UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

simDeepPON-RL — Deep Reinforcement Learning para la Planificación y Operación Autónoma de Redes XGS-PON: Implementación e Integración con un Simulador en Python

Presentado por Santiago Rodríguez Diez
en Universidad de Burgos — 8 de julio de 2025
Tutor: Rubén Ruiz González
Co-tutora: Antonia Maiara Marques do Nascimento

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Rubén Ruiz González, profesor del departamento de Digitalización, área de Ingeniería de Sistemas y Automática.

D. Antonia Maiara Marques do Nascimento, profesora del departamento de Digitalización, área de Ciencia de la Computación e Inteligencia Artificial.

Exponen:

Que el alumno D. Santiago Rodríguez Diez, con DNI 71963036S, ha realizado el Trabajo Fin de Grado en Ingeniería Informática titulado "simDeepPON-RL — Deep Reinforcement Learning para la Planificación y Operación Autónoma de Redes XGS-PON: Implementación e Integración con un Simulador en Python".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de quienes suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 8 de julio de 2025

Vº. Bº. del Tutor:

D. Rubén Ruiz González

Vº. Bº. de la co-tutora:

Dña. Antonia Maiara Marques do Nascimento

Resumen

El Deep Reinforcement Learning es una técnica de inteligencia artificial que combina el **Aprendizaje por Refuerzo**, técnica que se basa en el entrenamiento de un agente mediante la interacción con el entorno, donde este toma decisiones y recibe una recompensa del entorno en función de sus acciones; con el **Aprendizaje Profundo**, técnica también muy importante en el ámbito de la inteligencia artificial y que utiliza redes neuronales multicapa para almacenar representaciones más complejas de los datos y así lograr un aprendizaje más efectivo.
Por otro lado, las **Redes Ópticas Pasivas** son un tipo particular de redes de telecomunicación, que utilizan fibra óptica y componentes pasivos para lograr conexiones de banda ancha entre un punto centralizado, denominado OLT (Optical Line Terminal) y numerosos usuarios finales, denominados ONT (Optical Network Terminal).

En este contexto, la **optimización del ancho de banda** asignado a los usuarios de este tipo de redes es un aspecto clave para lograr un uso eficiente de ellas y, por tanto, unas mejores prestaciones. Y el aprendizaje por refuerzo profundo resulta ser una técnica muy acertada para tratar este problema, dada la complejidad de los datos. Por ello, este trabajo se centra en la integración de un agente de aprendizaje por refuerzo profundo -**DRL** por sus siglas en inglés- con un simulador de redes ópticas pasivas -**PON** por sus siglas en inglés- con el objetivo de optimizar la asignación de recursos en tiempo real.

Partiendo de un agente DRL desarrollado anteriormente por un compañero de la UBU en **Python** mediante el uso de las librerías **Gymnasium** y **StableBaselines3**; y de un simulador de redes PON, desarrollado por otro compañero de la UVA también en **Python** y con la librería **Simpy**, se ha realizado la integración de estos dos elementos para su utilización conjunta.

Descriptores

Aprendizaje profundo, redes ópticas pasivas, aprendizaje por refuerzo, optimización dinámica, simulación de redes, integración de sistemas, telecomunicaciones, Python.

Abstract

Deep Reinforcement Learning (DRL) is an artificial intelligence technique that combines **Reinforcement Learning**—a method based on training an agent through interaction with an environment, where it makes decisions and receives rewards based on its actions—with **Deep Learning**, another key approach in the field of AI that uses multilayer neural networks to learn complex data representations and thus achieve more effective learning.
On the other hand, **Passive Optical Networks** (PON) are a specific type of telecommunications network that uses fiber optics and passive components to establish broadband connections between a centralized point, known as the OLT (Optical Line Terminal), and multiple end users, referred to as ONTs (Optical Network Terminals).

In this context, **optimizing the bandwidth** allocation among users in this type of network is a crucial aspect to ensure efficient usage and, consequently, better performance. Deep reinforcement learning emerges as a highly suitable technique for addressing this challenge, given the complexity and dynamics of the data involved. This project focuses on the integration of a DRL agent with a PON network simulator, with the goal of optimizing resource allocation in real time.

The starting point involved a DRL agent previously developed by a fellow student at the University of Burgos (UBU) using Python, built with the Gymnasium and Stable-Baselines3 libraries; and a PON simulator developed by another student from the University of Valladolid (UVA), also in Python and based on the SimPy library. These two components have been integrated into a unified system for joint operation.

Keywords

Deep learning, passive optical networks, reinforcement learning, dynamic optimization, network simulation, system integration, telecommunications, python.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Las **Redes Ópticas Pasivas** (PON) son una tecnología de telecomunicaciones ampliamente utilizada para proporcionar servicios de banda ancha mediante el uso de **fibra óptica** y **componentes pasivos** -como divisores ópticos-, que permiten conectar de forma eficiente un único terminal óptico (OLT) a múltiples unidades de red óptica (ONTs) en el lado del usuario. Uno de los principales retos en este tipo de redes es la **asignación dinámica** del ancho de banda, especialmente en escenarios donde la demanda de tráfico varía constantemente entre usuarios. En estos casos, un reparto estático o mal ajustado puede provocar congestión en algunos nodos, infrautilización en otros y, en última instancia, una degradación de la calidad del servicio.

Ante esta problemática, las técnicas de inteligencia artificial —y en particular el **Aprendizaje por Refuerzo Profundo** (Deep Reinforcement Learning, DRL)— se posicionan como una solución prometedora. El DRL permite entrenar agentes capaces de aprender **políticas de asignación** óptimas en función del estado actual de la red, adaptándose a condiciones cambiantes y tomando decisiones en tiempo real sin necesidad de una programación manual de reglas.

En este proyecto se ha abordado el diseño e implementación de un sistema inteligente para la optimización de recursos en redes PON, integrando un agente DRL con un simulador realista de este tipo de redes. Aunque como punto de partida se han utilizado dos **Trabajos de Fin de Grado previos** —uno centrado en el desarrollo de un agente DRL y otro en la implementación de un simulador PON—, el trabajo realizado va mucho más allá de una simple integración de sistemas existentes.
Uno de los aspectos más complejos y críticos del proyecto ha sido la **co-**

rrección profunda del agente DRL, ya que el modelo original no era funcional: **el agente no aprendía**, las predicciones generadas no influían de forma significativa en el entorno, y el comportamiento observado durante las simulaciones era erróneo o aleatorio. A pesar de que externamente el sistema parecía operar, internamente el modelo **no tomaba decisiones reales** ni seguía una política aprendida. Esto supuso una importante barrera técnica que requirió una revisión exhaustiva del funcionamiento interno del entorno, la redefinición completa del sistema de recompensas y el ajuste de los espacios de observación y acción.

También se ha **reestructurado** toda la **arquitectura** general del agente, dividiéndolo en módulos claramente diferenciados, mejorando su organización interna y facilitando su extensión y mantenimiento. El resultado final es una herramienta coherente, **modular** y completamente **funcional**, capaz de entrenar y evaluar agentes sobre entornos realistas, y preparada para servir como base en futuras investigaciones sobre la gestión dinámica e inteligente del ancho de banda en redes de acceso ópticas.

Como punto de partida se han utilizado, como se ha comentado anteriormente, los siguientes trabajos:

- **Simulador de redes EPON Simulador EPON** (Ethernet Passive Optical Network) desarrollado por **Víctor Herrezuelo Paredes**, de la Universidad de Valladolid. Se trata de un simulador de redes ópticas pasivas basadas en protocolos Ethernet, implementado en **Python** utilizando la librería **SimPy**, que permite modelar los procesos concurrentes de transmisión de tráfico que se dan en este tipo de redes [1].
- **Agente DRL Agente de Deep Reinforcement Learning** desarrollado por **David Pérez Moreno**, de la Universidad de Burgos. Es un agente de aprendizaje profundo implementado en **Python** mediante la librería **Stable-Baselines3**, que utiliza entornos definidos con la librería **Gymnasium** para simular una red PON simplificada. El objetivo del agente es aprender una política de asignación de ancho de banda en función del estado de las colas de cada ONT [2].

El objetivo principal de este proyecto es **unificar** ambos sistemas para construir una herramienta funcional que permita entrenar y evaluar agentes de aprendizaje por refuerzo profundo sobre un entorno de simulación realista de redes ópticas pasivas, facilitando así su aplicación en tareas de

investigación relacionadas con la optimización de recursos en redes de acceso.

