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

Documentación Técnica

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

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Este apéndice recoge el Plan de Proyecto Software seguido durante el desarrollo del sistema "simDeepPON-RL". Su objetivo es analizar los aspectos clave relacionados con la planificación, la viabilidad económica y la adecuación legal del proyecto, evaluando tanto su sostenibilidad como su coherencia con los recursos disponibles.

En primer lugar, se presenta la planificación temporal del proyecto, basada en una metodología ágil y adaptada al trabajo individual. A continuación, se analiza la viabilidad económica del desarrollo, considerando tanto los recursos utilizados como los costes asociados. Por último, se incluye un estudio de viabilidad legal, en el que se examina el cumplimiento de los aspectos normativos y éticos aplicables al uso del software, datos y librerías empleados.

A.2. Planificación temporal

Durante el desarrollo del proyecto se planteó inicialmente el uso de una metodología ágil basada en Scrum [1], adaptada al trabajo individual. Con esta idea, se definieron una serie de sprints y se crearon múltiples issues para organizar las tareas y marcar objetivos a corto plazo. Sin embargo, a lo largo del desarrollo, la planificación temporal propuesta no se cumplió estrictamente, debido a la aparición de problemas técnicos imprevistos que requirieron reorganizar las prioridades, así como a la naturaleza iterativa de

muchas de las fases del proyecto.

Pese a ello, esta organización inicial resultó útil para establecer un marco de referencia que permitió mantener una visión general del avance del proyecto. A continuación se detallan las fases principales en las que puede dividirse el desarrollo, aunque algunas de ellas se solaparon parcialmente o se ejecutaron de forma iterativa.

1. **Investigación inicial y análisis de código existente**
   El proyecto comenzó con una fase de documentación y análisis orientada a adquirir los conocimientos necesarios sobre redes ópticas pasivas (PON), dado que esta tecnología no se aborda en profundidad durante el grado. Para ello se consultaron artículos científicos, documentos técnicos y bibliografía especializada, además de realizar diversas tutorías con los directores del trabajo. Paralelamente, se examinó en detalle el código fuente de los dos proyectos heredados —el agente DRL y el simulador— con el fin de comprender su estructura interna, evaluar su grado de funcionalidad y detectar posibles carencias o puntos de mejora.

2. **Diseño de la nueva arquitectura del sistema**
   Una vez comprendidos los sistemas de partida, se abordó el rediseño completo de la arquitectura del agente. Se propuso una estructura modular orientada a objetos que favoreciera la separación de responsabilidades, la escalabilidad y la mantenibilidad. Para ello se hizo uso de patrones de diseño como Adapter y Strategy, que permitieron integrar de forma flexible el simulador externo y definir distintos modos de operación del agente (entrenamiento local o integración con simulador). Esta fase también incluyó la planificación inicial de los nuevos módulos y clases necesarias, así como el diseño conceptual de la comunicación entre componentes.

3. **Refactorización del agente DRL y desarrollo parcial**
   A partir del diseño anterior, se comenzó con la refactorización del código original del agente, que estaba implementado en un único notebook sin modularización. Se extrajeron funcionalidades en clases y módulos independientes, como Agente, GestorModelos o Graficador. Se implementaron mejoras clave como la gestión correcta de entornos, el soporte básico para entrenamiento y evaluación, y la adaptación inicial al nuevo flujo de ejecución. Esta fase permitió disponer de

una primera versión funcional del agente, aunque aún limitada y sin resolver problemas estructurales en el aprendizaje.

4. **Solución de errores**
   Durante esta etapa se abordaron los problemas más complejos del proyecto. A pesar de que el sistema parecía operar visualmente, el agente no aprendía de forma efectiva, ya que las acciones generadas eran casi nulas y no influían realmente en el entorno. Esto evidenció un fallo profundo en la lógica interna del entorno de Gymnasium, que asignaba tráfico de forma determinista sin basarse en las decisiones del agente. Se reescribió el procesamiento del tráfico, se redefinió la función de recompensa (incluyendo penalización por colas, eficiencia en el uso del ancho de banda y cumplimiento de límites), y se normalizaron tanto las observaciones como las acciones para estabilizar el entrenamiento.

5. **Finalización del desarrollo del agente**
   Una vez solucionados los errores críticos, se completó la implementación del agente, añadiendo nuevas clases (BaseAgent, LocalAgent, SimAgent) para soportar distintos modos de ejecución. Se mejoró la evaluación, se corrigieron problemas con los entornos paralelos y se aseguraron los ciclos de entrenamiento. Además, se realizaron numerosos ajustes en los hiperparámetros, y se probó el comportamiento del modelo sobre distintos entornos, incluyendo tráfico desbalanceado o con distintos acuerdos de nivel de servicio, con buenos resultados.

6. **Integración con el simulador y pruebas de funcionamiento**
   Esta fase se centró en conectar el agente con el simulador PON heredado, adaptando ambas partes para que pudieran comunicarse mediante una interfaz definida. El agente fue convertido en un "módulo de decisión"dentro de la OLT, permitiendo que sus predicciones influyeran directamente en la asignación de ancho de banda. Se realizaron múltiples pruebas para validar que el comportamiento era coherente, que el sistema funcionaba de forma sincrónica y que las decisiones del agente se aplicaban correctamente durante la simulación. También se adaptaron los mecanismos de generación de tráfico realista desarrollados por compañeros colaboradores.

7. **Diseño de la interfaz gráfica, empaquetado y finalización del código y la memoria**
   Para mejorar la usabilidad del sistema, se desarrolló una interfaz gráfica de usuario (GUI) utilizando Tkinter[2], que permite ejecutar fácilmente tanto el agente como el simulador, seleccionar modelos, visualizar

resultados y configurar parámetros de simulación. Se reestructuraron los paquetes, se adaptaron las importaciones y se creó un punto de entrada único (main.py) para lanzar la aplicación desde un ejecutable. Finalmente, se empleó PyInstaller[3] para generar un .exe autónomo que encapsula todo el sistema. La memoria del proyecto, incluyendo este mismo apéndice, se fue redactando progresivamente durante todas las fases anteriores, en paralelo al desarrollo técnico.

A.3. Estudio de viabilidad

Viabilidad económica

Este análisis permite evaluar si los recursos necesarios para llevar a cabo el proyecto están disponibles y si los costes asociados resultan asumibles dentro del contexto en el que se desarrolla.

En este apartado se estudia la viabilidad económica del proyecto atendiendo a tres factores clave: el software utilizado, los recursos hardware requeridos y la implicación del personal responsable. Dado que el proyecto se enmarca en el ámbito académico, se prioriza el uso de herramientas gratuitas y recursos personales, lo que permite minimizar los costes sin comprometer la funcionalidad ni los objetivos técnicos.

Software

El desarrollo del proyecto se ha basado en herramientas y librerías de código abierto, lo que ha permitido evitar cualquier tipo de coste en licencias o suscripciones. Todas las tecnologías empleadas son de uso gratuito, ampliamente documentadas y compatibles entre sí:

- Lenguaje de programación: Python, de código abierto y con una comunidad muy activa.
- Librerías principales:
  - Gymnasium: librería para el diseño de entornos de simulación, sucesora de OpenAI Gym.
  - Stable-Baselines3: implementación de algoritmos de aprendizaje por refuerzo.

  - SimPy: utilizada en el simulador de redes para modelar procesos concurrentes.
  - Otras como NumPy, Matplotlib o pandas también han sido utilizadas, todas gratuitas.
- Entorno de desarrollo: Visual Studio Code, editor ligero, multiplataforma y gratuito.
- Documentación: Overleaf, entorno online gratuito para trabajar con LaTeX.

Gracias al uso de software libre, el coste económico en este apartado ha sido nulo, lo que favorece la viabilidad del proyecto en contextos académicos o de investigación.

Hardware

El proyecto ha sido desarrollado y ejecutado en un ordenador personal con las siguientes características:

- Procesador: Intel Core i7-11700K
- Memoria RAM: 16GB
- Almacenamiento: 1TB SSD
- Tarjeta Gráfica: Gigabyte RTX 3070
- Sistema operativo: Windows 11, aunque el entorno es multiplataforma (compatible también con Linux).

Con un coste aproximado de 1430€

Además, se han utilizado también los siguientes periféricos:

- 2 x Monitor Samsung
- Teclado + Ratón Logitech

Con un coste aproximado de 450€

Todo esto suma un total de 1880€ que, estimando una vida útil de 6 años, nos saldría un precio amortizado por hora de uso tal que:

$$\frac{1880}{6 * 365 * 24} = \frac{1880}{52560} = \boxed{0{,}03577 €/h}$$

Sabiendo además que sumando las horas de trabajo con las horas de ejecución en las que el ordenador se dedicaba únicamente a entrenar el modelo, nos da un total aproximado de 540 horas. Podemos calcular el coste total del hardware amortizado de la siguiente forma:

$$0{,}03577 * 450 = \boxed{16{,}09 €}$$

No se ha requerido acceso a servidores externos ni recursos en la nube, por lo que no se han generado costos adicionales por infraestructura computacional.

