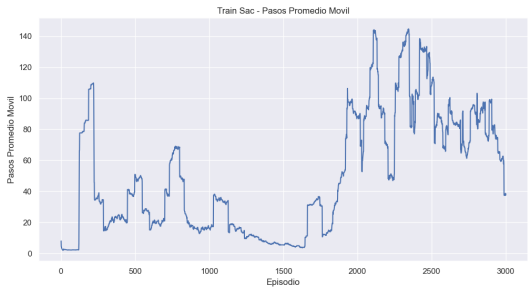


Figura 26: Recompensa por episodio - entrenamiento (SAC).

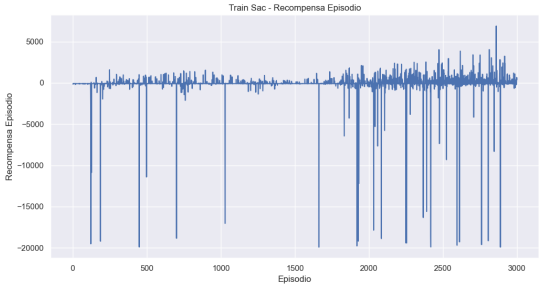


Figura 27: Recompensa promedio móvil - entrenamiento (SAC).

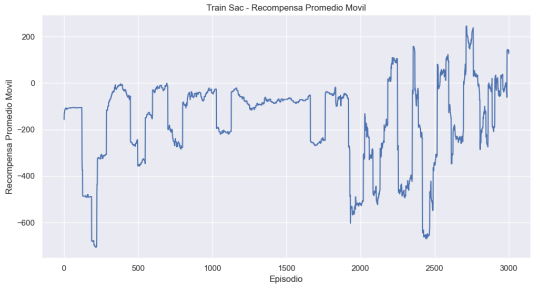


Figura 28: Metas alcanzadas por episodio - entrenamiento (SAC).

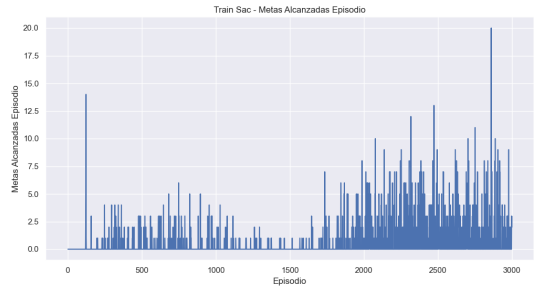
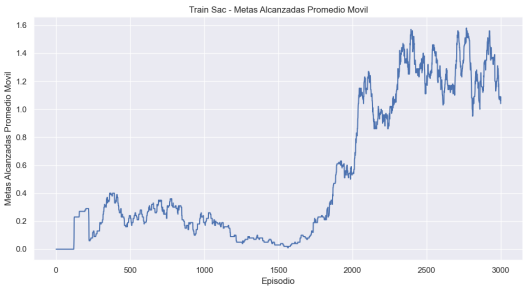


Figura 29: Metas alcanzadas promedio móvil - entrenamiento (SAC).



5.4.2. Análisis de convergencia y rendimiento

- **Recompensa:** La recompensa promedio móvil (Figura 27) muestra una evolución compleja. Inicialmente, hay un aprendizaje rápido donde la recompensa sube desde valores muy negativos hasta alcanzar picos positivos alrededor de los episodios 300-500 y nuevamente cerca del episodio 800. Posteriormente, entre los episodios 1000 y 1800 aproximadamente, el rendimiento en recompensa promedio móvil tiende a disminuir y estabilizarse en valores negativos o cercanos a cero. Hacia el final del entrenamiento (episodios 1900-3000), se observa una nueva fase de mejora significativa, con la recompensa promedio móvil alcanzando sus valores más altos, superando consistentemente los +100 e incluso llegando a picos por encima de +200. La recompensa por episodio individual (Figura 26) es extremadamente volátil a lo largo de todo el entrenamiento, con episodios que logran recompensas muy elevadas (superiores a 5000) pero también muchos otros con recompensas profundamente negativas (inferiores a -15000), especialmente en las fases de menor recompensa promedio.

- **Pasos por episodio:** El promedio móvil de pasos por episodio (Figura 25) sigue una tendencia general correlacionada con la recompensa. Se observa un aumento inicial, seguido de una fase de estancamiento o ligera disminución en la duración promedio de los episodios (coincidiendo con la fase de menor recompensa promedio), y finalmente un incremento notable hacia el final del entrenamiento, donde el promedio móvil de pasos supera consistentemente los 60-80 pasos y alcanza picos de hasta 140 pasos. La gráfica de pasos por episodio individual (Figura 24) muestra que, especialmente en las fases de mejor rendimiento (inicial y final), el agente frecuentemente alcanza el límite máximo de 2000 pasos. Esto indica que el agente aprende a sobrevivir durante periodos prolongados cuando su política es más efectiva. Sin embargo, la caída de la recompensa en estos casos sugiere que el vehículo símplemente se queda estancado en un punto fijo acumulando recompensa negativa, lo cual no es un comportamiento deseado.

- **Metas alcanzadas:** El promedio móvil de metas alcanzadas (Figura 29) es un claro indicador del progreso del aprendizaje. Comienza en cero, muestra una mejora inicial alcanzando un promedio de 0.2-0.4 metas, luego experimenta una fase de rendimiento decreciente (entre episodios 1200-1800) donde el promedio cae a valores muy bajos (cercanos a 0.0-0.1). A partir del episodio 1800 aproximadamente, se produce un aprendizaje muy significativo y sostenido, con el promedio móvil de metas superando consistentemente 1.0 y alcanzando picos de hasta 1.6 metas por episodio hacia el final del entrenamiento. El gráfico de metas por episodio individual (Figura 28) confirma esta tendencia, mostrando que en la fase final del entrenamiento el agente es capaz de alcanzar un número

considerable de metas en muchos episodios (picos de hasta 20 metas).

5.4.3. Conclusión

El agente SAC demuestra una notable capacidad de aprendizaje a lo largo de los 3000 episodios, aunque su trayectoria no es lineal. Inicialmente, logra mejoras rápidas tanto en recompensa como en la consecución de metas, probablemente gracias a la exploración incentivada por la maximización de entropía. Sin embargo, atraviesa una fase intermedia de estancamiento o incluso regresión en su rendimiento, donde la recompensa promedio y las metas alcanzadas disminuyen, a pesar de mantener una duración de episodios variable.

Es en la fase final del entrenamiento (aproximadamente los últimos 1000-1200 episodios) donde SAC muestra su mayor potencial, logrando los mejores promedios móviles en recompensa, pasos por episodio y, de forma destacada, en metas alcanzadas. Esto sugiere que, tras un periodo de exploración o ajuste subóptimo, el agente finalmente converge hacia políticas más efectivas que le permiten no solo sobrevivir más tiempo sino también cumplir con los objetivos de la ruta de manera más consistente.

A pesar de este progreso, la alta volatilidad en la recompensa por episodio a lo largo de todo el entrenamiento indica que la política aprendida, aunque capaz de lograr altos rendimientos, aún puede incurrir en comportamientos que resultan en penalizaciones severas. La estabilidad a largo plazo y la robustez ante todas las situaciones del entorno siguen siendo áreas de mejora. No obstante, de los algoritmos probados, SAC es el que muestra la **trayectoria de aprendizaje más prometedora** y los mejores resultados finales en las métricas clave durante el entrenamiento.

5.5. Comparativa

Figura 30: Metas alcanzadas - entrenamiento.

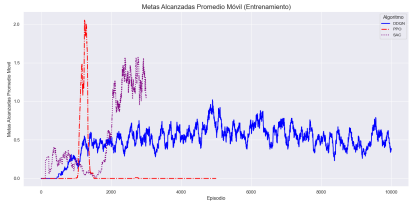


Figura 31: Pasos - entrenamiento.

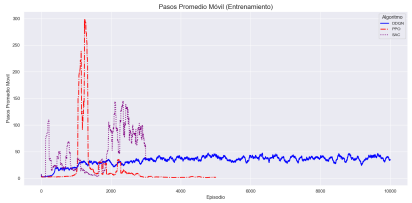
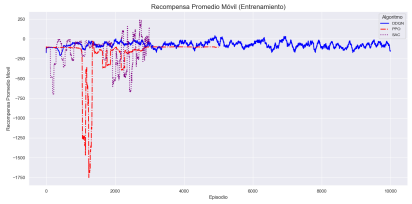


Figura 32: Recompensas - entrenamiento.