El siguiente objetivo clave del proyecto es abordar y **corregir** los **problemas críticos** del agente DRL. En su estado original, el modelo no logra entrenarse de forma adecuada, lo que se traduce en predicciones erróneas o prácticamente nulas durante su interacción con el entorno. Este comportamiento compromete seriamente el funcionamiento del sistema, ya que impide que el agente tome decisiones fundamentadas en una política aprendida. En lugar de optimizar la asignación de ancho de banda, las acciones resultantes son arbitrarias y no aportan valor al proceso de simulación. Por ello, ha resultado fundamental identificar las causas del fallo —ya sea en la configuración del entorno, en la función de recompensa o en los hiperparámetros del modelo— y aplicar las modificaciones necesarias para garantizar un aprendizaje efectivo y coherente con los objetivos del sistema.

Por último, el tercer objetivo es el de **mejorar** el sistema a nivel de **diseño** y **arquitectura**. Además de realizar una **refactorización general** —eliminando variables innecesarias, limpiando fragmentos de código redundantes y optimizando determinadas estructuras—, es necesario diseñar e implementar una arquitectura modular y escalable para el agente DRL, puesto que el original carece de ella. Esta nueva arquitectura debe facilitar la separación clara de responsabilidades entre los distintos componentes del sistema, como la gestión de modelos, el entrenamiento, la evaluación, el almacenamiento de resultados y la interacción con el entorno.

La presente memoria se organiza en siete capítulos:

- **Capítulo 1: Introducción**. Se introduce el contexto general del proyecto, su origen y los trabajos previos en los que se basa.
- **Capítulo 2: Objetivos del proyecto**. Recoge los objetivos concretos del trabajo, tanto a nivel funcional como técnico.
- **Capítulo 3: Conceptos teóricos**. Presenta los conceptos teóricos necesarios para comprender los fundamentos del sistema, incluyendo nociones sobre redes PON, aprendizaje por refuerzo y aprendizaje profundo.
- **Capítulo 4: Técnicas y herramientas**. Describe las herramientas y tecnologías empleadas durante el desarrollo, como librerías de Python, entornos de desarrollo y plataformas utilizadas.

- **Capítulo 5: Aspectos relevantes del proyecto**. Expone los aspectos más relevantes del proceso de desarrollo e integración, detallando las decisiones técnicas adoptadas, los problemas encontrados y cómo se resolvieron.
- **Capítulo 6: Trabajos relacionados**. Se analizan los trabajos previos relacionados con el proyecto, describiendo las aportaciones de los desarrollos anteriores y cómo han sido reutilizados y ampliados en este trabajo.
- **Capítulo 7: Conclusiones y líneas de trabajo futuras**. Recoge las conclusiones generales del trabajo, así como posibles líneas de mejora y evolución futura del sistema desarrollado.

2. Objetivos del proyecto

El objetivo principal de este Trabajo Fin de Grado es planificar, diseñar e implementar la integración funcional entre un agente de **Aprendizaje por Refuerzo Profundo** (DRL, por sus siglas en inglés) y un simulador de **redes ópticas pasivas** (PON), desarrollados previamente por compañeros de otras universidades. Concretamente, el agente fue elaborado por el estudiante **David Pérez Moreno** en la **Universidad de Burgos**, mientras que el simulador fue desarrollado por **Víctor Herrezuelo Paredes** en la **Universidad de Valladolid**.

Esta integración tiene como objetivo final la creación de una herramienta software que permita **evaluar agentes de inteligencia artificial** en un entorno de red más realista, con el fin de mejorar dinámicamente la **asignación de ancho de banda** en redes PON. Gracias a esta herramienta, se pretende facilitar futuras **tareas de investigación** en el ámbito de las telecomunicaciones, explorando nuevas estrategias de gestión de tráfico basadas en aprendizaje automático.

Además, el proyecto persigue la mejora cualitativa del código fuente original, abordando la **refactorización** y **modularización**. Para ello, se han aplicado principios de diseño orientado a objetos, patrones de diseño como **Adapter** y **Strategy**, y buenas prácticas de programación, con el objetivo de mejorar la mantenibilidad, la legibilidad y la escalabilidad del sistema.

Se busca, en definitiva, que el resultado sea una solución robusta y reutilizable, que pueda servir como base para proyectos posteriores, tanto en entornos académicos como en posibles aplicaciones industriales o experimentales relacionadas con la optimización de redes de acceso.

3. Conceptos teóricos

3.1. Aprendizaje por Refuerzo Profundo

El **Aprendizaje por Refuerzo Profundo** o **DRL**, por sus siglas en inglés, es una técnica de **aprendizaje automático** basada en la utilización de forma integrada de dos técnicas de machine learning ya bastante conocidas y desarrolladas. Estas son el aprendizaje por refuerzo (*reinforcement learning*) y aprendizaje profundo (*deep learning*).

Aprendizaje por refuerzo

Paradigma del aprendizaje automático en el que un **agente** aprende a comportarse en un **entorno** a través de la exploración de **acciones** y la observación de sus consecuencias, buscando maximizar una **recompensa** acumulada.

A diferencia del aprendizaje supervisado, donde tenemos una evaluación binaria para cada entrada —es decir, el resultado es correcto o no lo es—, en el aprendizaje por refuerzo la retroalimentación es cuantitativa y no hay una única solución correcta; existen múltiples decisiones posibles, cada una con una recompensa más o menos favorable. Además, el aprendizaje por refuerzo se basa en la interacción continua entre cuatro elementos clave: el agente, el entorno, las acciones y las recompensas.

Para contextualizar mejor este enfoque dentro del aprendizaje automático, la Tabla 3.1 presenta una comparación entre los tres paradigmas principales: aprendizaje supervisado, no supervisado y por refuerzo. Se destacan sus diferencias fundamentales en cuanto al tipo de datos necesarios, el objeti-

vo de aprendizaje y la forma en la que se recibe retroalimentación del entorno.

Tipo aprendizaje	**Etiquetas necesarias**	**Objetivo principal**	**Feedback**
Supervisado	Sí	Predecir salidas a partir de entradas	Preciso e inmediato
No supervisado	No	Encontrar patrones o estructuras	No explícito
Por refuerzo	No (recompensa)	Maximizar recompensa acumulada	Basado en la interacción

Tabla 3.1: Comparativa Aprendizajes Automáticos

Este tipo de aprendizaje se suele modelar mediante procesos de decisión de **Markov (MDP)**, estructura matemática que define formalmente el entorno como un conjunto de **estados** posibles, un conjunto de **acciones** disponibles, una **función de transición** entre estados y una **función de recompensa** [3].

Como refleja el esquema de la Figura 3.1, en cada instante de tiempo, el agente observa el estado del entorno y, en base a eso, siguiendo una **política**, selecciona una acción. El entorno pasa a estar en un nuevo estado y calcula una recompensa en función de ese nuevo estado, para luego devolverle ambos valores (estado y recompensa) al agente. Y así de forma **cíclica** hasta alcanzar una meta. El objetivo del agente es aprender una política (una estrategia de decisión) que maximice las recompensas futuras.

Un aspecto de los más importantes de este paradigma es el **dilema exploración-explotación**: el agente debe balancear entre explorar nuevas acciones que podrían conducir a mayores recompensas y explotar las acciones que ya sabe que son efectivas. Para lograr este aprendizaje, se utilizan algoritmos como **Q-Learning**, **SARSA** o métodos basados en políticas, y en el caso del DRL, estos se combinan con **redes neuronales profundas** para manejar entornos con estados continuos o de alta dimensión [4].

Aprendizaje profundo

Técnica de inteligencia artificial basada en redes neuronales con **múltiples capas**, que permite aprender **representaciones complejas** de los datos y que ha sido clave en el desarrollo de sistemas inteligentes capa-

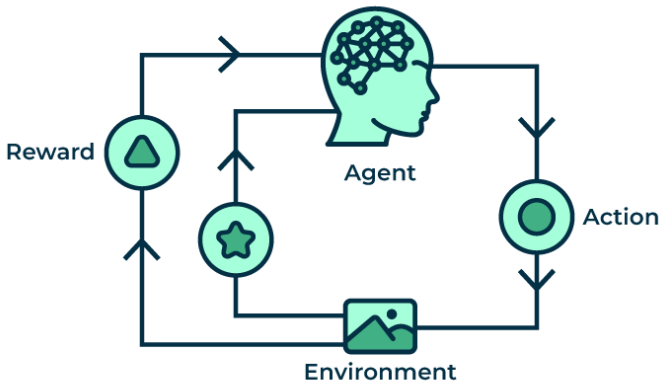

Figura 3.1: Esquema Aprendizaje por Refuerzo

ces de resolver tareas avanzadas como el reconocimiento de imágenes, el procesamiento de lenguaje natural o el control autónomo.

Están compuestas por una sucesión de capas conectadas entre sí, donde cada neurona realiza operaciones matemáticas sobre los datos de entrada y, en base a un umbral, transmite el resultado a la siguiente capa. A medida que los datos avanzan por la red, se van **extrayendo características** de mayor nivel de abstracción. Esto permite que el modelo no solo reconozca patrones simples, sino también **relaciones complejas** entre los datos.