Personal

Por último, calcularemos el coste de mi propio trabajo, teniendo en cuenta el sueldo que cobraría de forma realista. Para ello, partimos de la siguiente información:

- Se estima un salario bruto anual de referencia de **20.000 €**, valor habitual para un perfil junior en el sector [4].
- La empresa debe abonar aproximadamente un **32 % adicional** en concepto de cotizaciones a la Seguridad Social, incluyendo contingencias comunes, desempleo, formación, FOGASA, accidentes y el mecanismo de equidad intergeneracional.
- Se asume una jornada laboral estándar de **1.800 horas/año**.
- El proyecto ha supuesto una dedicación personal aproximada de **350 horas**.

Y a continuación realizamos los siguientes cálculos:

1. Cálculo del coste total anual para la empresa:

$$\text{Coste total anual} = \text{Salario bruto} + (\text{Salario bruto} \times \text{Cotización})$$

$$\text{Coste total anual} = 20{,}000 + (20{,}000 \times 0{,}32) = 20{,}000 + 6{,}400 = \boxed{26{,}400\ €}$$

2. Cálculo del coste por hora

$$\text{Coste por hora} = \frac{\text{Coste total anual}}{\text{Horas trabajadas al año}} = \frac{26{,}400}{1{,}800} \approx \boxed{14{,}67\ €/\text{hora}}$$

3. Cálculo del coste total atribuible al proyecto

$$\text{Coste total proyecto} = \text{Coste por hora} \times \text{Horas dedicadas}$$

$$\text{Coste total proyecto} = 14{,}67 \times 350 = \boxed{5134,50\ €}$$

Bajo estos supuestos, el coste total estimado del personal asciende a aproximadamente **5134,50 euros**, considerando tanto el salario bruto como las obligaciones sociales de la empresa.

Con todos estos costes desglosados, el resumen sería el siguiente:

Concepto	**Cuantía**
Software	0 €
Hardware	16,09 €
Personal	5134,50 €
Total	**5150,59 €**

Tabla A.1: Resumen de costes del proyecto

Viabilidad legal

Desde el punto de vista legal, el proyecto presenta una viabilidad plenamente favorable, ya que cumple con todos los requisitos normativos y éticos aplicables a su desarrollo en el contexto académico. A continuación se analizan los aspectos más relevantes:

- Titularidad del código base El proyecto se ha basado en dos trabajos anteriores realizados por compañeros de la Universidad de Burgos y la Universidad de Valladolid, cuyos códigos han sido reutilizados, refactorizados y ampliados.

  Ambos proyectos fueron desarrollados como Trabajos Fin de Grado y puestos a disposición del autor con consentimiento expreso, lo que garantiza la legitimidad en la reutilización del código original. En

cualquier caso, se ha realizado una referencia explícita a los autores originales en la memoria y se ha respetado su autoría intelectual, cumpliendo con la legislación sobre propiedad intelectual (Real Decreto Legislativo 1/1996)[5].

- Licencias de software utilizadas Todas las herramientas, librerías y plataformas empleadas en el proyecto están sujetas a licencias libres o de código abierto. En particular:

  Python y sus librerías (Gymnasium, Stable-Baselines3, SimPy, etc.) están distribuidas bajo licencias como MIT, Apache 2.0 o similares[6], que permiten su uso, modificación y redistribución incluso en contextos comerciales.

  Visual Studio Code y Overleaf también permiten su uso libre en entornos académicos.

  Por tanto, el uso de estas tecnologías no impone restricciones legales sobre el desarrollo ni la distribución del proyecto resultante.

- Protección de datos y ética El sistema desarrollado no trata datos personales ni sensibles, ya que se basa en simulaciones de tráfico de red artificialmente generadas. No se ha utilizado información de usuarios reales ni se ha accedido a sistemas de producción, por lo que no se incurre en riesgos legales relacionados con el Reglamento General de Protección de Datos (RGPD)[7] ni con la Ley Orgánica de Protección de Datos y Garantía de los Derechos Digitales (LOPDGDD)[8].

  Además, el enfoque del proyecto es puramente técnico y no conlleva implicaciones éticas sensibles relacionadas con la toma de decisiones automáticas sobre personas o entornos reales.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apartado se presentan los requisitos del sistema desarrollado, con el objetivo de definir de forma clara qué se espera del software y cómo debe comportarse. La estructura sigue un enfoque progresivo: empezaremos por los objetivos generales del sistema e iremos profundizando hasta llegar a una descripción detallada de los requisitos tanto funcionales como no funcionales.

Es importante aclarar que los requisitos aquí recogidos están centrados exclusivamente en el producto software, dejando de lado los aspectos relacionados con la gestión o planificación del proyecto. Lo que buscamos es identificar las funciones que debe ofrecer el sistema, cómo deben interactuar los distintos componentes, y qué expectativas debe cumplir desde el punto de vista técnico y del usuario final.

Este apartado servirá como base para validar si el sistema desarrollado cumple con lo previsto y también como referencia para futuras ampliaciones o mantenimientos.

B.2. Objetivos generales

Aunque profundizaremos y detallaremos los requisitos concretos en el siguiente punto, primero veremos cuales son los principales bojetivos del producto final completo:

- **Integración funcional del sistema completo.** Conseguir una integración efectiva entre el simulador y el agente, asegurando que funcionen correctamente como un sistema unificado donde el simulador utiliza las predicciones del agente para optimizar el tráfico.

- **Ejecución independiente de los componentes.** Permitir que tanto el simulador como el agente puedan ejecutarse de forma autónoma, facilitando su uso, prueba y evolución por separado y permitiendo así el entrenamiento del agente en un entorno separado.

- **Refactorización del código existente.** Mejorar la estructura interna del código de ambos componentes, eliminando redundancias y facilitando su comprensión mediante una organización más clara.

- **Modularización del sistema.** Diseñar la arquitectura del proyecto siguiendo principios de modularidad, de modo que cada componente pueda ser reutilizado, reemplazado o ampliado fácilmente.

- **Flexibilidad en la incorporación de modelos de IA.** Facilitar el reemplazo o la modificación del agente DRL, permitiendo intercambiar algoritmos dentro de Stable-Baselines3 u otras librerías sin necesidad de reescribir el simulador.

- **Mejorar la mantenibilidad y usabilidad del software.** Asegurar que el sistema sea fácil de mantener y extender, proporcionando una documentación técnica clara y una interfaz de uso comprensible para desarrolladores e investigadores.

B.3. Catálogo de requisitos

A continuación procederé a listar los diferentes requisitos concretos del sistema, tanto requisitos funcionales como no funcionales.

Requisitos Funcionales

RF-1 Ejecución del agente en local
El sistema debe permitir ejecutar el agente en modo local, utilizando su propio entorno y una simulación de tráfico interna.

RF-2 Creación de modelos
El sistema debe permitir crear modelos de entrenamiento utilizando cualquiera de los algoritmos disponibles en la librería Stable-Baselines3.

RF-3 Gestión de modelos entrenados
El sistema debe permitir guardar modelos entrenados en ficheros y cargarlos posteriormente para su reutilización.

RF-4 Entrenamiento del modelo
El sistema debe permitir configurar los parámetros de entrenamiento y ejecutar el entrenamiento del modelo sobre el entorno simulado.

RF-5 Evaluacion del modelo
El sistema debe permitir ejecutar simulaciones durante un tiempo configurable para evaluar el rendimiento del modelo entrenado.

RF-6 Modos de ejecución del simulador
El sistema debe permitir ejecutar el simulador con diferentes modos de decisión en la OLT: mediante lógica local o a través de una instancia del agente.

RF-7 Visualizacion de resultados de evaluación
El sistema debe permitir generar gráficos o métricas que muestren el comportamiento del agente y los resultados de la evaluación.

RF-8 Configuración del entorno
El sistema debe permitir modificar ciertos parámetros del entorno simulado (como número de ONTs, ancho de banda disponible, duración de la simulación, etc.).

Requisitos no Funcionales

RNF-1 Modularidad del agente
El agente debe estar implementado de forma modular, permitiendo una separación clara entre sus diferentes componentes funcionales.

RNF-2 Encapsulamiento del entrenamiento y la ejecución
El sistema debe permitir la creación de un objeto único del agente, que agrupe todas las variables y métodos necesarios para el entrenamiento y ejecución del modelo.

RNF-3 Separación de la lógica de gestión de modelos
La creación, guardado y carga de los modelos debe estar contenida en un módulo independiente, separado del resto de la lógica del agente.