5.6. Estimación de la huella de carbono del entrenamiento con CodeCarbon

En un esfuerzo por cuantificar el impacto ambiental asociado al entrenamiento de los modelos de Aprendizaje por Refuerzo Profundo desarrollados en este TFM, se ha utilizado la herramienta **CodeCarbon** [36, 37]. CodeCarbon es una librería de Python que estima las emisiones de dióxido de carbono (CO_2eq) generadas por la ejecución de código, basándose en el consumo energético del hardware y la intensidad de carbono de la red eléctrica de la región donde se realiza el cómputo.

5.6.1. Metodología de medición

La librería CodeCarbon se integró en el script principal para rastrear el consumo energético durante la fase de entrenamiento de los algoritmos DDQN, PPO y SAC. Se configuró el EmissionsTracker de CodeCarbon con los siguientes parámetros principales para cada ejecución de entrenamiento:

- **project_name**: Un nombre identificativo para cada algoritmo (ej., "Train_DDQN", "Train_PPO", "Train_SAC").
- **output_dir**: Los reportes individuales se guardaron en models/results/codecarbon_reports/.
- **measure_power_secs**: Intervalo de muestreo de energía de 10 segundos.
- **tracking_mode**: Se utilizó el modo "process"para intentar aislar el consumo del script Python y sus subprocesos (incluyendo SUMO), en lugar de medir la máquina completa. Esto se eligió para obtener una estimación más específica del impacto del TFM, dado que el hardware utilizado (Apple M2) es una máquina personal donde otros procesos podrían estar activos.
- **Ubicación**: Las mediciones se realizaron en Granada, España (detectado automáticamente por CodeCarbon como región "*andalusia*", ISO code "*ESP*").

Es importante notar que CodeCarbon emitió una advertencia indicando que no posee un perfil de consumo energético específico para el procesador Apple M2 (*"We saw that you have a Apple M2 but we don't know it. Please contact us."*). Por lo tanto, las estimaciones de consumo de CPU pueden basarse en aproximaciones más genéricas, lo que podría afectar la precisión absoluta de los valores. Sin embargo, los resultados siguen siendo útiles para una comparación relativa y una concienciación sobre el coste computacional.

5.6.2. Resultados

Los datos de las ejecuciones de entrenamiento de los algoritmos DDQN, PPO y SAC se unificaron para obtener una visión global. La Tabla 2 presenta un resumen de estos resultados, pero se pueden ver más detalles en unified_emissions_agents_train.csv.

Cuadro 2: Resumen de emisiones de carbono estimadas por CodeCarbon para el entrenamiento de agentes

Algoritmo (project_name)	Duración (segundos)	Emisiones (kg CO_2eq)	Energía Cons. (kWh)	CPU Power (W)	GPU Power (W)	RAM Power (W)
Train_DDQN	56 198,01	0,026 360	0,151 453	6,92	0,06	3,0
Train_PPO	28 446,42	0,012 778	0,073 414	7,08	0,35	3,0
Train_SAC	18 991,77	0,009 191	0,052 807	7,72	0,06	3,0
Total	**103636,20**	**0,048329**	**0,277674**	N/A	N/A	N/A

Nota: Las potencias promedio no se suman en la fila total.

5.6.3. Análisis y conclusiones

La utilización de CodeCarbon en este proyecto proporciona una estimación del coste computacional y el impacto ambiental asociados al entrenamiento de los agentes DDQN, PPO y SAC.

Es importante señalar que el número de episodios de entrenamiento no es idéntico para todos los algoritmos; esta diferencia se estableció en función de la velocidad observada con la que cada algoritmo tendía a alcanzar (o aproximarse a) su óptimo local y a estabilizar su rendimiento, buscando un balance entre el tiempo de cómputo y la exploración de su capacidad de aprendizaje.

Los resultados agregados Tabla 2 ofrecen las siguientes perspectivas:

- El conjunto de los entrenamientos (DDQN con 10,000 ep., PPO con 5,000 ep., SAC con 3,000 ep.) duró alrededor de 28.8h, generó un total estimado de 0.0483kg de CO_2eq y consumió 0.278kWh de energía.
- El entrenamiento más extenso resultó ser **DDQN**, como es de esperar, ya que ha requerido de un mayor número de episodios de entrenamiento.
- **PPO** requiere un mayor consumo de potencia para GPU, pero aun así mantiene unos niveles de consumo proporcionales en tiempo y episodios a DDQN.
- **SAC** es el más rapido de entrenar, ya que requiere un menor número de episodios para alcanzar resultados significativos, y por tanto el que menos emisiones produce.

Aunque las cifras absolutas son relativamente modestas para un proyecto de esta escala, estos resultados sirven para:

- Concienciar sobre el coste energético y ambiental asociado a la investigación en DRL, incluso a nivel de TFM.
- Establecer una línea base para futuras optimizaciones. Si se mejorara la eficiencia de los algoritmos (logrando convergencia más rápida o con menos recursos), se esperaría una reducción en estas cifras.
- Destacar la importancia de herramientas como CodeCarbon para visibilizar y cuantificar este impacto, fomentando prácticas de "Green AI".

6. Comparativa de rendimiento en evaluación

Tras la fase de entrenamiento, se ha llevado a cabo una fase de evaluación para medir el rendimiento de los modelos finales de los algoritmos DDQN, PPO y SAC en el entorno SUMO configurado. El objetivo de esta sección es comparar su capacidad para generalizar lo aprendido y desenvolverse en episodios no vistos previamente, centrándose en métricas clave de eficiencia, finalización de tareas y seguridad.

6.1. Resultados de evaluación por episodio

En este apartado se presentan las gráficas que muestran el rendimiento de cada algoritmo evaluado (DDQN, PPO, SAC) a lo largo de los 20 episodios de la fase de evaluación.

Para ello, se ha definido un método generate_eval_routes(), encargado de generar una serie de rutas aleatorias y de guardarlas en el fichero eval_routes.json para poder evaluar todos los agentes con los mismos objetivos y en igualdad de condiciones.

Las siguientes gráficas permiten observar la variabilidad del rendimiento entre episodios individuales, aunque se profundizará más acerca de los resultados obtenidos en los siguientes apartados de esta sección.

6.1.1. DDQN

Figura 33: Colisiones - evaluación (DDQN).

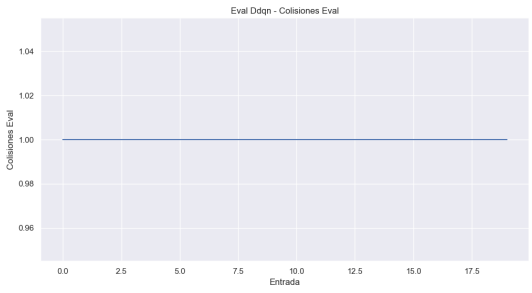


Figura 34: Tasa de éxito - evaluación (DDQN).

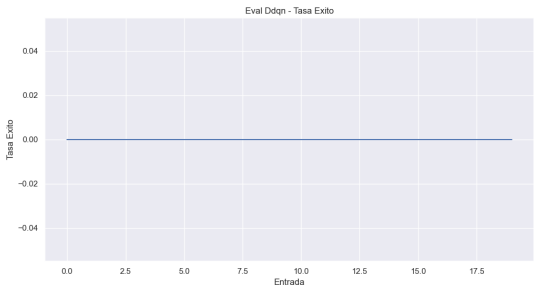


Figura 35: Metas alcanzadas - evaluación (DDQN).

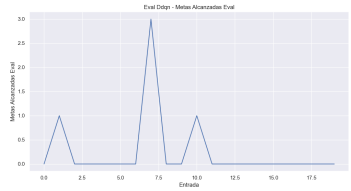


Figura 36: Pasos - evaluación (DDQN).

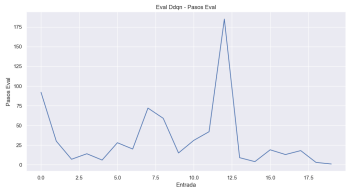
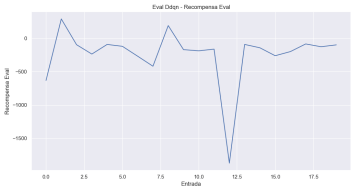


Figura 37: Recompensas - evaluación (DDQN).