Estas capas suelen estar clasificadas en 3 tipos, como observamos en la Figura 3.2: La **capa de entrada** o input layer, que recibe los datos de entrada y los transforma en una **representación numérica** que puede ser procesada por la red. Esta capa no realiza cálculos complejos: su función principal es presentar los datos (como píxeles de una imagen o características de un conjunto de datos) al sistema.

Las **capas ocultas**, situadas entre la entrada y la salida, son responsables del procesamiento interno. Cada una de sus neuronas aplica una función de activación no lineal a una combinación ponderada de las salidas de la capa anterior. A medida que los datos atraviesan estas capas, la red aprende representaciones cada vez más abstractas, permitiéndole identificar **estructuras complejas**, como bordes, formas o incluso conceptos.

Finalmente, la **capa de salida** produce la **predicción final** del modelo. Su formato depende del tipo de tarea: en clasificación, suele contener una neurona por clase y aplica funciones como softmax para obtener probabi-

lidades; en regresión, puede tener una sola neurona que entrega un valor continuo.

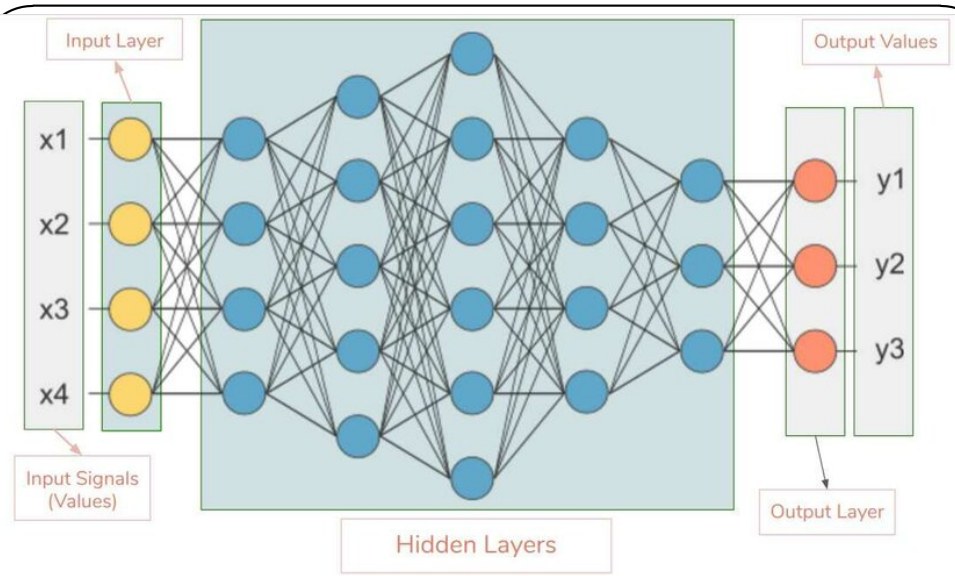

Figura 3.2: Esquema Red Neuronal Profunda

El proceso de aprendizaje se basa en el **ajuste** de los **pesos sinápticos** entre neuronas, es decir, los umbrales de activación o propagación de la señal, mediante algoritmos como el descenso del gradiente, utilizando la retropropagación del error (backpropagation). Este enfoque requiere una gran cantidad de datos y potencia computacional debido al alto grado de operaciones realizadas simultáneamente, pero a cambio ofrece un refinamiento en la comprensión de los datos muy útil y en ocasiones necesario [5].

En el contexto del Deep Reinforcement Learning, las redes neuronales profundas se utilizan como **aproximadores** de funciones, permitiendo al agente generalizar comportamientos a partir de experiencias previas, lo que resulta especialmente útil en entornos con espacios de estados amplios o continuos, donde las técnicas tradicionales de aprendizaje por refuerzo no escalan adecuadamente. Gracias a esta combinación, es posible abordar problemas complejos y de **alta dimensionalidad** que antes no se podían abordar.

Algoritmo PPO (Proximal Policy Optimization)

Uno de los algoritmos más utilizados en el campo del Aprendizaje por Refuerzo Profundo es **PPO** (Proximal Policy Optimization), desarrollado por OpenAI. Este algoritmo pertenece a la familia de métodos **basados en políticas**, y se ha convertido en una elección estándar debido a su buena relación entre rendimiento, estabilidad y facilidad de implementación.

PPO está diseñado para actualizar la política del agente de forma gradual y controlada, **evitando cambios bruscos** que puedan desestabilizar el aprendizaje. Para lograrlo, en lugar de aplicar actualizaciones agresivas como en otros métodos anteriores, PPO utiliza una estrategia de optimización que restringe cuánto puede cambiar la política en cada iteración, mediante un **mecanismo de penalización** o recorte (clipping) en la función objetivo.

El objetivo del algoritmo es maximizar la función de ventaja esperada del agente, es decir, **reforzar** aquellas **acciones** que resultaron **mejores de lo esperado**, y reducir la probabilidad de acciones menos efectivas. El término "proximal" hace referencia a que se favorecen cambios pequeños ("proximales") en la política, para asegurar un aprendizaje más estable.

Matemáticamente, PPO actualiza su política en base a esta fórmula [6]:

$$\theta_{k+1} = \underset{\theta}{argmax} \underset{s,a\sim\pi\theta_k}{E}[L(s,a,\theta_k),\theta]$$

Normalmente en múltiples pasos para maximizar el valor objetivo. Aquí, L viene dado por la siguiente fórmula simplificada:

$$L(s,a,\theta_k),\theta = min(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), g(\epsilon, A^{\pi_{\theta_k}}(s,a))$$

Donde:

- s es el estado del entorno.
- a es la acción tomada por el agente en el estado s.
- θ_k son los parámetros de la política antigua (antes de la actualización).
- θ son los parámetros actuales de la política (los que se están optimizando).
- $\pi_\theta(a|s)$ es la probabilidad de que la política actual π_θ tome la acción a dado el estado s.
- $\pi_{\theta_k}(a|s)$ es la probabilidad de que la política anterior π_{θ_k} tome la acción a dado el estado s.

- $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ es el ratio de probabilidad o *importance sampling ratio*.
- $A^{\pi_{\theta_k}}(s,a)$ es la ventaja estimada (advantage function) bajo la política anterior π_{θ_k}.
- $g(\epsilon, A)$ es la función de recorte (clipping), definida como:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & \text{si } A \geq 0 \\ (1-\epsilon)A & \text{si } A < 0 \end{cases}$$

3.2. Redes Ópticas Pasivas (PON)

Las Redes Ópticas Pasivas, o PON por sus siglas en inglés, son una tecnología de telecomunicaciones que permite la distribución de servicios de datos, voz y video a través de fibra óptica, sin necesidad de componentes activos entre el proveedor y el usuario final. Su arquitectura eficiente y de bajo mantenimiento las ha convertido en una de las soluciones más extendidas para el acceso de banda ancha en los últimos años.

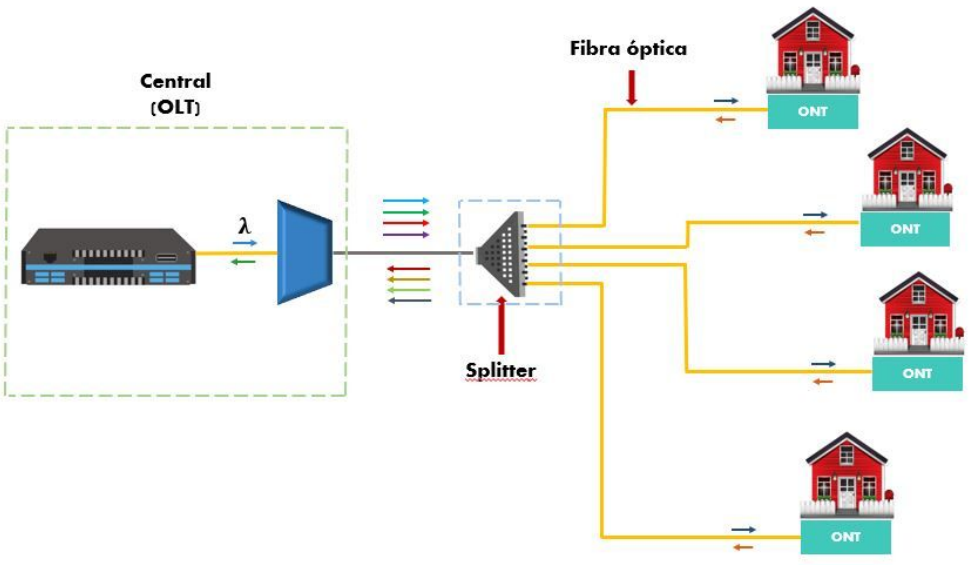

Figura 3.3: Esquema Red Óptica Pasiva

Como podemos ver en el esquema de la Figura 3.3, una red PON se basa en una topología punto a multipunto que utiliza divisores ópticos

(splitters) para distribuir la señal desde una única fuente central, el OLT (Optical Line Terminal), ubicado en la central del operador, hacia múltiples receptores llamados **ONUs** (Optical Network Units) o **ONTs** (Optical Network Terminals) en el domicilio del usuario. La red intermedia no requiere alimentación eléctrica ni elementos activos, lo que reduce costos y mejora la fiabilidad del sistema [7].