RNF-4 Organización orientada a objetos
La estructura del agente debe estar organizada mediante clases que reflejen de forma clara las distintas responsabilidades del sistema.

RNF-5 Soporte ejecución en local
El agente debe poder ejecutarse de forma autónoma en local, a través de la ejecucion de un Jupyter Notebook, sin necesidad de conexión con el simulador.

RNF-6 Soporte para ejecución desde el simulador
El agente debe poder ser invocado de forma externa por el simulador para participar en la toma de decisiones de la OLT mediante una interfaz que gestione el intercambio de datos entre ambos sistemas.

RNF-7 Modularidad de la OLT del simulador
La implementación de la OLT en el simulador debe estar separada en módulos independientes que permitan su reutilización y modificación.

RNF-8 Inetrcambiabilidad de la OLT
El simulador debe permitir modificar fácilmente el tipo de OLT utilizada, utilizando una clase abstracta y herencia para definir diversas OLTs que utilicen diferentes estrategias de toma de decisiones

RNF-9 Reutilización de componentes
Los componentes principales (agente, entorno, modelos, simulador) deben estar diseñados para poder ser reutilizados en otros proyectos o escenarios con mínima modificación.

RNF-10 Documentación del código
El código debe estar documentado adecuadamente mediante comentarios para facilitar su comprensión y mantenimiento.

RNF-11 Facilidad de integración
El sistema debe facilitar su integración con otros módulos externos, como nuevas versiones del simulador o agentes alternativos.

B.4. Especificación de requisitos

CU-1	**Ejecutar agente en local**
Versión	1.0
Autor	Santiago Rodríguez Diez
Requisitos asociados	**RF-1**, **RNF-5**
Descripción	El usuario ejecuta el agente desde un entorno local para entrenar o evaluar el modelo
Precondición	El agente debe estar correctamente instalado y el entorno configurado
Acciones	1. Ejecutar la interfaz gráfica 2. Seleccionar la opción de ejecutar el agente 3. Seleccionar la opción de entrenamiento o evaluación del agente 4. Proporcionar los valores para las variables correspondientes (o dejar los valores por defecto) 5. Comienza la ejecución del agente
Postcondición	El agente se ha ejecutado sin problemas
Importancia	Alta

Tabla B.1: CU-1 Ejecutar agente en local.

CU-2	**Entrenar modelo nuevo**
Versión	1.0
Autor	Santiago Rodríguez Diez
Requisitos asociados	**RF-2**, **RF-4**, **RNF-1**, **RNF-2**
Descripción	El usuario entrena un modelo de agente desde cero utilizando uno de los algoritmos disponibles y se almacena el modelo
Precondición	El agente debe estar correctamente instalado y el entorno configurado
Acciones	1. Ejecutar la interfaz gráfica 2. Seleccionar la opción de ejecutar el agente 3. Seleccionar la opción de entrenar de agente 4. Proporcionar los valores para las variables correspondientes (o dejar los valores por defecto) 5. Especificar un nombre para el fichero del nuevo modelo entrenado 6. Comienza el entrenamiento del agente
Postcondición	El agente ha creado un nuevo fichero dentro de la carpeta models con el nombre proporcionado anteriormente
Importancia	Alta

Tabla B.2: CU-2 Entrenar modelo nuevo.

CU-3	**Evaluar un modelo entrenado**
Versión	1.0
Autor	Santiago Rodríguez Diez
Requisitos asociados	**RF-3**, **RF-5**, **RNF-1**, **RNF-2**
Descripción	El usuario reliza la evaluación de un modelo ya entrenado anteriormente
Precondición	El agente debe estar correctamente instalado y el entorno configurado y debe existir un modelo entrenado en la carpeta models
Acciones	1. Ejecutar la interfaz gráfica 2. Seleccionar la opción de ejecutar el agente 3. Seleccionar la opción de evaluación de agente 4. Proporcionar los valores para las variables correspondientes (o dejar los valores por defecto) 5. Especificar un nombre para cargar el modelo entrenado 6. Comienza la evaluación del agente
Postcondición	El agente se ha ejecutado correctamente y ha aportado los resultados de la evaluación
Importancia	Alta

Tabla B.3: CU-3 Evaluar un modelo entrenado.

CU-4	**Visualizar resultados evaluacion**
Versión	1.0
Autor	Santiago Rodríguez Diez
Requisitos asociados	**RF-7**
Descripción	El usuario visualiza las gráficas de los resultados obtenidos tras la evaluación
Precondición	El agente debe estar correctamente instalado y el entorno configurado y debe existir un modelo entrenado en la carpeta models
Acciones	1. Ejecutar la interfaz gráfica 2. Seleccionar la opción de ejecutar el agente 3. Seleccionar la opción de evaluación de agente 4. Proporcionar los valores para las variables correspondientes (o dejar los valores por defecto) 5. Especificar un nombre para cargar el modelo entrenado 6. Comienza la evaluación del agente 7. Al terminar la evaluación, seleccionar la opción de visualizar los resultados 8. Comprobar que se pueden ver las gráficas de cualquier ONT
Postcondición	El agente se ha ejecutado correctamente y ha permitido visualizar las gráficas de tráfico de entrada y salida y del estado de las colas, para cualquier ONT seleccionada
Importancia	Media

Tabla B.4: CU-4 Visualizar resultados evaluacion.

CU-5	**Configuración del entorno**
Versión	1.0
Autor	Santiago Rodríguez Diez
Requisitos asociados	**RF-8**, **RNF-2**
Descripción	El usuario puede configurar fácilmente los parámetros de la red y del entorno de gymnasium
Precondición	El agente debe estar correctamente instalado
Acciones	1. Ejecutar la interfaz gráfica 2. Seleccionar la opción de ejecutar el agente 3. Seleccionar cualquiera de las opciones para el agente 4. Introducir valores para los diferentes parámetros de la red, como el número de ONTs, el tiempo de ciclo, el acuerdo de nivel de servicio de las ONTs... 5. Ejecutar la opción seleccionada
Postcondición	El agente se ejecuta perfectamente con diferentes parámetros y estos se ven reflejados en la ejecución
Importancia	Media

Tabla B.5: CU-5 Configuración del entorno.

CU-6	**Ejecución del simulador con DRL**
Versión	1.0
Autor	Santiago Rodríguez Diez
Requisitos asociados	**RF-6**, **RNF-6**, **RNF-7**, **RNF-8**
Descripción	El usuario puede ejecutar el simulador utilizando el agente DRL para tomar las decisiones
Precondición	El agente y el simulador deben estar correctamente instalados y configurados. Además, debe existir un modelo ya entrenado en el directorio models
Acciones	1. Ejecutar la interfaz gráfica 2. Seleccionar la opción de ejecutar el simulador 3. Seleccionar la opción de ejecución DRL 4. Introducir el nombre de un modelo de PPO ya entrenado y almacenado en el directorio correspondiente 5. Comprobar la correcta ejecución del simulador
Postcondición	El simulador se ejecuta correctamente y muestra en la ventana de powershell el progreso y los resultados de la simulación
Importancia	Media

Tabla B.6: CU-6 Ejecución del simulador con DRL.

CU-7	**Ejecución del simulador en modo normal**
Versión	1.0
Autor	Santiago Rodríguez Diez
Requisitos asociados	**RF-6**, **RNF-7**, **RNF-8**
Descripción	El usuario puede ejecutar el simulador sin política de toma de decisiones, sin utilizar ningún modelo de aprendizaje automático
Precondición	El simulador debe estar correctamente instalado y configurado.
Acciones	1. Ejecutar la interfaz gráfica 2. Seleccionar la opción de ejecutar el simulador 3. Seleccionar la opción de ejecución normal 4. Comprobar la correcta ejecución del simulador
Postcondición	El simulador se ejecuta correctamente y muestra en la ventana de powershell el progreso y los resultados de la simulación
Importancia	Media

Tabla B.7: CU-7 Ejecución del simulador normal.

CU-8	**Ejecución del simulador con DNN**
Versión	1.0
Autor	Santiago Rodríguez Diez
Requisitos asociados	**RF-6**, **RNF-7**, **RNF-8**
Descripción	El usuario puede ejecutar el simulador utilizando el modelo DNN para la toma de decisiones
Precondición	El simulador debe estar correctamente instalado y configurado.
Acciones	1. Ejecutar la interfaz gráfica 2. Seleccionar la opción de ejecutar el simulador 3. Seleccionar la opción de ejecución DNN 4. Comprobar la correcta ejecución del simulador
Postcondición	El simulador se ejecuta correctamente y muestra en la ventana de powershell el progreso y los resultados de la simulación
Importancia	Media

Tabla B.8: CU-8 Ejecución del simulador con DNN.