Estas gráficas muestran un rendimiento errático. Aunque en algunos episodios el agente logra sobrevivir un número considerable de pasos y alcanzar algunas metas (con picos de hasta 3 metas en un episodio), la recompensa es generalmente baja o negativa. La tasa de éxito es nula y las colisiones/fallos son constantes en 1.0 por episodio, indicando que ningún episodio se completó sin incidentes graves.

6.1.2. PPO

Figura 38: Colisiones - evaluación (PPO).

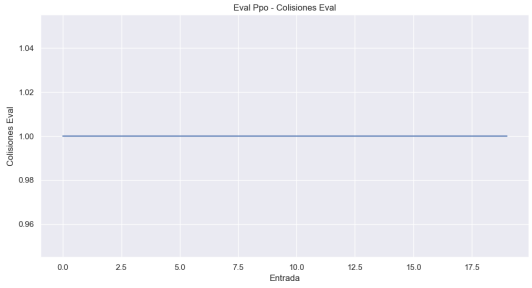


Figura 39: Tasa de éxito - evaluación (PPO).

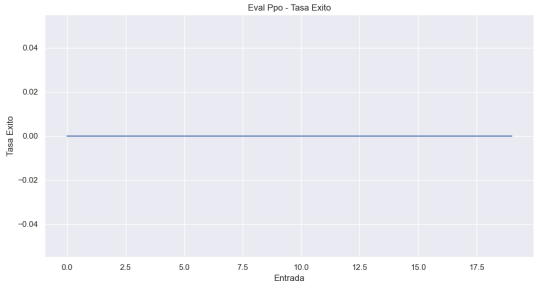


Figura 40: Metas alcanzadas - evaluación (PPO).

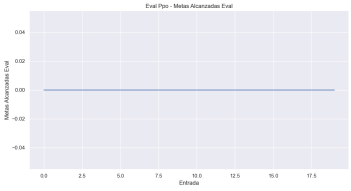


Figura 41: Pasos - evaluación (PPO).

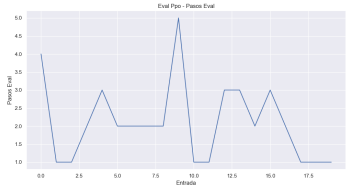
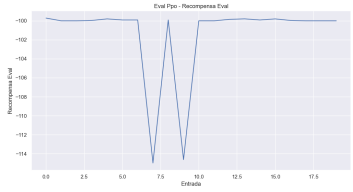


Figura 42: Recompensas - evaluación (PPO).



Como se observa, el rendimiento de PPO en evaluación es consistentemente bajo, confirmando el colapso de la política durante el entrenamiento. Los episodios terminan casi inmediatamente.

6.1.3. SAC

Figura 43: Colisiones - evaluación (SAC).

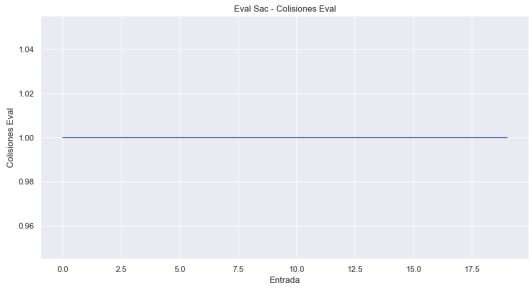


Figura 44: Tasa de éxito - evaluación (SAC).

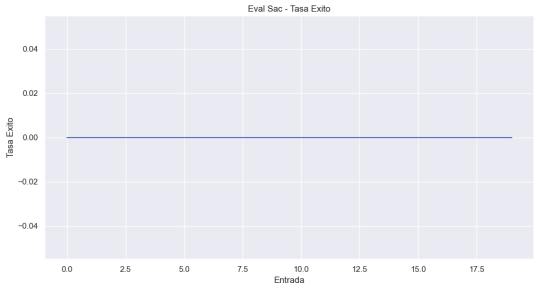


Figura 45: Metas alcanzadas - evaluación (SAC).

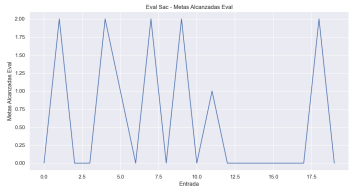


Figura 46: Pasos - evaluación (SAC).

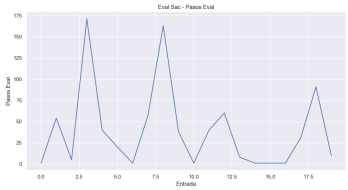
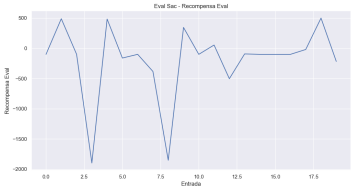


Figura 47: Recompensas - evaluación (SAC).



Las gráficas de SAC muestran el rendimiento más prometedor entre los algoritmos evaluados, aunque con una notable variabilidad. Se observan episodios con recompensas significativamente positivas (picos > 500) y alcanzando 2 metas en un número considerable de episodios. La duración de los episodios también es mayor en promedio comparado con DDQN y PPO. No obstante, al igual que los otros agentes, la tasa de éxito es nula y los fallos son constantes, indicando que, a pesar de un mejor desempeño en métricas de progreso, la seguridad a largo plazo y la finalización exitosa de rutas completas siguen siendo un desafío.

6.2. Comparativa

Figura 48: Metas alcanzadas - evaluación.

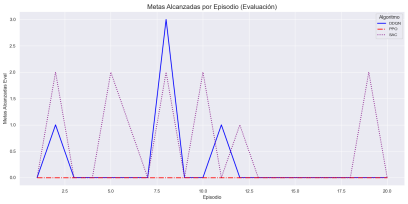


Figura 49: Pasos - evaluación.

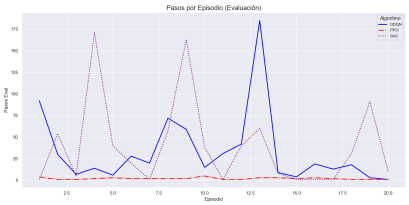


Figura 50: Recompensas - evaluación.

6.3. Tabla resumen de métricas de evaluación

La siguiente tabla resume las métricas promedio obtenidas durante la fase de evaluación para cada algoritmo.

Cuadro 3: Resumen de métricas promedio durante la evaluación (20 episodios)

Métrica	DDQN	PPO	SAC	A3C
Recompensa Prom.	**-271,22**	**-101,42**	**-217,16**	N/A
Desv. Est. Recompensa	**484,09**	**5,18**	**700,71**	N/A
Pasos Prom.	**29,95**	**2,15**	**44,70**	N/A
Metas Prom.	**0,25**	**0,00**	**0,50**	N/A
Tasa Éxito (%)	**0,00**	**0,00**	**0,00**	N/A
Colisiones/Fallos Prom.	**1,00**	**1,00**	**1,00**	N/A

Nota: La Tasa de Éxito del 0 % y Colisiones/Fallos Prom. de 1.0 indican que todos los episodios terminaron en fallo.

La métrica *Colisiones/Fallos Prom.* muestra un valor de 1.0 para todos los algoritmos. Observando los gráficos de colisiones, se ve una línea plana en 1.0. Esto, combinado con una "Tasa Éxito"del 0.0 %, sugiere que todos los episodios de evaluación terminaron prematuramente debido a algún tipo de fallo (colisión detectada por SUMO, colisión forzada por acción inválida, o posiblemente teleportación u otro error no explícitamente separado en esa métrica). Por lo tanto, aunque SAC y DDQN muestran cierta capacidad de alcanzar metas y sobrevivir más pasos en algunos episodios, ninguno demuestra un comportamiento consistentemente seguro o exitoso en esta configuración.

6.4. Análisis comparativo

Basándonos en los resultados de la evaluación (Tabla 3) y recordando el proceso de entrenamiento (Sección 5) podemos sacar las siguientes conclusiones:

- **¿Qué algoritmo aprendió más rápido?**
  Durante la fase de entrenamiento, **SAC** (Figura 27, Figura 29) mostró la curva de aprendizaje inicial más prometedora y una recuperación más robusta en la fase final, superando a DDQN en la velocidad para alcanzar recompensas positivas y un alto número de metas promedio. DDQN (Figura 12, Figura 14) también mostró aprendizaje, pero más lento y con mayor inestabilidad. PPO (Figura 18, Figura 20) colapsó y no demostró aprendizaje útil sostenido.