El funcionamiento se basa en la multiplexación en el dominio del tiempo (TDM) o en longitudes de onda (WDM) para gestionar la comunicación entre múltiples usuarios que comparten la misma línea de conexión de fibra. Entre las variantes más conocidas se encuentran las contenidas en la Tabla 3.2 [8].

Tecnología	**Estándar**	**V. Bajada**	**V. Subida**	**Multiplex**
EPON	IEEE 802.3ah	1,25 Gbps	1,25 Gbps	TDM
GPON	ITU-T G.984	2,5 Gbps	1,25 Gbps	TDM
XG-PON	ITU-T G.987	10 Gbps	2,5 Gbps	TDM
XGS-PON	ITU-T G.9807.1	10 Gbps	10 Gbps	TDM
NG-PON2	ITU-T G.989	40 Gbps	10 Gbps	WDM+TDM

Tabla 3.2: Variantes de redes PON

A pesar de ser una tecnología cuyo desarrollo comenzó en la década de 1990, se ha convertido, gracias a su gran **ancho de banda**, **escalabilidad** y **eficiencia energética**, en una pieza clave en las telecomunicaciones a día de hoy, dado el incremento en los últimos años de despliegues de infraestructuras de acceso de nueva generación, como FTTH (Fiber To The Home). Son también cada vez más relevantes en aplicaciones empresariales, 5G y redes de centros de datos.

4. Técnicas y herramientas

4.1. Metodología

Para llevar a cabo este proyecto he decidido utilizar **Scrum** como metodología de trabajo. Aunque está pensada originalmente para equipos, la he adaptado a nivel individual porque me ha parecido una forma muy útil de organizar el desarrollo por fases y marcar objetivos claros a corto plazo.

El enfoque **iterativo** e **incremental** de Scrum, basado en ciclos cortos llamados **sprints**, me ha permitido dividir el trabajo en bloques manejables, revisar lo que iba consiguiendo y ajustar la planificación según lo necesitaba. Esta flexibilidad ha sido clave para adaptarme a problemas inesperados o cambios en los requisitos, que han ido surgiendo a medida que avanzaba. Además, me ha ayudado a mantener una visión clara del progreso y a estructurar mejor las tareas, algo fundamental cuando se trabaja solo.

4.2. Herramientas

Visual Studio Code

Para el desarrollo del software he elegido **Visual Studio Code** como entorno de trabajo. Esta decisión se basa en varias razones. En primer lugar, se trata de un editor **ligero**, multiplataforma y de código abierto, con un rendimiento excelente incluso en proyectos de tamaño medio o grande. Además, su **amplia comunidad** y **ecosistema de extensiones** permiten adaptarlo fácilmente a distintos lenguajes y flujos de trabajo.
En mi caso, Visual Studio Code me ha resultado especialmente útil por su **interfaz limpia**, la posibilidad de **incorporar herramientas de pro-**

ductividad directamente en el editor, como gestores de tareas, visores de Markdown o seguimiento de pendientes, y la amplia **experiencia** que poseo trabajando con el, en comparación con otros editores.

GitHub

Para la gestión del código y el control de versiones he utilizado **GitHub**, ya que era un requisito establecido para el desarrollo del proyecto. Esta plataforma me ha permitido mantener una **copia segura** del proyecto en la nube y llevar un **registro ordenado** de los cambios que he ido realizando a lo largo del tiempo.

Durante el desarrollo, he hecho diversos **commits** descriptivos, lo cual me ha ayudado a seguir el progreso del trabajo y a identificar con facilidad cuándo y cómo se introdujeron determinadas modificaciones. Gracias a GitHub he podido trabajar de forma más **estructurada** y cumplir con los requisitos de trazabilidad que se pedían para el proyecto. [9]

Overleaf

Para la redacción de la memoria y los anexos del proyecto he utilizado **Overleaf** como entorno de trabajo en **LaTeX**. Elegí esta plataforma principalmente por su **facilidad de uso** y **disponibilidad online**, que me han permitido gestionar toda la documentación de forma organizada y accesible desde cualquier dispositivo.

En particular, he trabajado sobre la **plantilla oficial** proporcionada por la universidad, la cual ya incluía la estructura básica del documento, así como comandos definidos, guiones orientativos y aclaraciones sobre el contenido de cada sección.

A diferencia de otras herramientas locales como TeXstudio o MikTeX, Overleaf no requiere instalación ni configuración adicional, lo que me evitó posibles incompatibilidades y me facilitó el trabajo desde distintos dispositivos. Además, tener todos los archivos relacionados con la memoria (texto, anexos, figuras y bibliografía) unificados en un único entorno online me ayudó a tener una gestión más clara y eficiente. [10]

PyInstaller

Con el objetivo de facilitar el despliegue del sistema y permitir su ejecución en entornos donde no esté instalado Python ni sus dependencias, se ha utilizado la herramienta PyInstaller. Esta utilidad permite generar un archivo ejecutable (.exe) a partir del código fuente del proyecto, empaquetando internamente todos los módulos, librerías y recursos necesarios para su funcionamiento. Gracias a ello, el sistema puede distribuirse como una aplicación autónoma, ejecutable directamente en sistemas Windows sin necesidad de configuración adicional. Se ha configurado PyInstaller para que incluya correctamente los módulos internos, los ficheros de entorno y los scripts del agente y el simulador, asegurando que la interfaz gráfica funcione de forma estable en el ejecutable final. [11]

4.3. Librerías

Gymnasium

Gymnasium es una librería de Python diseñada para la creación de **entornos de simulación** compatibles con algoritmos de aprendizaje por refuerzo. Derivada originalmente de OpenAI Gym, proporciona una interfaz estándar basada en métodos como reset() y step(), lo que facilita la integración con distintas implementaciones de agentes.

En el contexto de este proyecto, Gymnasium se utiliza para implementar el entorno simulado de una red óptica pasiva (PON). A través de esta interfaz, el agente DRL puede interactuar con el simulador **tomando decisiones** (acciones), **recibiendo observaciones** del estado actual de la red, y **obteniendo recompensas** en función de su rendimiento. Este entorno encapsulado define la lógica de evolución del sistema, gestionando los ciclos de simulación, los eventos internos de la red y el sistema de recompensas que guía el aprendizaje del agente.

Gracias a Gymnasium, el entorno puede reutilizarse y evaluarse fácilmente con distintos algoritmos o configuraciones, manteniendo una **separación** clara entre la **lógica del simulador** y el **agente de aprendizaje**. [12]

StableBaselines3

StableBaselines es una librería de aprendizaje por refuerzo que proporciona implementaciones fiables y actualizadas de varios **algoritmos de DRL**. Está desarrollada sobre PyTorch y ofrece una interfaz sencilla para el entrenamiento, evaluación y gestión de agentes en entornos compatibles con Gymnasium.

En este proyecto, se ha utilizado para instanciar y entrenar el agente utilizando el algoritmo PPO (Proximal Policy Optimization). Con ella podemos **configurar** fácilmente los **hiperparámetros** del modelo, iniciar el **entrenamiento sobre el entorno simulado** y almacenar los modelos entrenados para su evaluación posterior. Además, proporciona herramientas integradas para el **registro de métricas** y callbacks durante el entrenamiento, lo que facilita el análisis del comportamiento del agente y su rendimiento a lo largo del tiempo.

La integración con Gymnasium es directa, lo que permite que el agente PPO interactúe con el entorno de simulación de forma eficiente y modular. [13]

Simpy

Simpy es una librería de simulación de **eventos discretos** en Python, orientada a modelar procesos concurrentes que interactúan en el tiempo. Está basada en generadores y permite **definir procesos** como funciones que se ejecutan en paralelo dentro de un entorno simulado, sin necesidad de utilizar múltiples hilos reales.

Empleamos SimPy dentro del simulador de redes XGS-PON para modelar el comportamiento de las distintas ONTs como **procesos independientes**. Cada ONT funciona de forma concurrente respecto a la OLT, generando tráfico, gestionando colas y reaccionando a la asignación de ancho de banda. Gracias a SimPy, es posible simular esta **actividad simultánea** de manera controlada y reproducible, lo que permite representar de forma realista la dinámica de una red óptica pasiva sin recurrir a concurrencia real ni programación multihilo. [14]

TKinter

Para el desarrollo de la **interfaz gráfica** del sistema se ha optado por utilizar Tkinter, la librería estándar para la creación de interfaces gráficas en Python.
Esta elección se debe principalmente a su facilidad de uso, su integración nativa con Python —ya que forma parte de la biblioteca estándar— y su capacidad para desarrollar interfaces funcionales y ligeras sin requerir dependencias externas.