Apéndice C

Especificación de diseño

C.1. Introducción

Este apéndice tiene como objetivo documentar las decisiones de diseño adoptadas durante el desarrollo del sistema, desde un punto de vista técnico y estructural. A lo largo del proyecto se ha trabajado con un enfoque modular y orientado a objetos, siguiendo principios de diseño que favorecen la claridad, la reutilización de componentes y la escalabilidad del sistema. Esta fase de diseño ha sido clave para garantizar que el desarrollo posterior pudiera realizarse de forma ordenada, mantenible y fácilmente ampliable.

El diseño del sistema ha evolucionado de forma iterativa, en paralelo al análisis de los componentes heredados y a las pruebas funcionales realizadas. Se han tomado decisiones no solo con el objetivo de hacer que el sistema funcionara correctamente, sino también buscando que su estructura fuera comprensible, flexible y alineada con las buenas prácticas de la ingeniería del software.

En este sentido, el uso de clases bien diferenciadas, la separación de responsabilidades entre módulos, la integración de patrones de diseño, y la normalización de flujos de datos y procesos han sido aspectos clave en la fase de diseño. Estas decisiones no solo han facilitado la implementación técnica del sistema, sino que también han permitido su empaquetado, distribución y futura reutilización en otros contextos o proyectos.

A lo largo de este apéndice se describen los principales aspectos de este diseño, proporcionando una visión global de cómo está organizado internamente

el sistema y cuáles son sus componentes más relevantes desde el punto de vista estructural y funcional.

C.2. Diseño de datos

El diseño de datos del sistema SimDeepPON-RL ha sido un aspecto clave para garantizar la correcta interacción entre el agente de aprendizaje por refuerzo profundo y el entorno simulado de red óptica pasiva. A lo largo del desarrollo, se han definido diferentes estructuras de datos para representar de forma coherente y eficiente los elementos involucrados en el entrenamiento, evaluación y simulación del sistema.

- Las observaciones que recibe el agente están formadas por vectores que representan el estado de las colas de las distintas ONTs, normalizados respecto a un valor máximo predefinido. Esta representación vectorial permite al modelo trabajar con magnitudes relativas en lugar de valores absolutos, lo cual facilita el proceso de aprendizaje y mejora la estabilidad numérica del sistema. Estas observaciones son devueltas por el entorno tras cada acción ejecutada y forman parte del bucle de entrenamiento.

- Las acciones generadas por el agente consisten en vectores continuos, también normalizados entre 0 y 1, que representan la proporción de ancho de banda que debe asignarse a cada ONT en el siguiente ciclo de simulación. Estas acciones son transformadas internamente por el entorno, multiplicándolas por el máximo ancho de banda disponible (B_max) para obtener los valores reales que se utilizarán en la asignación.

- La recompensa se calcula de forma continua en cada paso del entorno, como una combinación lineal ponderada de diferentes métricas: evolución de las colas, eficiencia en el uso del ancho de banda, y penalización por superar el límite de asignación impuesto por la OLT. Estas recompensas se representan como valores escalares y se utilizan para guiar el proceso de entrenamiento del modelo.

- Durante la ejecución del sistema, también se generan métricas y resultados adicionales que se almacenan en listas o arrays para su posterior análisis o visualización. Entre estas métricas se encuentran el uso total de ancho de banda por ciclo, la evolución de la longitud de las colas en cada ONT, o el rendimiento global del modelo. Estas estructuras

son utilizadas por los módulos de graficado, que permiten visualizar los resultados en forma de gráficos mediante la librería matplotlib.

- Por último, los modelos entrenados se almacenan utilizando el sistema de serialización de la librería Stable-Baselines3, en archivos comprimidos .zip que contienen tanto los pesos de la red neuronal como los hiperparámetros utilizados. Estos ficheros permiten recuperar el estado completo del agente para su posterior evaluación o reutilización en nuevas simulaciones.

En conjunto, el diseño de datos se ha orientado a facilitar la interoperabilidad entre componentes, la eficiencia durante la simulación y la trazabilidad de los resultados, manteniendo estructuras simples pero suficientes para cubrir las necesidades del sistema.

C.3. Diseño arquitectónico

La arquitectura del sistema desarrollado sigue una estructura modular, organizada en distintas carpetas funcionales y con una clara separación entre las dos partes principales del proyecto: el agente de aprendizaje por refuerzo y el simulador de red óptica pasiva (PON). En la raíz del proyecto se encuentra un archivo principal main.py, que actúa como interfaz gráfica de usuario (GUI) y punto de entrada general, permitiendo ejecutar cualquiera de los modos disponibles: entrenamiento o evaluación en local, o simulación con el agente.

A nivel estructural, el sistema se organiza en dos directorios principales:

1. agente/: contiene todo lo relacionado con el desarrollo, entrenamiento y evaluación del agente.
2. simulador/: contiene el motor de simulación y las clases necesarias para modelar el comportamiento de la red y de los elementos que la componen.

Cada una de estas carpetas incluye también su propio main.py, que permite ejecutar de forma independiente sus funcionalidades principales. No obstante, el foco del diseño arquitectónico ha estado centrado especialmente en la parte del agente, ya que ha sido el componente más profundamente rediseñado, refactorizado y ampliado durante el desarrollo del proyecto.

Arquitectura del agente

El diseño del agente se organiza en tres submódulos principales:

1. classes/: contiene la definición de los agentes. La clase *BaseAgent* actúa como una clase base (semi-abstracta), proporcionando atributos y métodos comunes como *create_model*, *make_env*, *save_model*, etc. De esta clase derivan dos implementaciones concretas:
   - *LocalAgent*, que representa al agente ejecutado de forma autónoma en su propio entorno simulado mediante Gymnasium.
   - *SimAgent*, diseñado para integrarse directamente con el simulador externo, a través de una interfaz de comunicación.
2. modulos/: contiene módulos auxiliares que encapsulan funcionalidades específicas:
   - *model_manager.py*: módulo dedicado a la gestión de modelos. Permite crear, guardar y cargar modelos entrenados utilizando la librería Stable-Baselines3. Incluye también configuraciones por defecto para hiperparámetros como POLICY, N_STEPS, LEARNING_RATE, etc.
   - *plotter.py*: módulo encargado de procesar y visualizar gráficamente los resultados obtenidos en las simulaciones y entrenamientos. Permite representar métricas como tráfico de entrada/salida, uso de ancho de banda, evolución de colas, etc.
3. entornos/: define entornos personalizados compatibles con Gymnasium. En concreto, el entorno *RedesOpticasEnv* simula el comportamiento de una red PON de forma simplificada para ser usado durante el entrenamiento del agente. Contiene métodos clave como *step()*, *reset()*, *get_obs()*, y atributos que representan el estado interno de la red (colas, tráfico, ancho de banda, etc.).

Todos estos elementos podemos visualizarlos en el diagrama de clases expuesto en la figura **??** El *main.py* del agente actúa como coordinador: instancia las clases de agente necesarias (*LocalAgent* o *SimAgent*), llama a los módulos de entrenamiento, evaluación o graficado según el modo de ejecución, y gestiona los parámetros generales de funcionamiento.

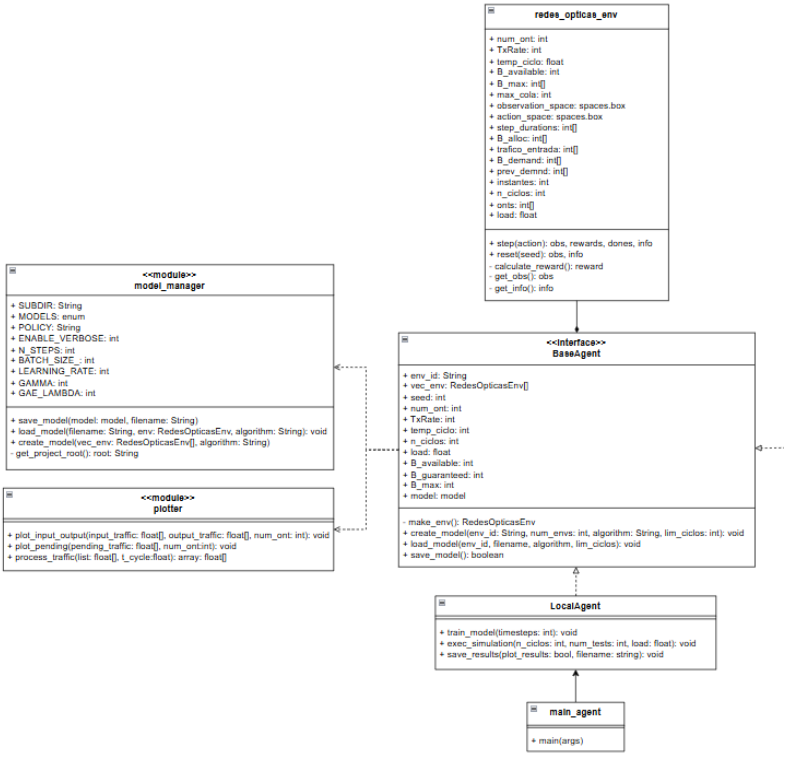

Figura C.1: Diagrama de clases del Agente

Comunicación agente–simulador

Uno de los elementos clave de esta arquitectura es la integración entre el agente y el simulador. Para ello, se ha diseñado un canal de comunicación estructurado, basado en dos componentes:

- SimAgent (dentro del agente), que implementa el método *get_prediction()* y se encarga de recibir observaciones del simulador, transformarlas en acciones y devolverlas.