- **¿Cuál alcanzó el mejor rendimiento promedio final (Eval)?**
  En términos de recompensa promedio durante la evaluación, **SAC** obtuvo el un mejor equilibrio entre la recompensa y el número de metas alcanzando, aunque con una gran variabilidad. DDQN tuvo un rendimiento promedio algo menor y también alta con variabilidad. PPO tuvo un rendimiento consistentemente bajo con varianza casi nula.

- **¿Cuál fue más estable/consistente en evaluación?**
  Si la consistencia se mide por baja varianza, PPO fue el más consistente (5.18), pero en un estado de fallo inmediato (2.15 pasos y -101.42 recompensa promedios). Entre los que mostraron alguna capacidad, ninguno fue estable. Tanto DDQN como SAC tuvieron desviaciones estándar muy altas en la recompensa (484.09 y 700.71 respectivamente), reflejando episodios muy buenos mezclados con otros muy malos. Los gráficos de evaluación por episodio (Figura 37 y Figura 47) confirman esta alta variabilidad para DDQN y SAC.

- **¿Cuál logró completar más metas (Eval)?**
  SAC fue claramente superior en este aspecto, logrando un promedio de 1.25 metas por episodio en evaluación (Figura 45), con picos de hasta 7 metas en un solo episodio. DDQN logró muy pocas metas (promedio 0.15, Figura 35). PPO no logró ninguna meta en evaluación (Figura 40).

- **¿Cuál tuvo el comportamiento más seguro (menos colisiones/fallos) (Eval)?**
  Según la interpretación de los datos (Tasa Éxito 0 % y Fallos Promedio 1.0 para todos), ninguno de los algoritmos demostró un comportamiento seguro en la evaluación. Todos los episodios terminaron en algún tipo de fallo. Aunque SAC y DDQN sobrevivieron más pasos en promedio (44.70 y 29.95 respectivamente) que PPO (2.15), esto no se tradujo en episodios completados sin incidentes graves.

6.5. Discusión de las diferencias observadas

Las diferencias significativas en el rendimiento de evaluación entre los algoritmos pueden atribuirse a una combinación de sus características intrínsecas y los desafíos encontrados durante

el entrenamiento en este entorno específico:

- **PPO (Fracaso total)**: El colapso observado durante el entrenamiento (Subsección 5.3) resultó en una política final inútil. El agente PPO evaluado simplemente ejecutaba 1 o 2 pasos y fallaba consistentemente. Esto subraya la sensibilidad de PPO a los hiperparámetros, la función de recompensa y potencialmente la **dificultad de los algoritmos on-policy** para recuperarse de fases de exploración muy negativas o divergencias en entornos con fallos frecuentes y recompensas posiblemente sparse o conflictivas.
- **DDQN (Aprendizaje parcial e inestable)**: Como algoritmo off-policy basado en valor, DDQN pudo aprender estrategias iniciales para evitar los peores resultados y ocasionalmente alcanzar metas (aprendizaje visible en Figura 12 y Figura 14). Sin embargo, tiene una gran potencial inestabilidad, especialmente con aproximadores de función (redes neuronales) y espacios de estados complejos. La alta varianza en recompensa y pasos durante la evaluación, junto con la incapacidad de converger a una política estable en el entrenamiento, reflejan estas limitaciones. La separación Dueling y el Deep Q-learning (DDQN) mitigan, pero no eliminan por completo, problemas como la **sobreestimación de valores Q**.
- **SAC (Mejor rendimiento, pero inestable e inseguro)**: SAC, siendo off-policy y maximizando la entropía, demuestra la mejor capacidad de exploración y aprendizaje inicial. La maximización de entropía le ayuda a encontrar políticas más diversas y a evitar quedarse atascado en mínimos locales tan fácilmente como DDQN o PPO en la fase inicial. Esto se tradujo en la mayor recompensa promedio, el mayor número de pasos promedio y la mayor cantidad de metas alcanzadas en evaluación. Sin embargo, la extrema varianza en la evaluación y la degradación del rendimiento observada al final del entrenamiento (Figura 27) indican que tampoco alcanzó una política robusta. Las posibles causas incluyen:
  - **Errores acumulados en el buffer**: Al ser off-policy, puede aprender de datos antiguos que ya no son representativos de la política actual, especialmente si la política cambia drásticamente.
  - **Ajuste de alfa**: Aunque el ajuste automático de alpha (temperatura de entropía) es una ventaja, puede volverse inestable o converger a un valor subóptimo en ciertas fases, afectando el equilibrio exploración/explotación.
  - **Complejidad del entorno**: El entorno en sí, con rutas dinámicas cortas y la posibilidad constante de fallos, puede ser intrínsecamente difícil para lograr estabilidad a largo plazo.

- **Falta universal de seguridad/exito**: El hecho de que ningún algoritmo lograra una tasa de éxito superior al 0 % en evaluación es la conclusión más preocupante. Indica que, a pesar de los esfuerzos en el diseño de la recompensa y la exploración de diferentes algoritmos, la configuración actual (entorno + estado + acción + recompensa + algoritmos/hiperparámetros) no fue suficiente para producir un agente de conducción autónoma mínimamente fiable en el largo plazo. Los fallos constantes (colisiones, teleportaciones o acciones inválidas) sugieren que los agentes no aprendieron a manejar consistentemente situaciones de riesgo o a planificar de forma segura, incluso en los episodios donde lograban avanzar y cumplir metas parciales. Esto podría deberse a:

  - Representación de estado insuficiente (falta de información predictiva o de planificación a más largo plazo).
  - Función de recompensa que, aunque mejorada, todavía no penaliza suficientemente los comportamientos sutilmente arriesgados o no recompensa adecuadamente la conducción defensiva y fluida a largo plazo.
  - Exploración insuficiente o ineficaz para descubrir políticas verdaderamente seguras.
  - Complejidad inherente del entorno SUMO simulado.

En resumen, la evaluación estandarizada confirma que **SAC** es el algoritmo con mayor potencial entre los probados, mostrando una mejor capacidad para seguir rutas y alcanzar metas. Sin embargo, la falta total de episodios completados sin fallos en todos los algoritmos subraya que aún existen desafíos significativos en términos de seguridad, robustez y la capacidad de los agentes para generalizar a trayectorias más largas y complejas. **DDQN** muestra un aprendizaje limitado y también inestable. **PPO** no logra aprender una política útil en esta configuración. La falta de éxito (alcanzar los $max\_steps = 2000$ pasos sin colisiones) generalizado en términos de seguridad resalta los desafíos pendientes en este proyecto.

Si bien este estudio ha permitido establecer una base sólida, identificar las fortalezas y debilidades relativas de diferentes enfoques de DRL, y, crucialmente, ha puesto de manifiesto áreas clave para futuras mejoras. La capacidad de SAC para adaptarse y progresar, incluso con las dificultades presentes, sugiere que con un refinamiento continuo de la función de recompensa, una representación del estado aún más informativa y una exploración más profunda de los hiperparámetros, es plausible alcanzar niveles de rendimiento significativamente más altos y robustos. Los desafíos pendientes, por lo tanto, no representan un punto final, sino una hoja de ruta clara para la optimización y el desarrollo futuro en este prometedor campo de la conducción autónoma inteligente.

7. Análisis de problemas comunes y soluciones intentadas

Durante la fase de experimentación con los diferentes algoritmos de DRL, han surgido varios desafíos recurrentes que han afectado significativamente al rendimiento y capacidad de aprendizaje de los agentes. Esta sección analiza estos problemas clave y las distintas estrategias y ajustes que se han implementado en un intento por mitigarlos.

7.1. Diseño de la función de recompensa (iteraciones y efectos)

La definición de una función de recompensa adecuada demostró ser uno de los aspectos más críticos y desafiantes. Una recompensa mal diseñada puede guiar al agente hacia óptimos locales no deseados o impedir completamente el aprendizaje. En esta sección se explica cuál ha sido la evolución del sistema de recompensas hasta llegar a la versión actual (ver DEFAULT_REWARD_WEIGHTS).

- **Desbalance inicial entre penalizaciones y recompensas**: Las primeras configuraciones presentaban penalizaciones muy altas por eventos negativos (colisiones, teleportaciones) y recompensas positivas muy bajas por comportamientos deseados como avanzar en dirección al objetivo (*approaching_goal*). Esto se manifestó en unos resultados iniciales donde los agentes rápidamente aprendían a minimizar la penalización fallando lo antes posible (episodios muy cortos, recompensas consistentemente negativas). El incentivo para explorar y encontrar secuencias de acciones largas y potencialmente recompensantes era prácticamente inexistente.