Tkinter proporciona un conjunto de widgets suficiente para construir menús, botones, campos de entrada, paneles y otras estructuras necesarias para la interacción con el usuario. Dado que el objetivo era construir una interfaz sencilla pero funcional, que permitiera controlar desde un único punto la ejecución del agente y del simulador, configurar parámetros y visualizar resultados, Tkinter resultó ser una opción adecuada por su bajo coste de aprendizaje y su versatilidad. [15]

Librería	Función	Uso en el proyecto
Gymnasium	Simulación de entornos RL	Creación de entornos compatibles con DRL
Stable-Baselines3	Creación de modelos PPO	Implementar PPO y gestionar modelos con este algoritmo
Simpy	Simulación de eventos discretos	Ejecutar ONTs paralelas para simular tráfico PON
Tkinter	Interfaz gráfica de usuario	Construcción de la GUI del sistema como punto de entrada

Tabla 4.1: Resumen librerías

En la tabla 4.1 (resumen) podemos ver todas las librerías de Python utilizadas en el proyecto.

5. Aspectos relevantes del desarrollo del proyecto

5.1. Investigación y análisis inicial

El desarrollo de este proyecto comenzó con una fase de investigación y análisis centrada en las redes ópticas pasivas (PON), ya que se trata de una tecnología que no se aborda durante el grado. Esta etapa también incluyó el estudio detallado de la documentación y el código fuente de los proyectos desarrollados previamente por otros compañeros.

Para familiarizarme con el funcionamiento de las redes PON, además de asistir a diversas tutorías, llevé a cabo una búsqueda bibliográfica que incluyó la lectura de artículos científicos y libros especializados sobre redes ópticas pasivas, muchos aportados también por mis tutores.

Una vez asimilados los conceptos fundamentales, procedí a ejecutar y analizar el código existente con el fin de comprender su estructura, identificar posibles puntos de mejora y comenzar a planificar los cambios necesarios para el nuevo diseño arquitectónico. Este análisis inicial fue clave para detectar oportunidades de refactorización y sentar las bases para una futura integración entre los distintos componentes del sistema.

5.2. Diseño de la arquitectura

Tras la fase inicial de investigación y análisis, comencé con el diseño de la nueva arquitectura del sistema. En un primer momento consideré refactorizar

por completo tanto el agente como el simulador. Sin embargo, al revisar el código del simulador, observé que ya contaba con una estructura bastante clara y una cantidad considerable de componentes bien definidos, por lo que decidí centrarme principalmente en el agente DRL, que presentaba un mayor margen de mejora y simplificación.

El agente original consistía en un único fichero en formato notebook y una clase adicional que implementaba un entorno personalizado heredando de gym.Env. Como primer paso, procedí a extraer las funcionalidades principales en diferentes ficheros .py. Por un lado, diseñé una clase Agente, encargada de la lógica principal de entrenamiento y evaluación del modelo. Por otro, desarrollé varios módulos auxiliares que contienen funcionalidades específicas, como la impresión de resultados o la gestión de modelos (creación, guardado y carga desde fichero).

A partir de esta estructura modular inicial, amplié el diseño para incluir la parte de comunicación externa con el simulador, lo que me llevó a implementar dos patrones de diseño con objetivos concretos:

- **Patrón Adapter** Este patrón lo utilicé para desarrollar la interfaz de conexión entre el simulador y el agente, permitiendo que el simulador acceda al modelo de forma transparente y pueda solicitar predicciones para tomar decisiones sobre la asignación de ancho de banda.
  Como vemos en la Figura 5.1, con este patrón el cliente que requiere ejecutar una acción -en este caso el simulador solicita una asignación- no accede directamente al método o clase que genera esos resultados, sino que existe un elemento intermedio que se encarga de gestionar la comunicación.

- **Patrón Strategy** Gracias a este patrón, estructuré el agente para soportar dos modos de ejecución diferenciados:

  1. Un modo local, en el que el agente entrena y evalúa utilizando su propio entorno simulado (basado en Gymnasium), incluyendo su lógica de generación de tráfico.
  2. Un modo integrado, en el que el simulador invoca directamente al agente, omitiendo el entrenamiento y solicitando únicamente predicciones para la toma de decisiones.

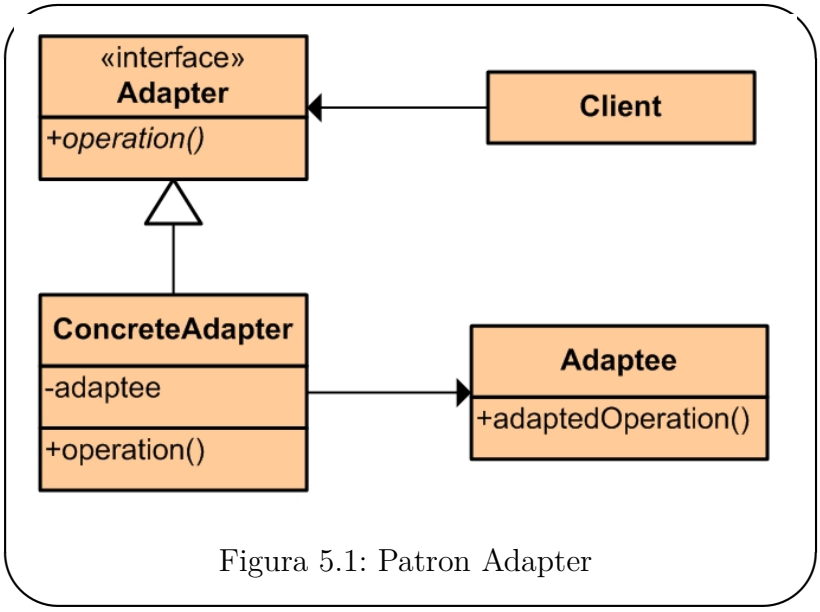

Figura 5.1: Patron Adapter

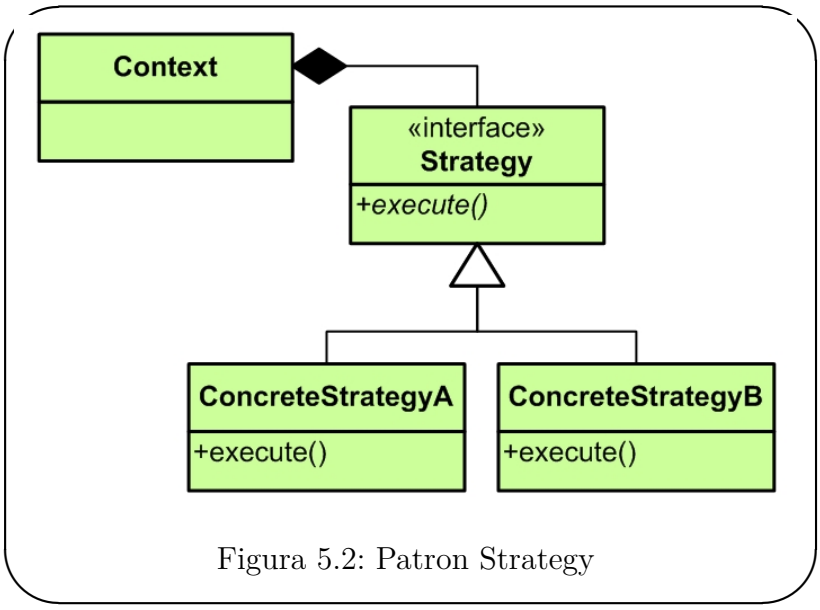

Figura 5.2: Patron Strategy

5.3. Desarrollo del agente

Ya finalizada esta primera fase de análisis y diseño, comencé a implementar el nuevo diseño en el código, empezando con el agente al ser la parte más desarrollada.

Implementé los primeros cambios básicos, extrayendo funcionalidades del script principal para crear la clase Agente y los diferentes módulos para el graficado de resultados y la gestión de modelos (creación, guardado y carga). Con ello resolví diversos problemas que tenía el código inicial:

- Entornos paralelos: El programa inicial estaba pensado para permitir los entornos paralelos durante el entrenamiento, para poder reducir los tiempos y aprovechar mejor los recursos computacionales. Este entrenamiento paralelo no funcionaba correctamente por ya que, aunque el entrenamiento se realizaba correctamente en paralelo, la evaluación se ejecutaba con el mismo vector de entornos y esto provocaba que en los vectores de resultados, se almacenara de forma concatenada los datos de todos los entornos, generando unas gráficas incoherentes.