- InterfazAgente (dentro del simulador), que actúa como puente entre ambos sistemas. Esta clase realiza la conversión de datos entre el formato del simulador y el formato que espera el agente (*convert_data_from_sim*, *convert_data_from_agent*) y encapsula el uso del objeto *SimAgent*.

Podemos visualizar la estructura de esta conexión en el diagrama de clases en la figura C.2. Gracias a esta arquitectura desacoplada, el simulador puede enviar el estado actual de la red al agente, obtener una predicción y aplicar la decisión correspondiente sin conocer la implementación interna del agente.

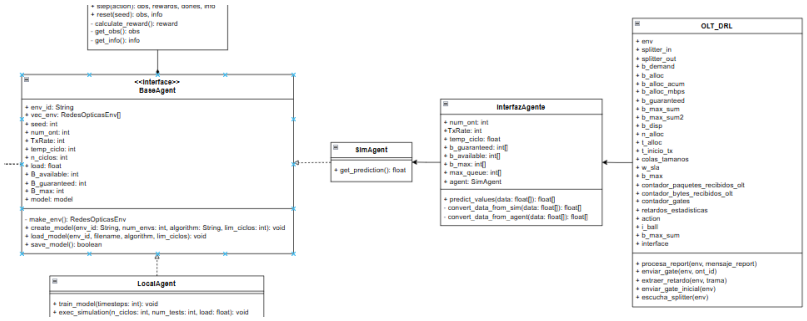

Figura C.2: Diagrama de clases la conexión agente-simulador

Arquitectura del simulador

El simulador mantiene la estructura heredada del trabajo original, basada en SimPy para la simulación de eventos discretos. La OLT se modela mediante una jerarquía de clases, entre las cuales destaca *OLT_DRL*, que es la encargada de comunicarse con el agente externo. Esta clase invoca a la InterfazAgente para obtener decisiones durante la simulación y actuar en consecuencia.

Existen además otras clases de OLT como *OLT.py* -que asigna como ancho de banda el contenido de la cola, sin tomar decisiones criticas- u *OLT_DNN.py* -que utiliza un modelo DNN para la toma de decisiones-, que permiten simular distintos modos de toma de decisiones, facilitando la comparación entre diferentes estrategias y algoritmos.

C.4. Diseño procedimental

El diseño procedimental del sistema se centra en describir el comportamiento dinámico de sus principales componentes, así como el flujo de ejecución que se activa a partir de la interacción del usuario con la interfaz gráfica. Este apartado explica cómo se orquesta internamente la creación de agentes, la ejecución de entrenamientos y simulaciones, y la gestión de los resultados en función de las decisiones del usuario.

La ejecución del sistema comienza cuando el usuario lanza la aplicación mediante el archivo principal *main.py*, situado en la raíz del proyecto. Esta interfaz gráfica, desarrollada con la librería Tkinter, actúa como punto de entrada centralizado y permite al usuario seleccionar si desea trabajar con el agente DRL (entrenamiento y evaluación) o con el simulador de red PON.

Ejecución del agente

Si el usuario selecciona la opción de ejecutar el agente, se habilitan tres posibilidades:

1. Realizar únicamente un entrenamiento.
2. Ejecutar únicamente una evaluación sobre un modelo ya entrenado.
3. Realizar ambas acciones de forma secuencial: entrenar y luego evaluar.

En cualquiera de los tres casos, el procedimiento comienza con la creación de un objeto de la clase LocalAgent, que representa al agente en modo local. Este objeto se instancia con todos los parámetros necesarios para simular correctamente la red: número de ONTs, duración del ciclo (step_duration), capacidad de la OLT (OLT_capacity), velocidad contratada por cada ONT (v_contratada), número máximo de ciclos de simulación, tipo de entorno, etc.

- Entrenamiento del agente
  Cuando el usuario selecciona la opción de entrenamiento, el sistema invoca internamente al módulo model_manager.py, encargado de gestionar los modelos. En concreto, se llama a la función create_model, que recibe el entorno y el algoritmo especificado (por defecto, PPO de la librería Stable-Baselines3), y devuelve un modelo inicial listo para entrenar.

  Una vez creado el modelo, el método train_model() de LocalAgent ejecuta el proceso de entrenamiento durante el número de timesteps indicado por el usuario. Durante este entrenamiento, el modelo interactúa con el entorno de Gymnasium (RedesOpticasEnv), recibiendo observaciones normalizadas, tomando decisiones de asignación de ancho de banda y recibiendo recompensas diseñadas para penalizar colas largas o ineficiencia.

Al finalizar el entrenamiento, el modelo se guarda automáticamente en disco con el nombre de fichero especificado por el usuario. Este archivo se almacena en el subdirectorio de modelos (/models) en formato .zip, incluyendo todos los pesos y parámetros necesarios para su posterior reutilización.

- Evaluación de un modelo Si el usuario selecciona la opción de evaluación, el flujo de ejecución es similar, pero con algunas diferencias clave. El sistema crea igualmente un objeto de la clase LocalAgent con los parámetros de simulación indicados, pero en este caso se llama al método load_model() del módulo model_manager.py, que carga un modelo previamente entrenado a partir del nombre de fichero proporcionado.

  Una vez cargado el modelo, se ejecuta el método exec_simulation(), que realiza una evaluación completa durante el número de ciclos indicado. Durante esta simulación, el entorno genera tráfico para cada ONT según una distribución determinada, el modelo toma decisiones de asignación de ancho de banda en función del estado de las colas, y se actualiza internamente el estado de la red. Este proceso se repite hasta completar los ciclos definidos.

  Finalizada la evaluación, los datos recogidos (colas, tráfico, asignación, etc.) se almacenan en un fichero CSV para su posterior análisis, y se genera una representación gráfica de los resultados mediante el módulo plotter.py. Este módulo permite visualizar métricas clave como el uso total de ancho de banda, la evolución de las colas por ONT, el rendimiento del modelo o las colisiones de tráfico.

- Entrenamiento + Evaluación Si el usuario selecciona ambas opciones (entrenamiento y evaluación consecutiva), el sistema sigue secuencialmente ambos flujos. Primero entrena un nuevo modelo desde cero, lo guarda, y a continuación lo carga para evaluar su rendimiento sobre un entorno equivalente. Este enfoque permite analizar rápidamente la efectividad de los modelos recién entrenados sin necesidad de ejecutar procesos por separado.

Ejecución del simulador

Cuando el usuario selecciona la opción de ejecutar el simulador, el comportamiento del sistema cambia. En este caso, no se trabaja con el entorno simplificado de Gymnasium, sino con el simulador de red PON basado en eventos discretos, implementado con la librería SimPy. Todos los parámetros de la red (tiempo de simulación, tipo de tráfico, número de ONTs,

etc.) se definen en un fichero de configuración, que es leído automáticamente al lanzar la simulación.

Una de las decisiones clave en este modo es la selección de la estrategia de asignación de ancho de banda que utilizará la OLT. El usuario puede elegir entre tres opciones:

1. OLT básica (asignación tradicional)
   Se utiliza una clase OLT que sigue una lógica determinista simple: asignar ancho de banda únicamente si existen elementos en la cola, sin ningún tipo de prioridad ni optimización. Esta opción sirve como referencia base para comparar con modelos más avanzados.

2. OLT con red neuronal (OLT_DNN)
   En este caso se utiliza un modelo preentrenado basado en redes neuronales profundas (DNN). Este modelo toma decisiones a partir de observaciones del estado de la red, pero sin interacción directa con el entorno ni proceso de aprendizaje en tiempo real.

3. OLT con aprendizaje por refuerzo profundo (OLT_DRL)
   Esta es la opción más avanzada, en la que la clase OLT_DRL actúa como interfaz con un agente externo entrenado previamente. Para ello, se crea un objeto de la clase InterfazAgente, al que se le pasa toda la información de la red (estado, colas, configuración, etc.) y el nombre del modelo a utilizar.

La InterfazAgente transforma los datos del simulador (colas, tráfico, etc.) al formato que requiere el agente, y crea internamente un objeto de la clase SimAgent. Este objeto se encarga de cargar el modelo entrenado y realizar predicciones sobre la asignación de ancho de banda.