- **Iteraciones en los pesos**: Se han realizado varios ajustes a los pesos en *DEFAULT_REWARD_WEIGHTS*:

  - Se incrementó significativamente la recompensa por progreso hacia la meta (approaching_goal), utilizada una fórmula basada en la diferencia entre las distancias al objetivo del paso actual y el paso anterior.
  - Se redujo la penalización por paso (step_reward), creada inicialmente para fomentar llegar a la meta en el menor número de pasos posibles, para no desincentivar episodios largos, buscando alcanzar varias metas.
  - Se redujeron ligeramente algunas penalizaciones como emergency_stop y keep_lane.

- **Efectos observados**: Estos ajustes, particularmente el aumento de la recompensa por progreso/distancia y la eliminación de la penalización por paso, permitieron que agentes

como DDQN (ver Figura 14) y SAC (ver Figura 29) mostraran signos claros de aprendizaje, aumentando la duración de los episodios y el número de metas alcanzadas. Sin embargo, el balance óptimo seguía siendo difícil de encontrar, como lo demuestran las oscilaciones en DDQN y la inestabilidad tardía en SAC. PPO, incluso con los ajustes, siguió mostrando problemas graves (Figura 20).

7.2. Representación del estado

La forma en que se representa el estado del entorno para el agente es fundamental para el aprendizaje de los diferentes agentes. El diseño del vector de estado busca un equilibrio entre proporcionar información suficientemente rica para una toma de decisiones compleja y mantener un espacio de estados manejable para los algoritmos de DRL.

- **Suficiencia y planificación a corto/medio plazo**: El estado de **59** dimensiones implementado captura información local importante (posición y velocidad del ego-vehículo, información del carril actual, vehículos cercanos, próximos giros posibles en intersecciones, y estado del semáforo más cercano). Crucialmente, con las últimas mejoras, **se ha incorporado información sobre el siguiente segmento de la ruta asignada al vehículo**. Esto incluye la posición relativa normalizada del punto final de este siguiente *edge* y su longitud normalizada (ver 4.1.1 para detalles). Esta adición proporciona al agente una capacidad de **planificación a medio plazo**, permitiéndole anticipar la geometría y escala del tramo inmediatamente posterior a aquel en el que se encuentra.

  A pesar de esta mejora, el estado aún podría considerarse que carece de información de planificación a *muy largo plazo* (como la ruta completa hasta el destino final del viaje o la distancia total restante), la cual podría ser beneficiosa para estrategias más globales, especialmente en escenarios mucho más extensos. No obstante, la información del siguiente *edge* ya representa un avance significativo para una mejor estimación de la función de valor y la toma de decisiones.

- **Normalización**: Se aplica normalización a la mayoría de las características continuas (velocidad, límite de velocidad, coordenadas relativas, distancias), dividiendo por valores máximos esperados o el radio de detección. Una normalización inadecuada o inconsistente podría haber contribuido a la inestabilidad observada, especialmente en la pérdida de valor (Value Loss) de PPO.

- **Padding/Truncamiento de vehículos cercanos**: El uso de un tamaño fijo (8) para los vehículos cercanos, rellenando con valores placeholder ([0, 0, 0, 0, 2.0]) o truncando si hay más, es una simplificación necesaria para las redes neuronales utilizadas. Sin embargo,

introduce limitaciones: pérdida de información si hay más de 8 vehículos relevantes y la necesidad de que la red aprenda a ignorar el padding, lo cual puede ser ineficiente. El valor 2.0 como distancia placeholder podría ser ambiguo. Se consideraron alternativas como usar valores fuera de rango (-1) o máscaras, pero no se implementaron.

7.3. Estabilidad y convergencia algorítmica

Cada algoritmo presenta sus propios desafíos de estabilidad.

- **DDQN**: Muestra aprendizaje inicial pero luego entró en fuertes oscilaciones (Figura 12), típico de algoritmos Q-learning donde las estimaciones de valor pueden fluctuar. Los ajustes en *target_update* y *epsilon_decay* intentaron mitigar esto, con éxito parcial.
- **A3C**: La principal barrera es la inestabilidad a nivel de sistema al intentar lanzar múltiples instancias de SUMO/TraCI concurrentemente desde hilos, impidiendo evaluar su convergencia algorítmica.
- **PPO**: Sufrió una gran divergencia inicial, marcada por la explosión de la *Value Loss* (Figura 21). Aunque los ajustes en *n_steps*, *n_epochs*, *lr* y *entropy_coef*, junto con el *clipping* de gradientes, evitaron la divergencia completa en ejecuciones posteriores, la política colapsó rápidamente a un mínimo local subóptimo (entropía cero, cero metas en Figura 20). Esto sugiere una alta sensibilidad a los datos iniciales (posiblemente muy negativos o ruidosos) y/o una dificultad intrínseca del algoritmo on-policy para explorar eficazmente en este entorno con fallos frecuentes.
- **SAC**: Demuestra la mejor capacidad de exploración inicial y convergencia hacia recompensas positivas (Figura 27), probablemente gracias a la maximización de entropía. Sin embargo, mostró inestabilidad a largo plazo, con un declive en el rendimiento y picos negativos extremos en la recompensa por episodio. Esto podría deberse a la deriva de la política off-policy acumulando errores o a una mala sintonización del ajuste automático de alfa.

7.4. Interacción con el entorno SUMO

La propia dinámica del entorno simulado y la interacción con TraCI presentaron desafíos.

- **Rutas dinámicas**: La asignación de una nueva ruta corta ([actual, siguiente]) al alcanzar una meta funcionó, pero la detección del "final"del *edge* (dist_to_end $< 2{,}0$) combinada con la posibilidad de que el siguiente *edge* fuera muy corto, llevó inicialmente a recompensas

goal_reached múltiples e indeseadas en pasos consecutivos. Esto se corrigió rastreando si la meta del segmento ya había sido recompensada.

- **Fallo en asignación de ruta**: En ocasiones, _assign_new_route no encontraba un siguiente *edge* válido (ej., callejones sin salida). Inicialmente, esto podía dejar al agente atascado; se modificó para que en ese caso se marcara el episodio como terminal (info["terminal"] = "goal_reached_no_new_route").

- **Estado _get_state y Errores TraCI**: Se encontraron y corrigieron errores IndexError al acceder a datos de traci.lane.getLinks y ValueError por inconsistencias en la dimensión del estado devuelto (especialmente en estados terminales o de error), haciendo más robusta la obtención del estado.

- **Inicio Concurrente (A3C)**: Como se detalló en Subsección 5.2, el inicio concurrente de SUMO vía traci.start desde hilos resultó ser el principal obstáculo técnico irresoluble con esa aproximación.

8. Conclusiones y trabajo futuro

Este Trabajo Final de Máster se embarcó en la exploración de técnicas avanzadas de Aprendizaje por Refuerzo Profundo (DRL) para la tarea de conducción autónoma en un entorno simulado utilizando SUMO. A lo largo del proyecto, se ha diseñado e implementado un sistema completo, desde la creación del entorno de simulación y la interfaz con SUMO, hasta la implementación, entrenamiento y evaluación de diversos algoritmos de DRL.

8.1. Conclusiones

Los experimentos realizados han proporcionado información valiosa sobre la aplicabilidad y los desafíos de los algoritmos DDQN, PPO y SAC en el contexto de la conducción autónoma simulada.

1. **Desarrollo de un entorno de simulación funcional:** Se ha logrado crear con éxito un entorno en Python que interactúa con SUMO vía TraCI, permitiendo controlar un vehículo, asignarle rutas dinámicas, extraer un vector de estado de 59 dimensiones y aplicar una función de recompensa modular. Este entorno constituye una base sólida para la experimentación futura.

2. **Rendimiento diferencial de los algoritmos:**

   - El algoritmo **Soft Actor-Critic (SAC)** demostró ser el más prometedor, exhibiendo la capacidad de aprender políticas que le permitieron navegar durante más tiempo, alcanzar un mayor número de metas de segmento y obtener recompensas promedio más altas en la evaluación estandarizada en comparación con los otros agentes. Su mecanismo de maximización de entropía pareció facilitar una exploración inicial más efectiva.
   - **Dueling Double Deep Q-Network (DDQN)** mostró indicios de aprendizaje, logrando evitar los peores resultados iniciales y completar algunos segmentos de ruta. Sin embargo, su rendimiento fue consistentemente inferior al de SAC pese a entrenar durante más episodios y sufrió de una notable inestabilidad tanto en entrenamiento como en evaluación.
   - **Proximal Policy Optimization (PPO)**, a pesar de los ajustes en hiperparámetros, no logró desarrollar una política funcional en este entorno, colapsando a un rendimiento mínimo tras una fase inicial de divergencia. Esto subraya la sensibilidad de los algoritmos *on-policy* a la configuración y la complejidad del entorno.