- Gestión del entorno: El entorno de gymnasium cuenta con un mecanismo para poder detectar cuando se termina la simulación, a través de una variable Done que indica la finalización del entorno. Este comportamiento no estaba bien utilizado ya que la finalización del entorno se gestionaba desde fuera del entorno comprobando, con un contador, si se había alcanzado el limite de ciclos. Adapté este mecanismo de comprobación de ciclos de forma interna en el entorno, devolviendo este una variable booleana llamada Done, que indica cuando un entorno ha finalizado su ejecución. Esto, además de ser un comportamiento más lógico, aprovechando esa flag interna con la que cuentan los entornos de gymnasium, también ayuda en el entrenamiento a que el modelo sepa cuando un entorno ha terminado y debe resetearlo para volver a comenzar una simulación.

- Graficado de entornos paralelos: Relacionado con el problema de los entornos paralelos, la representación gráfica de los resultados de una evaluación no funcionaba correctamente al realizarla con entornos paralelos ya que no se tenía en cuenta al realizar el guardado de los resultados. Esto se solucionó adaptando el proceso de evaluación para que este se ejecutara con un único entorno.

- Variables no utilizadas: A mayores también realicé varios cambios para mejorar la estructura del código, eliminando numerosas variables que o bien no se utilizaban o bien se podrían gestionar de una manera mas eficiente.

Tras estas implementaciones, descubrí que el modelo no estaba funcionando, así que tuve que dejar todo aparcado temporalmente para ponerme a solucionar el principal problema.

Y, tras solucionar este comportamiento, continué adaptando el agente a la nueva arquitectura, implementando la clase abstracta BaseAgent de la cual heredan las nuevas clases LocalAgent y SimAgent, para permitir estos dos nuevos tipos de ejecución: Una utilizando los entornos de gymnasium, y otra utilizando el simulador realista.

5.4. Colaboración

Durante todo este proceso de desarrollo, he colaborado de forma conjunta con los alumnos Clara Ruiz de las Heras y David Rodríguez Aragón, ambos alumnos de la Universidad de Valladolid y tutelados por Noemí Merayo Álvarez. Colaborando con ellos en los siguientes apartados:

- Clara ha estado estos meses trabajando en el mecanismo de **generación de tráfico** en el entorno de Gymnasium. Antes teníamos una versión de Pareto muy simple y sencilla, y Clara ha estado desarrollando un fichero con una serie de clases para hacer una generación de tráfico más realista, también basada en Pareto, pero de una forma más compleja y detallada, con diferentes clases y parámetros.

  Ha adaptado el entorno de Gymnasium para que utilice la generación de tráfico de este fichero TrafficoParetoPython, donde a través de una clase ONT simulamos la generación de tráfico mediante Pareto. Además, ha preparado diferentes entornos de simulación para poder simular tráfico en redes con diferentes características como redes con tráfico desbalanceado o con ONTs con diferentes acuerdos de nivel de servicio. Y para terminar ha utilizado la herramienta **Optune** para afinar los **hiperparámetros** del modelo, realizando diversas pruebas muy extensas.

- Por otro lado, David ha estado trabajando con el **simulador**. Lo ha adaptado para su funcionamiento con redes de hasta 10 Gbps (XGS-PON), mejorando las estadísticas, añadiendo nuevas variables para la visualización el total de ancho de banda asignado y diferentes métricas. También lo ha preparado para una futura extensión o adaptación a redes de 25 Gbps y ha integrado la parte de generación del tráfico desarrollada por Clara.

De esta forma tenemos por un lado el entorno de Gymnasium y por otro lado el simulador que, aunque generen el tráfico de la misma forma, para el entrenamiento tenemos un comportamiento de la red más sencillo con el entorno de Gymnasium y para la evaluación tenemos un simulador mucho más realista en el que tenemos más parámetros y los paquetes, por ejemplo, no son simples números de tamaño, sino que tienen una estructura interna y la red se comporta de una forma más realista donde las ONTs y la OLT intercambian paquetes.

5.5. Problemas

El desarrollo de este proyecto ha estado muy marcado por el casi único y principal problema, que además descubrí a mitad de desarrollo: El modelo no funcionaba.

Durante la implementación de la nueva arquitectura del agente, comencé a realizar tareas de depuración para asegurar que el sistema ejecutaba correctamente cada uno de sus componentes. En una de estas pruebas, observé que la variable action, que representa la predicción generada por el modelo, presentaba valores anormalmente bajos, cercanos a cero. Este comportamiento resultaba inesperado, ya que las acciones del agente están directamente relacionadas con la asignación de ancho de banda, cuyos valores se encuentran en el orden de 10e8, al ser valores en bits. Además, verifiqué que no se estaba aplicando ningún proceso de desnormalización a la predicción del modelo antes de ser utilizada.
Lo sorprendente era que, a pesar de este comportamiento, el sistema parecía funcionar correctamente: en las gráficas generadas se podía observar que las ONTs recibían y transmitían tráfico en cada ciclo, y las colas se llenaban y vaciaban como cabría esperar.

Fue entonces cuando descubrí que el sistema contenía una lógica interna que permitía simular un funcionamiento correcto incluso sin que el modelo aprendiera de forma efectiva.
Concretamente, al revisar el método ***step()*** —encargado de procesar la acción predicha, actualizar las colas y gestionar el tráfico entrante y saliente— encontré las siguientes líneas de código:

```
self.trafico_salida = np.clip(action, 0, self.Max_bits_ONT)

for i in range(self.num_ont):
    self.trafico_pendiente[i] +=  self.trafico_entrada[i] -
       self.trafico_salida[i]
    if self.trafico_pendiente[i] > 0:
        self.trafico_salida[i] = min(self.trafico_pendiente[
           i], self.Max_bits_ONT)
        self.trafico_pendiente[i] -= self.trafico_salida[i]
```

Aquí podemos observar cómo el valor de la variable ***action*** es asignado a la variable ***trafico_salida***, sin aplicar transformaciones de ningún tipo y como, a continuación, se le resta este valor al tráfico de entrada para después agregar el resultado a la cola. El problema viene cuando, acto seguido, se comprueba si hay elementos en la cola y, en caso afirmativo, se vuelve a calcular un nuevo valor para ***trafico_salida*** para luego volver a restarle este valor a la cola.

Esto convierte al agente en un sistema que simula generación de tráfico y simplemente asigna ancho de banda a los diferentes terminales cuando contienen tráfico en la cola, sin tomar ningún tipo de decisión crítica ni dar prioridades a unos u otros terminales.

Tras solucionar este problema, arreglando ese mal procesamiento del tráfico con las siguientes líneas:

```
# Comprobamos si el ancho de banda asignado excede el
   disponible
total = np.sum(self.trafico_salida)
if total > self.Max_bits_ONT:
    self.trafico_salida *= (self.Max_bits_ONT / total)

# Procesamos el trafico de entrada y el asignado
for i in range(self.num_ont):
    self.trafico_pediente[i] += self.trafico_entrada[i]

    if self.trafico_salida[i] > self.trafico_pendiente[i]:
        self.trafico_salida[i] = self.trafico_pendiente[i]

    self.trafico_pendiente[i] -= self.trafico_salida[i]
```

El modelo seguía sin aprender correctamente, por lo que tuve que modificar distintas funcionalidades e hiperparámetros con el objetivo de mejorar el proceso de entrenamiento y obtener un buen comportamiento por parte del agente.
Tras varias semanas de pruebas, y tras ejecutar numerosas simulaciones con diferentes combinaciones de ajustes, logré alcanzar una configuración con un rendimiento de aprendizaje notable y resultados consistentes. Para ello, implementé los siguientes cambios:

- **Espacios de observaciones y de acciones** Modifiqué los espacios de observaciones y de acciones para que ahora estuvieran normalizados entre 0 y 1. Para ello tuve que cambiar la forma en la que el entorno procesaba la acción del modelo, multiplicándola por el máximo de ancho de banda de las ONTs, ***B_max***, al principio del método ***step()*** para desnormalizarla. E implementando un tope de cola que me permitiera normalizar las observaciones que el entorno devolvía tras ejecutar la acción, dividiendo ***B_demand*** entre este máximo.

- **Cálculo de la recompensa** Para lograr este buen aprendizaje por parte del modelo, cambié por completo el cálculo de la recompensa, logrando aportarle una mejor información sobre las consecuencias de sus acciones.
  Para ello utilicé los siguientes componentes:

  - ***delta_cola*** Es la diferencia de la suma de las colas en el ciclo actual respecto de la suma en el ciclo anterior. Así conseguimos que penalice cuando las colas aumenten, y que premien el vaciado de éstas
  - ***uso_bw*** Representa el ancho de banda asignado a todas las ONTs respecto del máximo que se puede asignar, para premiar la utilización de ancho de banda
  - ***limit_queue*** Penaliza al modelo cuando la suma total de los anchos de banda asignados, supera el limite impuesto por la OLT

  Todos ellos normalizados y combinados de forma lineal con la siguiente fórmula:

  $$\alpha \cdot delta\_cola + \beta \cdot uso\_bw - \gamma \cdot limit\_queue$$

  Y a partir de aquí fuí probando diferentes valores para los coeficientes hasta conseguir unos resultados aceptables

- **Ajuste de hiperparámetros** Por último, para ajustar mejor el entrenamiento del modelo, realicé un pequeño ajuste en los valores de los hiperparámetros a la hora de crear el modelo.