El flujo en este caso es bidireccional:

1. El simulador proporciona observaciones del estado.

2. La interfaz transforma esos datos y se los proporciona al modelo.

3. El modelo devuelve una acción (asignación).

4. La acción se transforma nuevamente y se aplica en el simulador.

Este proceso se repite cada vez que una ONT reporta a la OLT, sin interacción con Gymnasium ni necesidad de entrenamiento adicional, ya que el modelo está congelado en su estado entrenado.

En resumen, el diseño procedimental de permite al usuario ejecutar diferentes modos del sistema —entrenamiento, evaluación, simulación— de forma estructurada y flexible. La lógica interna de cada proceso se ha diseñado para ser coherente, reutilizable y fácilmente configurable, facilitando tanto la validación de modelos como la exploración de distintas estrategias de gestión de ancho de banda en redes ópticas pasivas

Apéndice D

Documentación técnica de programación

D.1. Introducción

Este apéndice recoge la documentación técnica necesaria para comprender, instalar, ejecutar y mantener el sistema desarrollado en el marco de este proyecto. Está orientado a programadores o futuros desarrolladores que deseen trabajar sobre el código, extender su funcionalidad o adaptarlo a nuevos entornos.

Se describe en detalle la estructura de archivos y directorios del proyecto, se proporciona un manual del programador con información sobre los principales módulos y clases implementadas, y se incluyen las instrucciones necesarias para compilar, instalar y ejecutar el sistema. Finalmente, se documentan las pruebas realizadas, así como los resultados obtenidos durante el proceso de validación.

D.2. Estructura de directorios

El sistema desarrollado se organiza en una estructura modular de directorios que permite separar claramente los componentes funcionales del proyecto: la interfaz principal, el agente de aprendizaje por refuerzo profundo (DRL) y el simulador de red óptica pasiva (PON). Esta organización facilita tanto el mantenimiento como la extensibilidad del sistema, y permite ejecutar de forma independiente las distintas partes del proyecto.

A continuación, se detalla la estructura principal del proyecto:

Raíz del proyecto

main.py

Archivo principal del sistema, que contiene la interfaz gráfica desarrollada con Tkinter. Actúa como punto de entrada para lanzar el agente, el simulador o ambas funcionalidades, según la opción seleccionada por el usuario.

agente/

Contiene todos los elementos relacionados con la implementación, entrenamiento y evaluación del agente DRL.

- main_agent.py
  Punto de entrada específico para ejecutar el agente de forma independiente.
- classes/
  Incluye las definiciones de las clases que implementan el comportamiento del agente:
  - BaseAgent.py
    Clase base del agente, con métodos comunes para entrenamiento, evaluación y gestión de modelos.
  - LocalAgent.py
    Agente que interactúa con un entorno simulado local (Gymnasium).
  - SimAgent.py
    Agente diseñado para integrarse con el simulador de red externo.
- custom_env/
  Contiene los ficheros relacionados con el entorno personalizado de Gymnasium:
  - RedesOpticasEnv.py
    Entorno que simula el comportamiento de una red PON, incluyendo estados, recompensas y pasos.
  - traficoparetopython.py
    Script para generar tráfico sintético siguiendo una distribución de Pareto, representativa del comportamiento real de los usuarios.

- modules/
  Módulos auxiliares reutilizables en diferentes fases del flujo del agente:
  - model_manager.py
    Se encarga de crear, guardar y cargar modelos entrenados mediante la librería Stable-Baselines3.
  - plotter.py
    Genera gráficas a partir de los resultados de las simulaciones, facilitando el análisis del rendimiento del agente.

simulador/

Este directorio contiene todos los componentes necesarios para ejecutar la simulación realista de una red PON.

- main_sim.py
  Punto de entrada del simulador completo.
- ejecutar_simulacion.py
  Función principal que lanza la simulación con los parámetros indicados.
- calcular_estadisticas.py
  Módulo que recoge y procesa estadísticas generadas durante la simulación.
- data/
  Contiene ficheros de datos utilizados por el simulador, como colas predefinidas o muestras de tráfico.
- packages/
  Incluye la lógica de simulación separada en dos carpetas:
  - classes/
    Núcleo del simulador, compuesto por todas las clases que modelan los componentes de red y su comportamiento:

    TramaEthernet, ParetoGenerator, ONT, OLT, OLTDRL, OLTDNN, MensajeReport, MensajeGate, InterfazAgente, GeneraTrafico, EstadisticasWelford, Enlace. Estas clases implementan la lógica de simulación de eventos discretos usando SimPy, y modelan aspectos como la comunicación entre ONTs y OLT, la generación de tráfico, y el uso de diferentes estrategias de asignación de ancho de banda.

- configuration/
  Contiene archivos de configuración clave:
  - parametros.py
    Define los valores de la red como el número de ONTs, el tiempo de ciclo o la capacidad de transmisión de la OLT.
  - configuracion.py
    Contiene los parámetros mas orientados a la simulación, como el tiempo de simulación, la carga de la red, activar o desactivar el verbose, etc

Esta estructura modular permite ejecutar, depurar y mejorar el sistema de forma flexible. Cada componente (agente y simulador) puede evolucionar de manera independiente, y la separación entre lógica, configuración y presentación favorece la mantenibilidad del código y su futura reutilización en otros escenarios o proyectos.

D.3. Manual del programador

Este manual está dirigido a desarrolladores que deseen comprender el funcionamiento interno del proyecto, modificar sus componentes o extender sus funcionalidades. El proyecto ha sido diseñado siguiendo principios de modularidad, reutilización y orientación a objetos, lo que facilita su mantenimiento y evolución. A continuación, se describen los principales paquetes, clases y métodos, junto con recomendaciones para futuras modificaciones.

Estructura general

El sistema se divide en dos componentes principales: el agente DRL y el simulador de red PON, ubicados en los directorios agente/ y simulador/ respectivamente. Cada uno cuenta con su propio archivo de entrada (main_agente.py y main_simulador.py) y su lógica encapsulada en submódulos bien definidos.

Agente DRL

El componente agente/ está diseñado para entrenar, evaluar y aplicar políticas de decisión en un entorno simulado mediante la librería Gymnasium. Su estructura incluye:

- classes/BaseAgent.py
  Clase base común a todos los agentes. Define la estructura general de un agente: creación del entorno, funciones para entrenar (train_model), guardar modelos (save_model) y cargarlos (load_model). Puede extenderse para implementar agentes con diferentes modos de ejecución. Cuenta también con todas las variables correspondientes a la red con la que va a tratar el modelo, como el número de ONTs, la velocidad de transmisión, etc.

- classes/LocalAgent.py
  Hereda de BaseAgent. Utiliza un entorno local (RedesOpticasEnv) compatible con Gymnasium para entrenar y evaluar modelos. Este agente se usa durante el desarrollo y en pruebas independientes del simulador. Cuenta con el método para realizar una simulación (exec_simulation) y para guardar los resultados de esta simulación (save_results).

- classes/SimAgent.py
  También hereda de BaseAgent. Está diseñado para integrarse con el simulador externo a través de la clase InterfazAgente. Implementa el método get_prediction() para devolver acciones en tiempo real que solicita al modelo en función de las observaciones del simulador que aporta la interfaz.

- custom_env/RedesOpticasEnv.py
  Define un entorno personalizado compatible con Gymnasium. Simula una red PON simplificada, gestionando las colas, la generación de tráfico y el cálculo de recompensas. Incluye los métodos estándar step, reset, get_obs, calculate_reward, etc. Al final del fichero, después de la clase, se registra el entorno asignándole un identificador y especificando el endpoint para el acceso a este. Con esto podemos registrar diferentes entornos de gym y cambiar fácilmente entre ellos utilizando esos identificadores.

- custom_env/traficoparetopython.py
  Script auxiliar para generar tráfico sintético siguiendo una distribución de Pareto, representando el comportamiento realista de los usuarios de red. Fue desarrollado por Clara Ruiz y cuenta con una simulación más realista al implementar una clase que representa cada ONT.

- modules/model_manager.py
  Contiene funciones para crear, guardar y cargar modelos usando Stable-Baselines3. Utiliza configuraciones por defecto pero puede modificarse

para experimentar con distintos algoritmos, políticas o hiperparámetros. Se encarga también de calcular la ruta donde deberán de guardarse o cargarse estos modelos.

- modules/plotter.py
  Se encarga de generar gráficas a partir de los resultados una evaluación. Permite visualizar el tráfico de entrada y de salida de cada ONT, así como el estado de las colas.