- La implementación de **Asynchronous Advantage Actor-Critic (A3C)** enfrentó obstáculos técnicos irresolubles con la gestión concurrente de TraCI desde hilos de Python, impidiendo su evaluación completa.

3. **Importancia de la representación del estado y la función de recompensa:** La iteración en el diseño del vector de estado, incluyendo información sobre el siguiente *objetivo* de la ruta, y los ajustes en la función de recompensa, fueron cruciales para observar mejoras en el aprendizaje, especialmente en SAC y DDQN. No obstante, la persistencia de fallos en todos los algoritmos durante la evaluación indica que estos componentes aún pueden ser optimizados.

4. **Desafíos en seguridad y robustez:** A pesar de los avances, ninguno de los agentes logró completar consistentemente las episodios de evaluación completos (2000 pasos) sin incurrir en fallos (colisiones, errores de ruta, etc.) o quedarse estancados en un punto fijo. La tasa de éxito del 0 % en la finalización de episodios sin incidentes graves resalta que la robustez y la seguridad siguen siendo los mayores desafíos pendientes.

5. **Impacto ambiental cuantificado:** La utilización de CodeCarbon permitió estimar la huella de carbono del proceso de entrenamiento, revelando que la duración total del cómputo es el factor principal. Aunque modesta en este TFM, esta cuantificación subraya la importancia de la eficiencia algorítmica y la consideración de prácticas de "IA Verde".

8.2. Evaluación crítica de objetivos y metodología

El objetivo principal del TFM, desarrollar e implementar un entorno de simulación para entrenar y comparar agentes de DRL para conducción autónoma, se ha cumplido en gran medida. Se logró crear el entorno, implementar múltiples agentes y realizar un ciclo de entrenamiento y evaluación.

Los objetivos secundarios se alcanzaron parcialmente:

- *Definir y desarrollar el entorno de simulación:* Completado con éxito, incluyendo la interfaz con SUMO y la gestión de rutas dinámicas/fijas.
- *Diseñar la función de recompensa:* Se diseñó e iteró, logrando guiar el aprendizaje, aunque se reconoce que puede mejorarse aún más.
- *Implementar el cambio dinámico de rutas:* Funcional para entrenamiento y evaluación.
- *Recopilar y procesar el estado del vehículo:* Implementado con un vector de 59 dimensiones.

- *Evaluar y comparar distintos algoritmos de RL:* Realizado para DDQN, PPO y SAC. A3C no pudo ser evaluado completamente.
- *Lograr una conducción autónoma segura y eficiente:* Este objetivo final no se alcanzó plenamente, ya que ningún agente demostró ser consistentemente seguro.

La planificación inicial (Diagrama de Gantt, Figura 2) fue una guía útil, aunque la fase de depuración de algoritmos y la resolución de problemas con A3C consumieron más tiempo del previsto. La metodología de investigación bibliográfica, desarrollo iterativo del entorno y los agentes, y posterior experimentación fue adecuada. La decisión de utilizar SUMO fue acertada por su flexibilidad y realismo.

8.3. Desafíos de sostenibilidad, diversidad y etico/sociales

Desde la perspectiva de la **sostenibilidad**, este proyecto ha tocado dos puntos:

1. El objetivo a largo plazo de la conducción autónoma es optimizar flujos de tráfico y reducir el consumo de combustible así como el resto de beneficios sociales, contribuyendo positivamente.
2. El entrenamiento de modelos de DRL, como se cuantificó con CodeCarbon, tiene un coste energético. Este TFM buscó ser consciente de ello y sienta las bases para trabajos futuros que busquen mayor eficiencia computacional.

En cuanto a **diversidad y derechos humanos**, el proyecto en sí es técnico. Sin embargo, una aplicación exitosa de la conducción autónoma podría mejorar la accesibilidad al transporte para personas con movilidad reducida. Es crucial que futuros desarrollos aseguren que los sistemas sean equitativos y no perpetúen sesgos. Los **desafíos ético-sociales** de la conducción autónoma (dilemas en caso de accidente, responsabilidad, privacidad de datos) están fuera del alcance técnico de este TFM, pero son consideraciones fundamentales para la industria. Este trabajo se ha centrado en un entorno simulado, minimizando riesgos directos.

8.4. Trabajo futuro

Este TFM ha sentado una base experimental robusta, y los resultados obtenidos abren numerosas vías para investigaciones futuras:

1. **Refinamiento de la función de recompensa:** Explorar funciones de recompensa más sofisticadas que penalicen comportamientos de riesgo sutiles (ej., acercarse demasiado, frenadas bruscas innecesarias) y recompensen de forma más explícita la conducción fluida,

defensiva y el cumplimiento de objetivos a más largo plazo. Se podrían investigar técnicas de **reward shaping** [38, 39].

2. **Mejora de la representación del estado:**
   - Incorporar información de planificación a muy largo plazo.
   - Experimentar con diferentes formas de codificar la información de vehículos cercanos, como el uso de mecanismos de atención o representaciones basadas en grafos.
   - Investigar el impacto de añadir predicciones de trayectoria de otros vehículos.
3. **Optimización Exhaustiva de Hiperparámetros para SAC:** Dado el rendimiento superior pero aún inestable de SAC, una línea prioritaria de trabajo futuro es realizar una búsqueda sistemática y exhaustiva de sus hiperparámetros. Esto incluiría:
   - Tasas de aprendizaje para el actor, los críticos y el parámetro de entropía alfa.
   - El factor de descuento gamma (γ).
   - El coeficiente de actualización suave tau (τ).
   - El valor inicial de alfa y la estrategia de aprendizaje de alfa (si se mantiene su ajuste automático).
   - Tamaño del buffer de repetición y tamaño del batch.
   - Arquitectura de las redes neuronales (número de capas, neuronas por capa, funciones de activación).

   Se podrían emplear herramientas de optimización automática de hiperparámetros como **Optuna** [40] o **Ray Tune** [41], evaluando el rendimiento en el conjunto de rutas de evaluación estandarizadas para encontrar la configuración que maximice la recompensa promedio, la tasa de éxito (finalización segura de rutas) y la estabilidad.

En conclusión, si bien la meta de una conducción autónoma completamente robusta y segura en el entorno simulado no se ha alcanzado en su totalidad, este trabajo ha demostrado el potencial de los algoritmos de DRL, especialmente SAC, y ha proporcionado una plataforma y una serie de aprendizajes valiosos que servirán de trampolín para futuras investigaciones y desarrollos en este apasionante campo. El camino hacia la autonomía inteligente es iterativo, y cada experimento y desafío superado nos acerca un paso más.

9. Glosario

A2C (Advantage Actor-Critic) Algoritmo de Aprendizaje por Refuerzo de tipo Actor-Crítico que utiliza la función de ventaja para guiar el aprendizaje de la política. Es una versión síncrona de A3C.

A3C (Asynchronous Advantage Actor-Critic) Algoritmo de Aprendizaje por Refuerzo de tipo Actor-Crítico que utiliza múltiples agentes (workers) entrenando en paralelo de forma asíncrona para actualizar una política global.

Acción (Action) Decisión que toma el agente en un estado particular del entorno. En este TFM, corresponde a maniobras de conducción como acelerar, frenar o cambiar de carril.

Actor-Crítico (Actor-Critic) Familia de algoritmos de Aprendizaje por Refuerzo que combinan dos componentes: un .ªctor"que aprende la política (qué acciones tomar) y un çrítico"que aprende una función de valor (qué tan buenos son los estados o las acciones).

Agente (Agent) Entidad que aprende a tomar decisiones interactuando con un entorno para maximizar una señal de recompensa acumulada. En este TFM, el agente es el vehículo autónomo controlado.

Aprendizaje por Refuerzo (Reinforcement Learning - RL) Área del aprendizaje automático donde un agente aprende a tomar una secuencia de decisiones interactuando con un entorno para maximizar una noción de recompensa acumulada a largo plazo.

Aprendizaje por Refuerzo Profundo (Deep Reinforcement Learning - DRL) Subcampo del Aprendizaje por Refuerzo que utiliza redes neuronales profundas como aproximadores de funciones para la política, la función de valor, o ambas, permitiendo manejar espacios de estado y acción complejos.