5.6. Desarrollo final

En la fase final del desarrollo del proyecto, los esfuerzos se han centrado en dotar al sistema de una interfaz gráfica de usuario (GUI) completa que permita interactuar con el agente y el simulador de forma sencilla, intuitiva y sin necesidad de acceder directamente al código fuente. Esta parte ha sido clave para convertir el conjunto de scripts y módulos desarrollados durante las fases anteriores en una herramienta accesible, lista para ser utilizada por terceros con distintos niveles de conocimiento técnico.

La interfaz gráfica, aunque sencilla en su diseño, ofrece una serie de funcionalidades que permiten gestionar de forma centralizada los principales procesos del sistema. Desde esta interfaz es posible:

- Ejecutar únicamente el agente DRL en modo local, utilizando su entorno Gymnasium integrado para entrenar y evaluar modelos.
- Ejecutar el simulador con el agente acoplado, haciendo que sea este quien tome las decisiones sobre asignación de ancho de banda en cada ciclo.
- Elegir entre entrenar un nuevo modelo desde cero o evaluar un modelo previamente entrenado, seleccionándolo desde el propio panel.
- Visualizar los resultados directamente en la interfaz, incluyendo gráficas generadas automáticamente tras la simulación (uso de ancho de banda, evolución de las colas, rendimiento del agente, etc.).
- Configurar opciones generales del sistema, como la duración de la simulación, el número de ONTs o la estrategia de generación de tráfico.

Esta interfaz actúa como punto de entrada centralizado para el sistema, permitiendo controlar desde un único lugar los distintos componentes y funcionalidades, lo cual ha supuesto un paso clave hacia una mayor usabilidad y accesibilidad de la herramienta.

Además del desarrollo de esta GUI, otra parte fundamental de esta última etapa ha sido la reorganización completa del proyecto a nivel de estructura de archivos, paquetes e importaciones, con el objetivo de facilitar su despliegue y distribución. Esto ha implicado:

- La refactorización de directorios y nombres de paquetes, siguiendo convenciones estándar de Python.
- La adaptación de las rutas internas e importaciones relativas, para asegurar que los módulos se encuentren correctamente sin necesidad de modificar manualmente sys.path ni configurar rutas absolutas.
- La creación de un punto de entrada único que lanza la interfaz gráfica y se encarga de gestionar la carga de los subcomponentes de forma transparente para el usuario.

Con el objetivo de hacer el sistema completamente portable y utilizable sin necesidad de instalaciones manuales, se ha preparado un proceso de empaquetado mediante PyInstaller, que permite generar un ejecutable (.exe) del sistema. Gracias a ello, el usuario final puede ejecutar la herramienta directamente, sin necesidad de instalar previamente Python ni configurar entornos virtuales o dependencias externas. Todo el sistema queda encapsulado en un único fichero ejecutable, lo que facilita su distribución y su uso en equipos sin entorno de desarrollo.

En conjunto, en esta fase final he transformado el proyecto en una aplicación ejecutable y lista para su distribución, con un flujo de trabajo accesible desde una interfaz visual, configuraciones simplificadas, y sin necesidad de conocimientos técnicos avanzados para su utilización. Este aspecto representa un valor añadido significativo al proyecto, especialmente de cara a su posible uso por parte de investigadores, docentes o estudiantes que deseen trabajar con simulaciones de redes ópticas pasivas y aprendizaje por refuerzo sin enfrentarse a las barreras técnicas del código base.

Se puede consultar todo el código fuente del proyecto final en mi repositorio de GitHub [16].

6. Trabajos relacionados

El presente trabajo parte de la integración y mejora de dos proyectos previamente desarrollados como Trabajos Fin de Grado por otros alumnos de universidades españolas. Ambos proyectos abordan, de forma independiente, aspectos clave del sistema que aquí se propone: por un lado, el desarrollo de un agente de aprendizaje por refuerzo profundo (DRL), y por otro, la simulación realista de una red óptica pasiva (PON). El objetivo de este TFG ha sido unificar, refactorizar y ampliar estos trabajos para construir una herramienta funcional, coherente y reutilizable. A continuación se describen brevemente estos trabajos y se detalla el modo en que han sido aprovechados en esta memoria.

6.1. Agente DRL

El primer trabajo sobre el que se construye esta memoria fue desarrollado en 2024 por el alumno David Pérez Moreno, como TFG del Grado en Ingeniería Informática en la Universidad de Burgos, bajo el título "Aprendizaje por refuerzo en redes ópticas pasivas" [2].

Este proyecto consistía en la implementación de un agente de aprendizaje por refuerzo profundo utilizando la librería Stable-Baselines3 con el algoritmo PPO (Proximal Policy Optimization). Para ello, se diseñó un entorno personalizado compatible con Gymnasium, que simulaba de forma muy simplificada el comportamiento de una red PON mediante colas y demanda de ancho de banda generada aleatoriamente. El agente era entrenado sobre este entorno con el objetivo de aprender una política que asignara el ancho de banda de forma eficiente entre distintas ONTs, en función del estado de las colas.

El código original estaba estructurado principalmente en un único notebook de Jupyter, con una clase de entorno que heredaba directamente de gym.Env y algunos métodos auxiliares. Aunque funcional, el proyecto presentaba limitaciones en cuanto a modularidad, escalabilidad y separación de responsabilidades, además de contener ciertos errores o comportamientos que impedían el aprendizaje efectivo del modelo.

En el marco del presente trabajo, este agente ha sido profundamente refactorizado, rediseñado y mejorado. Se ha implementado una arquitectura orientada a objetos, se ha dividido el código en módulos reutilizables, se han corregido errores en el procesamiento del tráfico y en el cálculo de recompensas, y se han añadido nuevas funcionalidades como la posibilidad de operar en modo local o modo integrado con el simulador. Además, se ha diseñado una clase base (BaseAgent) que permite instanciar distintos tipos de agentes en función del modo de operación, facilitando la extensión futura del sistema.

6.2. Simulador de redes PON

El segundo proyecto base fue desarrollado también en 2024 por el alumno Víctor Herrezuelo Paredes, como TFG del Grado en Ingeniería de Tecnologías de Telecomunicación en la Universidad de Valladolid, bajo el título "Implementación de un simulador de redes de acceso ópticas pasivas en Python" [1].

Este trabajo consistía en la construcción de un simulador de redes EPON y XGS-PON utilizando la librería SimPy, especializada en simulación de eventos discretos. El simulador modelaba el comportamiento de una red punto a multipunto en la que una OLT (Optical Line Terminal) repartía el ancho de banda entre varias ONTs (Optical Network Terminals). Cada ONT generaba tráfico en paralelo y respondía a las asignaciones de la OLT en función del ancho de banda disponible.

El diseño del simulador estaba bastante avanzado, con una estructura clara, separación de módulos y soporte para distintas configuraciones de red. Sin embargo, el sistema no contaba con ningún mecanismo de optimización inteligente: la OLT seguía una lógica fija o aleatoria para repartir los recursos, sin tener en cuenta el estado global de la red o la evolución temporal de las colas.

En este TFG, el simulador ha sido adaptado para interactuar con un agente externo, que actúa como módulo de decisión dentro de la OLT. Esta

integración ha requerido la implementación de una interfaz de comunicación entre el simulador y el agente, así como la redefinición de ciertas partes del flujo de ejecución para permitir la invocación del modelo de forma sincronizada durante la simulación. Además, se han llevado a cabo mejoras internas, como la incorporación de nuevos parámetros configurables, la ampliación de estadísticas generadas y la preparación de la arquitectura para futuras extensiones como redes de 25 Gbps o nuevos tipos de tráfico.

7. Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

El desarrollo de este Trabajo Fin de Grado ha permitido alcanzar satisfactoriamente los objetivos planteados al inicio del proyecto, logrando la integración funcional entre un agente de aprendizaje por refuerzo profundo (DRL) y un simulador realista de redes ópticas pasivas (PON). A través de esta integración, se ha desarrollado una herramienta útil y versátil para la investigación en la optimización dinámica de recursos en redes de acceso, combinando conocimientos avanzados en inteligencia artificial, simulación de redes y diseño de software modular.

Desde un punto de vista técnico, uno de los logros más relevantes ha sido la refactorización completa del agente DRL, originalmente implementado como un simple notebook, para transformarlo en una estructura modular y escalable basada en programación orientada a objetos. La implementación de patrones de diseño como Adapter y Strategy ha permitido desacoplar completamente el funcionamiento del agente y del simulador, posibilitando su uso conjunto o independiente según las necesidades de experimentación. Asimismo, se han implementado dos modos operativos diferenciados: uno local, mediante entornos Gymnasium, y otro integrado, en el que el simulador invoca al agente directamente.