Simulador de red

El componente simulador/ implementa una simulación de red PON basada en eventos discretos mediante la librería SimPy. Sus principales componentes son:

- main_simulador.py
  Punto de entrada del simulador. Llama a el módulos ejecutar_simulacion.py pasándole como argumento el tipo de simulación a ejecutar (normal, con DRL o con DNN)

- ejecutar_simulacion.py
  Función que inicia el proceso de simulación, leyendo la configuración definida en los ficheros del submódulo configuracion y creando todos los objetos de las clases necesarios. Aquí es donde, en función del modo de ejecución seleccionado, creamos objetos de las clases OLT, OLT_DRL o OLT_DNN.

- calcular_estadisticas.py
  Procesa los resultados de la simulación, generando métricas sobre tráfico, uso de ancho de banda, colas, etc.

- packages/classes/
  Contiene las clases que modelan el comportamiento de la red:

  - OLT, OLT_DRL, OLT_DNN
    Representan el Optical Line Terminal y sus distintas estrategias de asignación de ancho de banda.
  - ONT, Enlace, TramaEthernet
    Modelan los dispositivos físicos y el tráfico.
  - InterfazAgente
    Puente entre el simulador y el agente. Convierte datos, gestiona la comunicación y solicita predicciones al modelo.

  - GeneraTrafico, ParetoGenerator
    Generación dinámica de tráfico por ONT.
  - MensajeReport, MensajeGate
    Modelan los mensajes de control en la red.
  - EstadisticasWelford
    Implementa el algoritmo de Welford para el cálculo incremental de la media y la varianza.

- packages/configuration/

  - parameters.py
    Contiene los parámetros de la red, como los tiempos de transmisión, el tamaño de las cabeceras, el tamaño de los buffers de las ONTs, etc.
  - configuration.py
    Contiene los parámetros de la simulación, como el tiempo de simulación, la carga de la red, los flags para activar las múltiples colas, el verbose, etc.

Recomendaciones para desarrolladores

Para añadir un nuevo tipo de agente, basta con crear una clase que herede de BaseAgent e implementar los métodos necesarios. Además, el model manager cuenta también con algo de soporte para diferentes algoritmos de DRL incluidos en Stable-Baselines3, aunque es necesario especificar los nuevos hiperparámetros si se desea cambiar de algoritmo.

Para añadir una nueva política de decisión en la OLT, crear una clase que herede de OLT y sobreescribir la asginación de ancho de banda que se realiza en el método procesa_report(). Aunque el resto del código es prácticamente igual y es un poco ineficiente tener varios ficheros con un 90 % de líneas iguales, a menos que se implemente una clase abstracta de la cual hereden todas las clases OLT con el código compartido, por el momento la única opción es copiar toda la clase.

Para probar un nuevo entorno Gym, se recomienda duplicar y adaptar RedesOpticasEnv.py. para asegurarnos de tener una buena compatibilidad.

Los módulos model_manager y plotter están desacoplados del núcleo del agente y pueden reutilizarse en otros proyectos.

Despliegue

Para distribuir el sistema como una aplicación ejecutable en entornos Windows sin necesidad de instalar Python ni sus dependencias, se ha utilizado la herramienta PyInstaller. Esta permite empaquetar todos los ficheros necesarios en un único ejecutable (.exe), que lanza directamente la interfaz gráfica del sistema (main.py).

El proceso de empaquetado se basa en un archivo de configuración llamado main.spec, donde detallamos qué elementos del sistema irán empaquetados en el ejecutable y qué librerías necesita compilar también. Además, en este fichero de especificaciones podemos definir también los parámetros de la compilación, como la opción *–onedir* que, al contrario que *–onefile* donde se crea un único .exe, en esta opción se crea un directorio, es clave en nuestro sistema ya que este necesita escribir y leer ficheros en tiempo de ejecución.
Con el fichero ya configurado, el empaquetado no necesita más que la ejecución, dentro del directorio raíz del proyecto, del siguiente comando: `pyinstaller main.spec`

Esto crea una carpeta dist/ con el ejecutable autónomo que puede distribuirse directamente.

Es importante tener en cuenta también que el simulador de red utiliza archivos Python en el directorio packages/configuration/(parameters.py y configuration.py) para definir los valores de simulación: tiempos, tráfico, tipo de OLT, número de ONTs, etc. Al tratarse de ficheros .py que se incluyen dentro del ejecutable mediante PyInstaller, cualquier cambio posterior realizado manualmente en esos archivos no tendrá efecto sobre el .exe ya generado.

Por tanto, si se desea modificar la configuración predeterminada del simulador, es necesario editar los ficheros .py correspondientes dentro del proyecto y luego recompilar el ejecutable con PyInstaller. De lo contrario, el ejecutable seguirá funcionando con los valores anteriores.

D.4. Compilación, instalación y ejecución del proyecto

Este apartado describe los pasos necesarios para instalar y ejecutar el sistema SimDeepPON-RL en un entorno local. Se explican los requisitos,

la estructura de ejecución, y cómo generar un ejecutable autónomo para Windows mediante PyInstaller.

Requisitos previos

Para ejecutar el sistema desde el código fuente es necesario disponer de:

- Python 3.10 o superior
- Librerías externas como: stable-baselines3, gymnasium, simpy, tkinter, matplotlib, numpy, pytorch, entre otras.

Estas dependencias deben instalarse manualmente, o bien mediante una herramienta de gestión de entornos como venv o conda.

Ejecución del sistema

El sistema puede ejecutarse de dos formas: desde la interfaz gráfica o desde línea de comandos.

Modo gráfico (recomendado)

Ejecutando el archivo principal main.py desde la raíz del proyecto python main.py. Esto lanza la interfaz gráfica del sistema, permitiendo al usuario seleccionar entre ejecutar el agente (entrenamiento, evaluación o ambos) o el simulador con distintos modos de decisión.

Modo línea de comandos (avanzado)

También es posible ejecutar directamente los componentes internos:

- python agente/main_agente.py
- python simulador/main_simulador.py

No obstante, es importante tener en cuenta que estos scripts fueron diseñados para ser lanzados desde la raíz del proyecto, donde reside main.py. Por ello, los imports relativos al sistema funcionan correctamente solo cuando la raíz es el directorio actual. Si se intenta ejecutar directamente uno de estos archivos desde su propia carpeta, pueden aparecer errores de importación. En ese caso, sería necesario ajustar manualmente las rutas de importación para cada módulo, lo cual no se recomienda.

Compilación del ejecutable

Para facilitar la distribución del sistema, se ha configurado un proceso de empaquetado mediante PyInstaller, que permite generar un ejecutable único (.exe) para entornos Windows. Este ejecutable incluye internamente todo el sistema, sin necesidad de instalación adicional de Python ni de librerías externas.
Para simplificar este empaquetado se ha configurado un fichero main.spec donde se especifican los datos adicionales que añadir al ejecutable, las librerías requeridas y las opciones de PyInstaller necesarias. El proceso para esta compilación sería:

1. Instalar PyInstaller
   Ejecutando el comando `pip install pyinstaller`
2. Editar el fichero main.spec
   Modificar el main.spec si es necesario para incluir algún tipo de fichero de datos, módulo python o librerías.
3. Compilar el ejecutable final
   Una vez configurado el fichero de especificaciones, solo falta ejecutar pyinstaller mediante el siguiente comando y siempre desde la raiz del sistema donde se enuentra main.py: `pyinstaller main.spec`

Tras esto, el ejecutable se generará en el directorio dist/ con el nombre main.exe. Puede ejecutarse directamente y abrirá la interfaz gráfica completa del sistema.

Nota importante: el simulador utiliza archivos de configuración ubicados en simulador/paquetes/configuracion/. Estos archivos .py se integran dentro del ejecutable en el momento de la compilación. Por tanto, si se desea modificar los parámetros del simulador (como tipo de OLT, número de ONTs, o duración de la simulación), es necesario recompilar el ejecutable con los nuevos valores previamente modificados en el código fuente.

D.5. Pruebas del sistema

Para validar el correcto funcionamiento del sistema, se han realizado pruebas de ejecución tanto desde el código fuente mediante línea de comandos como desde el ejecutable generado con PyInstaller. El objetivo principal ha sido comprobar la estabilidad de la interfaz gráfica, la carga de módulos

Identificador	CP01
Descripcion	Confirmar la correcta ejecución de la aplicación en línea de comandos.
Precondiciones	Se tiene el sistema descargado e instalado python y las dependencias.
Postcondiciones	El sistema no ha mostrado ninguna excepción.
Pasos	Abrir una consola de powershell o cmd dentro del directorio donde se tenga descomprimido el proyecto. Tras esto ejecutar el comando *python main.py*
Salida esperada	Se abre la ventana de la interfaz gráfica con todas las opciones esperadas y sin ningún error

Tabla D.1: CP01 Ejecución línea comandos

y la interacción entre componentes en ambos entornos. A continuación, se detallan los casos de prueba realizados.