Buffer de Repetición (Replay Buffer / Experience Replay) Mecanismo utilizado en algoritmos RL *off-policy* (como DQN o SAC) que almacena las transiciones (estado, acción, recompensa, siguiente estado) pasadas. El agente muestrea aleatoriamente de este buffer para romper correlaciones temporales y mejorar la eficiencia de los datos.

CodeCarbon Librería de Python utilizada para estimar las emisiones de dióxido de carbono (CO_2eq) generadas por la ejecución de código, basándose en el consumo energético del hardware.

Conducción Autónoma (Autonomous Driving) Capacidad de un vehículo para operar y navegar sin intervención humana, utilizando sistemas de percepción, planificación y control.

DDQN (Dueling Double Deep Q-Network) Algoritmo de Aprendizaje por Refuerzo basado en valor que mejora Deep Q-Network (DQN) utilizando dos innovaciones: la arquitectura *Dueling Network* (que separa la estimación del valor del estado y la ventaja de las acciones) y *Double Q-learning* (para reducir la sobreestimación de los valores Q).

Descuento (Factor de Descuento - γ) Hiperparámetro en RL (generalmente entre 0 y 1) que determina la importancia de las recompensas futuras. Un valor cercano a 1 da más peso a las recompensas a largo plazo, mientras que un valor cercano a 0 prioriza las recompensas inmediatas.

DQN (Deep Q-Network) Algoritmo pionero en DRL que utiliza una red neuronal profunda para aproximar la función de valor-acción Q óptima.

Edge (SUMO) En SUMO, un segmento de carretera o calle que conecta dos *junctions* (intersecciones o nodos). Es la unidad básica de la red vial.

Entorno (Environment) En RL, el mundo con el que interactúa el agente. Proporciona estados al agente y devuelve recompensas y nuevos estados como respuesta a las acciones del agente. En este TFM, el entorno es la simulación de tráfico en SUMO.

Episodio (Episode) Una secuencia completa de interacciones del agente con el entorno, desde un estado inicial hasta un estado terminal (ej., colisión, alcanzar un objetivo, o un número máximo de pasos).

Espacio de Acciones (Action Space) Conjunto de todas las acciones posibles que el agente puede tomar. Puede ser discreto (un conjunto finito de acciones) o continuo.

Espacio de Estados (State Space) Conjunto de todas las posibles representaciones del entorno que el agente puede observar.

Estado (State) Una representación de la situación actual del entorno y del agente en un momento dado. En este TFM, es un vector numérico que describe la posición del vehículo, velocidad, vehículos cercanos, etc.

Exploración vs. Explotación (Exploration vs. Exploitation) Dilema fundamental en RL. La exploración implica probar nuevas acciones para descubrir mejores recompensas, mientras que la explotación implica tomar la acción que actualmente se cree que es la mejor.

Función de Recompensa (Reward Function) Define la señal escalar que el entorno proporciona al agente después de cada acción, indicando qué tan buena (o mala) fue esa acción en ese estado. El objetivo del agente es maximizar la recompensa acumulada.

Función de Valor (Value Function) Una función que estima la recompensa acumulada esperada a largo plazo desde un estado particular (V(s)) o desde un par estado-acción (Q(s,a)), asumiendo que el agente sigue una política particular.

Hiperparámetros (Hyperparameters) Parámetros del algoritmo de aprendizaje que no se aprenden directamente de los datos, sino que se configuran antes del entrenamiento (ej., tasa de aprendizaje, factor de descuento, tamaño del batch).

Junction (SUMO) En SUMO, un nodo de la red vial donde se conectan dos o más *edges*, típicamente una intersección.

Normalización (Normalization) Proceso de escalar los datos de entrada (como las características del estado) a un rango común (ej., [0,1] o [-1,1]) para mejorar la estabilidad y eficiencia del entrenamiento de las redes neuronales.

Off-policy Algoritmos de RL que pueden aprender una política objetivo (la que se quiere optimizar) utilizando datos generados por una política de comportamiento diferente (la que se usó para explorar). Ejemplos: DQN, SAC.

On-policy Algoritmos de RL que aprenden una política evaluando y mejorando la misma política que se está utilizando para tomar decisiones y generar datos. Ejemplos: SARSA, PPO (en su forma más común), A2C/A3C.

Optimización de Hiperparámetros (Hyperparameter Optimization - HPO) Proceso de encontrar el conjunto de hiperparámetros que resulta en el mejor rendimiento del modelo. Herramientas como Optuna o Ray Tune automatizan este proceso.

Optuna Librería de Python para la optimización de hiperparámetros que utiliza algoritmos de búsqueda eficientes como TPE.

Placeholder Valor utilizado para rellenar espacios en un vector de estado de tamaño fijo cuando la información real no está disponible o es menor a la esperada (ej., menos de 8 vehículos cercanos detectados).

Política (Policy - π) Estrategia que utiliza el agente para decidir qué acción tomar en cada estado. Puede ser determinista (siempre la misma acción para un estado) o estocástica (una distribución de probabilidad sobre las acciones).

PPO (Proximal Policy Optimization) Algoritmo de Aprendizaje por Refuerzo de tipo Actor-Crítico y *on-policy* que busca realizar actualizaciones de política estables utilizando un objetivo recortado"(clipped surrogate objective) para limitar el tamaño de los cambios en la política.

Ray Tune Librería de Ray para la optimización de hiperparámetros distribuida y escalable.

Ray RLlib Librería escalable de Aprendizaje por Refuerzo construida sobre Ray, que proporciona implementaciones de muchos algoritmos DRL.

Recompensa (Reward) Señal numérica que el agente recibe del entorno después de realizar una acción, indicando la bondad inmediata de esa acción.

Red Neuronal (Neural Network) Modelo computacional inspirado en la estructura del cerebro humano, compuesto por capas de nodos (neuronas) interconectados, utilizado en DRL para aproximar funciones complejas como la política o la función de valor.

SAC (Soft Actor-Critic) Algoritmo de Aprendizaje por Refuerzo de tipo Actor-Crítico y *off-policy* que busca maximizar tanto la recompensa esperada como la entropía de la política. Esto fomenta una mayor exploración y robustez.

SUMO (Simulation of Urban MObility) Simulador de tráfico microscópico, multimodal y de código abierto utilizado en este TFM como entorno para entrenar y evaluar los agentes de conducción autónoma.

Tasa de Aprendizaje (Learning Rate - α) Hiperparámetro que controla cuán grandes son los ajustes a los pesos de una red neuronal durante el entrenamiento.

TraCI (Traffic Control Interface) API de Python que permite la comunicación y el control en tiempo real de la simulación en SUMO.

Value Loss (Pérdida de Valor) Métrica utilizada durante el entrenamiento de algoritmos basados en valor o actor-crítico, que cuantifica el error en la predicción de la función de valor (o Q-valor).

Bibliografía

[1] Diaz V. Repositorio GitHub del proyecto. (2025). https://github.com/VictorDiazBustos/TFM_Explorando_tecnicas_avanzadas_de_RL_para_conduccion_autonoma

[2] López P. (2023). *"Aplicación de técnicas de aprendizaje por refuerzo profundo para conducción autónoma en el simulador CARLA"*. Trabajo de Fin de Master. Master de Ciencia de Datos de la *Universitat Oberta de Catalunya*.