Durante el desarrollo se han resuelto múltiples problemas técnicos significativos, entre ellos: la gestión defectuosa de entornos paralelos durante la

evaluación, el mal uso de la variable done para la finalización de episodios, y errores estructurales en la lógica de asignación de ancho de banda que impedían el correcto aprendizaje del modelo. Tras solucionarlos, se rediseñaron aspectos clave como los espacios de observación y acción, que fueron normalizados para mejorar la estabilidad del entrenamiento, y la función de recompensa, que fue completamente reformulada para reflejar con mayor fidelidad los objetivos del sistema.

La calidad del proyecto también se ha visto enriquecida gracias a la colaboración con otros estudiantes de la Universidad de Valladolid, lo que ha permitido incorporar mejoras significativas tanto en la generación de tráfico mediante procesos de Pareto más realistas, como en la evolución del simulador hacia una infraestructura capaz de representar redes de hasta 10 Gbps, incluyendo paquetes estructurados y nuevos indicadores estadísticos.

Además, el uso de tecnologías de código abierto como Python, Gymnasium, SimPy, Stable-Baselines3 y plataformas como Visual Studio Code y Overleaf ha permitido un desarrollo ágil y replicable, sin incurrir en costes económicos y cumpliendo con los estándares técnicos y académicos esperados.

7.2. Líneas futuras

Además de las posibles mejoras en la arquitectura interna y el rendimiento del agente, se identifican varias líneas de trabajo futuras que pueden ampliar las capacidades del sistema desde una perspectiva funcional, experimental y visual:

- Ampliación y mejora de la interfaz gráfica de usuario: Aunque actualmente el sistema ya cuenta con una interfaz gráfica funcional, esta ha sido diseñada de forma sencilla, con un enfoque principalmente utilitario y con poco desarrollo en cuanto a aspectos estéticos o de experiencia de usuario. Una línea de trabajo futura consiste en mejorar tanto el diseño visual como la funcionalidad de esta interfaz, de modo que no solo resulte más atractiva y profesional, sino que también permita gestionar de forma más intuitiva las distintas opciones del sistema. Estas mejoras podrían incluir, por ejemplo, la incorporación

  de controles más detallados para configurar parámetros del entorno de

simulación, visualizar métricas en tiempo real mediante gráficas interactivas, seleccionar modelos de entrenamiento previamente guardados, y comparar el rendimiento de diferentes sesiones. También se podrían añadir paneles para la gestión de logs, herramientas de exportación de resultados o asistentes paso a paso que faciliten el uso a usuarios no expertos. Todo ello contribuiría a transformar la interfaz actual en una plataforma visual potente y versátil, orientada tanto a usuarios técnicos como a perfiles más académicos o investigadores.

- Entrenamiento del agente directamente sobre el simulador: Una segunda línea de mejora, de carácter más técnico y experimental, consiste en permitir que el agente DRL pueda ser entrenado directamente utilizando el simulador realista de redes PON, en lugar de hacerlo únicamente sobre entornos simplificados como los de Gymnasium. Actualmente, esta opción no se ha explorado debido a que el simulador, al ser mucho más complejo y detallado, presenta tiempos de ejecución significativamente más largos, lo que haría que el entrenamiento fuera computacionalmente muy costoso. Sin embargo, si se aplicaran técnicas de paralelización, simplificación dinámica del entorno o utilización de hardware especializado (como GPUs o servidores), esta posibilidad se volvería factible. El entrenamiento sobre un simulador más realista permitiría obtener agentes con un comportamiento más afinado, capaces de generalizar mejor a condiciones de red reales y de adaptarse a fenómenos que no están presentes en entornos simplificados.

- Implementación de nuevos esquemas de multiplexación: Una línea de trabajo ambiciosa, pero con un alto potencial de impacto, es la incorporación de nuevos tipos de multiplexación en el simulador, más allá de la multiplexación en el dominio del tiempo (TDM) actualmente implementada. En particular, se plantea como reto la inclusión de multiplexación por división en longitud de onda (WDM), técnica utilizada en redes de nueva generación que permite asignar diferentes longitudes de onda a diferentes flujos de tráfico. Esta mejora requeriría modificaciones profundas tanto en el simulador como en el agente, ya que implicaría representar múltiples canales ópticos, modelar su comportamiento físico y modificar la acción del agente para que seleccione no solo cuánto ancho de banda asignar, sino también en qué canal o longitud de onda hacerlo. Aunque se trata de una tarea compleja, su implementación abriría las puertas a un abanico mucho más amplio de escenarios de simulación y experimentación, alineando el sistema con las tecnologías emergentes en redes ópticas avanzadas.

- Refactorización del simulador: Una línea de trabajo futura de especial interés sería la refactorización completa del simulador de redes PON, con el objetivo de mejorar su mantenibilidad, eficiencia y extensibilidad. Aunque el simulador actual es plenamente funcional y presenta una estructura clara, al haberse desarrollado de forma incremental por diferentes personas, contiene ciertos elementos que podrían optimizarse o reorganizarse para facilitar su evolución futura. Una primera acción

  en este sentido consistiría en eliminar o consolidar líneas de código innecesarias, comentarios obsoletos o fragmentos duplicados que ya no aportan valor al mantenimiento del sistema. Esta limpieza permitiría aligerar el código base, mejorar su legibilidad y reducir el esfuerzo requerido para incorporar nuevas funcionalidades. Además, una mejora

  clave sería la formalización de una arquitectura basada en herencia para la clase OLT. Aunque actualmente ya existen varias clases OLT que implementan distintas estrategias de asignación de ancho de banda, la selección entre ellas requiere modificar directamente el código fuente. La propuesta consiste en crear una clase abstracta o base común para la OLT, que defina una interfaz genérica para la toma de decisiones, y de la cual hereden las distintas implementaciones específicas. Esto permitiría unificar el acceso a las distintas variantes y facilitar su uso mediante instanciación dinámica o configuración externa, sin necesidad de alterar el código del simulador. Además, esta separación clara de responsabilidades mejoraría la extensibilidad del sistema, permitiendo añadir fácilmente nuevas estrategias, algoritmos inteligentes o esquemas híbridos en el futuro, todo ello siguiendo principios sólidos de diseño como la abstracción, la reutilización y el Open/Closed Principle.

Bibliografía

[1] Victor Herrezuelo Paredes. Implementación de un simulador de redes de acceso ópticas pasivas en Python, 2024. [Obtenido por medio del propio autor].

[2] David Pérez Moreno. Aprendizaje por refuerzo en redes ópticas pasivas. https://github.com/dpm1002/TFG-GII-Aprendizaje-por-refuerzo-Reinforcement-Learning-en-redes-opticas-pasivas, 2024.

[3] Wikipedia. Aprendizaje por refuerzo. https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo, 2024. [Consultado en Abril 2025].

[4] Lv Zhong. Comparison of Q-learning and SARSA Reinforcement Learning Models on Cliff Walking Problem. In *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*, pages 207–213. Atlantis Press, 2024.

[5] Jürgen Schmidhuber. Deep Learning in Neural Networks: An Overview. *CoRR*, abs/1404.7828, 2014.

[6] OpenAI. Proximal Policy Optimization. https://spinningup.openai.com/en/latest/algorithms/ppo.html, 2018.

[7] Wikipedia. Red Óptica Pasiva. https://es.wikipedia.org/wiki/Red_%C3%B3ptica_pasiva, 2023. [Consultado en Abril 2025].

[8] Tomas Horvath, Petr Munster, Vaclav Oujezsky, and Ning-Hai Bao. Passive Optical Networks Progress: A Tutorial. *Electronics*, 9(7), 2020.

[9] Microsoft. Github. https://github.com/, 2025.

[10] Overleaf. Overleaf. https://es.overleaf.com/, 2025.

[11] Community. Pyinstaller manual. https://pyinstaller.org/en/stable/, 2025.

[12] Farama Foundation. Gymnasium Documentation. https://gymnasium.farama.org/, 2022.

[13] Community. Stable baselines docs. https://stable-baselines.readthedocs.io/en/master/index.html#, 2021.

[14] Community. Documentation for Simpy. https://simpy.readthedocs.io/en/latest/contents.html#, 2024.

[15] Python. Python documentation. https://docs.python.org/3/library/tkinter.html, 2025.

[16] Santiago Rodríguez Diez. simDeepPON-RL — Deep Reinforcement Learning para la Planificación y Operación Autónoma de Redes XGS-PON: Implementación e Integración con un Simulador en Python. https://github.com/SantiagoRD18/TFG_GII_DRL_Redes_GS-PON, 2025.