[3] Sutton R. S. & Barto A. G. (2020). *Reinforcement Learning: An Introduction"* (2ª ed.). MIT Press. http://incompleteideas.net/book/RLbook2020.pdf

[4] Goodfellow I., Bengio Y. & Courville A. (2016). *"Deep Learning"*. MIT Press. https://www.deeplearningbook.org/

[5] Grigorescu M., Trasnea B., Cocias T. & Macesanu, G. (2020). *" A Survey of Deep Learning Techniques for Autonomous Driving"*. IEEE Access, 8. https://arxiv.org/pdf/1910.07738

[6] Schwall M., Daniel T., Victor T., Favarò F. & Hohnhold H.(2020). *"Waymo Public Road Safety Performance Data"*. https://arxiv.org/pdf/2011.00038

[7] Litton M. L., Drusinsky D. & Michael J. B. (2024). *Reliable Autonomous Vehicles: How Do We Get There?"*. In IEEE Reliability Magazine, vol. 1, no. 1 https://ieeexplore.ieee.org/abstract/document/10416807

[8] Rosenzweig J., Bartl M. (2015). *" A Review and Analysis of Literature on Autonomous Driving"*. https://michaelbartl.com/wp-content/uploads/Lit-Review-AD_MoI.pdf

[9] Kiran B. R., Sobh I., Talpaert V., Mannion P., Al Sallab A. A., Yogamani S. & Pérez P. (2020). *"Deep Reinforcement Learning for Autonomous Driving: A Survey"*. https://arxiv.org/pdf/2002.00444

[10] Pérez-Gil O. et al. (2022). *"Deep reinforcement learning based control for Autonomous Vehicles in CARLA"*. En: Multimedia Tools and Applications 81.3. https://doi.org/10.1007/s11042-021-11437-3

[11] Kendall A., Hawke J., Janz D., Mazur P., Reda D., Allen J.-M., Lam V.-D., Bewley A. & Shah A.(2018). *"Learning to Drive in a Day"*. https://arxiv.org/pdf/1807.00412

[12] Anisimov Y. O. & Katcai D. A. (2021). *"Deep reinforcement learning approach for autonomous driving"*. https://www.researchgate.net/publication/351755367_Deep_reinforcement_learning_approach_for_autonomous_driving

[13] Dosovitskiy A., Ros G., Codevilla F., Lopez A. & Koltun, V. (2017). *"CARLA: An open urban driving simulator"*. https://arxiv.org/pdf/1711.03938

[14] Sallab A. A., Abdou M., Perot E. & Yogamani S. (2017). *"Deep Reinforcement Learning Framework for Autonomous Driving"*. https://arxiv.org/pdf/1704.02532

[15] Oates B. J. (2006). *Researching Information Systems and Computing"*. Wiley. https://dokumen.pub/researching-information-systems-and-computing-978-1-4129-022 html

[16] Alvarez Lopez P., Behrisch M., Bieker-Walz L., Erdmann J., Flötteröd Y.P., Hilbrich R., Lücken L., Rummel J., Wagner P. & Wiebner E. (2018). *"Microscopic Traffic Simulation using SUMO"*. IEEE Intelligent Transportation Systems Conference (ITSC). Último acceso 24 mayo 2025. https://sumo.dlr.de/docs/

[17] Alvarez Lopez P., Behrisch M., Bieker-Walz L., Erdmann J., Flötteröd Y.P., Hilbrich R., Lücken L., Rummel J., Wagner P. & Wiebner E. (2025). *TraCI Documentation*. Último acceso 24 mayo 2025. https://sumo.dlr.de/docs/TraCI.html

[18] European Commission, High-Level Expert Group on Artificial Intelligence. (2020). "*Ethics Guidelines for Trustworthy AI"*. Último acceso 18 marzo 2025. https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

[19] Clifton J. & Laber E. (2020). *"Q-Learning: Theory and Applications"*. Annual Review of Statistics and Its Application, Volume 7, 279-301. https://www.annualreviews.org/content/journals/10.1146/annurev-statistics-031219-041220

[20] Alavizadeh H., Jang-Jaccard J. (2022). *"Deep Q-Learning Based Reinforcement Learning Approach for Network Intrusion Detection"*. Computers 2022, 11, 41. https://doi.org/10.3390/computers11030041

[21] Wang Z., Schaul T., Hessel M., van Hasselt H., Lanctot M. & de Freitas N. (2016). *"Dueling Network Architectures for Deep Reinforcement Learning"*. Proceedings of The 33rd International Conference on Machine Learning. https://doi.org/10.48550/arXiv.1511.06581

[22] Chen G., Sun J., Zeng, Q., Jing G. & Zhang Y. (2023). *"Joint Edge Computing and Caching Based on D3QN for the Internet of Vehicles"*. Electronics, 12(10), 2311. https://doi.org/10.3390/electronics12102311

[23] Noorani E. & Baras J. S. (2021) *Risk-sensitive REINFORCE: A Monte Carlo Policy Gradient Algorithm for Exponential Performance Criteria"*. 60th IEEE Conference on Decision and Control (CDC), Austin, TX, USA. https://doi.org/10.1109/CDC45484.2021.9683645

[24] Grondman I., Busoniu L., Lopes G. A. D. & Babuska R. (2012). "*A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients"*. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, nº 6. https://doi.org/10.1109/TSMCC.2012.2218595

[25] Paczolay G. & Harmati I. (2020). "*A New Advantage Actor-Critic Algorithm For Multi-Agent Environments"*. 23rd International Symposium on Measurement and Control in Robotics (ISMCR), Budapest, Hungary. https://doi.org/10.1109/ISMCR51255.2020.9263738

[26] Shen H., Zhang K., Hong M. & Chen T. (2023) *"Towards Understanding Asynchronous Advantage Actor-Critic: Convergence and Linear Speedup"* in IEEE Transactions on Signal Processing, vol. 71. https://doi.org/10.1109/TSP.2023.3268475

[27] Schulman J., Levine S., Abbeel P., Jordan M. & Moritz P. (2015). *"Trust Region Policy Optimization"*. Proceedings of the 32nd International Conference on Machine Learning. https://proceedings.mlr.press/v37/schulman15.html

[28] Schulman J., Wolski F., Dhariwal P., Radford A. & Klimov O. (2017). *"Proximal Policy Optimization Algorithms"*. arXiv:1707.06347 https://doi.org/10.48550/arXiv.1707.06347

[29] Tan H. (2021). *Reinforcement Learning with Deep Deterministic Policy Gradient"*. International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA), Xi'an, China. https://doi.org/10.1109/CAIBDA53561.2021.00025

[30] Mohammadpour S., Bengio E., Frejinger E. & Bacon P.L. (2024). *"Maximum entropy GFlowNets with soft Q-learning"*. Proceedings of The 27th International Conference on Artificial Intelligence and Statistics.
https://proceedings.mlr.press/v238/mohammadpour24a.html

[31] Haarnoja T., Zhou A., Hartikainen K., Tucker G., Ha S., Tan J., Kumar V., Zhu H., Gupta A., Abbeel P. & Levine S. (2018). *"Soft Actor-Critic Algorithms and Applications"*. arXiv:1812.05905 https://doi.org/10.48550/arXiv.1812.05905

[32] Haarnoja T., Zhou A., Abbeel P. & Levine S. (2018). *"Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor"*. Proceedings of the 35th International Conference on Machine Learning.
https://proceedings.mlr.press/v80/haarnoja18b

[33] Binding. Microsoft Learn. Último acceso 24 mayo 2025.
https://learn.microsoft.com/es-es/dotnet/api/system.windows.data.binding?view=windowsdesktop-8.0

[34] Multiprocessing. (2025). *Python Documentation*. Último acceso 1 mayo 2025.
https://docs.python.org/3/library/multiprocessing.html

[35] Liang E., Liaw R., Moritz P., Nishihara R., Fox R., Goldberg K., Gonzalez J.E., Jordan M.I. & Stoica I. (2018). "*RLlib: Abstractions for Distributed Reinforcement Learning*". arXiv:1712.09381 https://doi.org/10.48550/arXiv.1712.09381

[36] Schmidt V., Goyal K., Joshi A., Feld B., Conell L., Laskaris N., Blank D., Wilson L., Friedler S. & Luccioni S. (2021). *"CodeCarbon: Estimate and Track Carbon Emissions from Machine Learning Computing"*. Último acceso 24 mayo 2025.
https://mlco2.github.io/codecarbon/

[37] Lacoste A., Luccioni A., Schmidt V. & Dandres, T. (2019). *"Quantifying the Carbon Emissions of Machine Learning"*. arXiv:1910.09700. Workshop on Tackling Climate Change with Machine Learning, NeurIPS 2019. https://arxiv.org/abs/1910.09700

[38] Miller, T. (2023). *Reward shaping"*. Mastering Reinforcement Learning.
https://gibberblot.github.io/rl-notes/single-agent/reward-shaping.html

[39] Hu Y., Wang W., Jia H., Wang Y., Chen Y, Hao Y, Wu F., Fan C. (2020). *"Learning to Utilize Shaping Rewards: A New Approach of Reward Shaping"*. arXiv:2011.02669. https://arxiv.org/abs/2011.02669

[40] Akiba T., Sano S., Yanase T., Ohta T. & Koyama M. (2019). *"A Next-generation Hyperparameter Optimization Framework"*. In KDD. Último acceso 24 mayo 2025. https://optuna.org/

[41] Liaw R., Liang E., Nishihara R., Moritz P., Gonzalez J. & Stoica I.. (2018). *"Tune: A Research Platform for Distributed Model Selection and Training"*. Último acceso 24 mayo 2025. https://docs.ray.io/en/latest/tune/index.html