Identificador	CP02
Descripcion	Confirmar la correcta ejecución de un entrenamiento del modelo en línea de comandos.
Precondiciones	Se tiene el sistema descargado e instalado python y las dependencias.
Postcondiciones	El sistema no ha mostrado ninguna excepción y se ha creado correctamente un nuevo fichero pruebas.zip en la carpeta de modelos, con el modelo entrenado.
Pasos	Abrir una consola de powershell o cmd dentro del directorio donde se tenga descomprimido el proyecto. Tras esto ejecutar el comando *python main.py*. Seleccionar la opción de *Ejecutar Agente* y dentro de esta, la opción de *Entrenamiento*. Dejar los datos por defecto y cambiar el nombre del modelo a "pruebas". Por último hacer click en el botón para comenzar el entrenamiento
Salida esperada	El sistema ejecuta un entrenamiento y en la consola de cmd vemos los datos del entrenamiento, además en el directorio *models* se ha creado un fichero pruebas.zip con el modelo entrenado

Tabla D.2: CP02 Entrenamiento línea comandos

Identificador	CP03
Descripcion	Confirmar la correcta ejecución de una evaluación del modelo en línea de comandos.
Precondiciones	Se tiene el sistema descargado e instalado python y las dependencias. Existe un modelo entrenado en el directorio *models*
Postcondiciones	El sistema ejecuta la evaluación correctamente y además de almacenar los resultados en csv en el directorio *logs*, nos da la opción de visualizar las gráficas
Pasos	Abrir una consola de powershell o cmd dentro del directorio donde se tenga descomprimido el proyecto. Tras esto ejecutar el comando *python main.py*. Seleccionar la opción de *Ejecutar Agente* y dentro de esta, la opción de *Evaluación*. Dejar los datos por defecto y cambiar el nombre del modelo al nombre que tenga el fichero almacenado en *models*. Por último hacer click en el botón para comenzar la evaluación
Salida esperada	El sistema ejecuta una evaluación y nos crea un fichero csv dentro de *logs* con los resultados de la evaluación. Además, al terminar nos lanza una ventana para seleccionar una ONT y unos datos que visualizar para mostrarnos la gráfica correspondiente.

Tabla D.3: CP03 Evaluación línea comandos

Identificador	CP04
Descripcion	Confirmar la correcta ejecución del simulador en modo normal en línea de comandos.
Precondiciones	Se tiene el sistema descargado e instalado python y las dependencias.
Postcondiciones	El sistema ejecuta correctamente una simulación completa y muestra en la ventana de cmd el progreso y los resultados de la simulación
Pasos	Abrir una consola de powershell o cmd dentro del directorio donde se tenga descomprimido el proyecto. Tras esto ejecutar el comando *python main.py*. Seleccionar la opción de *Ejecutar Simulador* y dentro de esta, la opción de *Simulador Normal*. Por último hacer click en el botón para comenzar la simulación
Salida esperada	El sistema ejecuta una simulación y nos imprime el progreso en la ventana de cmd. Al terminar nos muestra también en la ventana de cmd las estadísticas de la simulación.

Tabla D.4: CP04 Simulación línea comandos

Identificador	CP05
Descripcion	Confirmar la correcta ejecución del simulador en modo DRL en línea de comandos.
Precondiciones	Se tiene el sistema descargado e instalado python y las dependencias. Además exsite también un modelo entrenado en el directorio *models*
Postcondiciones	El sistema ejecuta correctamente una simulación completa y muestra en la ventana de cmd el progreso y los resultados de la simulación
Pasos	Abrir una consola de powershell o cmd dentro del directorio donde se tenga descomprimido el proyecto. Tras esto ejecutar el comando *python main.py*. Seleccionar la opción de *Ejecutar Simulador* y dentro de esta, la opción de *Simulador DRL*. Introducir el nombre del modelo entrenado existente en el directorio *models*. Por último hacer click en el botón para comenzar la simulación
Salida esperada	El sistema ejecuta una simulación y nos imprime el progreso en la ventana de cmd. Al terminar nos muestra también en la ventana de cmd las estadísticas de la simulación.

Tabla D.5: CP05 Simulación DRL línea comandos

Identificador	CP06
Descripcion	Confirmar el correcto despliegue de la aplicación.
Precondiciones	Se tiene el sistema descargado e instalado python y las dependencias, así como PyInstaller. Se necesita también el fichero main.spec correctamente configurado
Postcondiciones	PyInstaller ejecuta correctamente el empaquetado y crea el directorio *dist* con el ejecutable en su interior
Pasos	Abrir una consola de powershell o cmd dentro del directorio donde se tenga descomprimido el proyecto. Tras esto ejecutar el comando *pyinstaller main.spec*.
Salida esperada	Se ejecuta pyinstaller sin ningún error y como resultado se crea en el directorio la carpeta *dist* conteniendo el ejecutable y una carpeta con todo el proyecto empaquetado

Tabla D.6: CP06 Despliegue aplicación

Identificador	CP07
Descripcion	Confirmar la correcta ejecución de la aplicación desplegada.
Precondiciones	Se tiene el sistema desplegado, con el ejectuable y la carpeta con el sistema empaquetado
Postcondiciones	El sistema no ha mostrado ninguna excepción.
Pasos	Ejecutar la aplicación haciendo doble click en el fichero *SimDeepPON-RL.exe*
Salida esperada	Se abre la ventana de la interfaz gráfica con todas las opciones esperadas y sin ningún error

Tabla D.7: CP07 Ejecución despliegue

Identificador	CP08
Descripcion	Confirmar la correcta ejecución de un entrenamiento del modelo la aplicación desplegada.
Precondiciones	Se tiene el sistema desplegado, con el ejectuable y la carpeta con el sistema empaquetado.
Postcondiciones	El sistema no ha mostrado ninguna excepción y se ha creado correctamente un nuevo fichero pruebas.zip en la carpeta de modelos, con el modelo entrenado.
Pasos	Ejecutar la aplicación haciendo doble click en el fichero *SimDeepPON-RL.exe*. Seleccionar la opción de *Ejecutar Agente* y dentro de esta, la opción de *Entrenamiento*. Dejar los datos por defecto y cambiar el nombre del modelo a "pruebas". Por último hacer click en el botón para comenzar el entrenamiento
Salida esperada	El sistema ejecuta un entrenamiento y en la consola de cmd vemos los datos del entrenamiento, además en el directorio *models* se ha creado un fichero pruebas.zip con el modelo entrenado

Tabla D.8: CP08 Entrenamiento despliegue

Identificador	CP09
Descripcion	Confirmar la correcta ejecución de una evaluación del modelo en la aplicación desplegada.
Precondiciones	Se tiene el sistema desplegado, con el ejectuable y la carpeta con el sistema empaquetado. Existe un modelo entrenado en el directorio *models* dentro del sistema empaquetado.
Postcondiciones	El sistema ejecuta la evaluación correctamente y además de almacenar los resultados en csv en el directorio *logs*, nos da la opción de visualizar las gráficas
Pasos	Ejecutar la aplicación haciendo doble click en el fichero *SimDeepPON-RL.exe*. Seleccionar la opción de *Ejecutar Agente* y dentro de esta, la opción de *Evaluación*. Dejar los datos por defecto y cambiar el nombre del modelo al nombre que tenga el fichero almacenado en *models*. Por último hacer click en el botón para comenzar la evaluación
Salida esperada	El sistema ejecuta una evaluación y nos crea un fichero csv dentro de *logs* con los resultados de la evaluación. Además, al terminar nos lanza una ventana para seleccionar una ONT y unos datos que visualizar para mostrarnos la gráfica correspondiente.

Tabla D.9: CP09 Evaluación despliegue

Identificador	CP10
Descripcion	Confirmar la correcta ejecución del simulador en modo normal en la aplicación desplegada.
Precondiciones	Se tiene el sistema desplegado, con el ejectuable y la carpeta con el sistema empaquetado.
Postcondiciones	El sistema ejecuta correctamente una simulación completa y muestra en la ventana de cmd el progreso y los resultados de la simulación
Pasos	Ejecutar la aplicación haciendo doble click en el fichero *SimDeepPON-RL.exe*. Seleccionar la opción de *Ejecutar Simulador* y dentro de esta, la opción de *Simulador Normal*. Por último hacer click en el botón para comenzar la simulación
Salida esperada	El sistema ejecuta una simulación y nos imprime el progreso en la ventana de cmd. Al terminar nos muestra también en la ventana de cmd las estadísticas de la simulación.

Tabla D.10: CP10 Simulación despliegue

Identificador	CP11
Descripcion	Confirmar la correcta ejecución del simulador en modo DRL en la aplicación desplegada.
Precondiciones	Se tiene el sistema desplegado, con el ejectuable y la carpeta con el sistema empaquetado. Existe un modelo entrenado en el directorio *models* dentro del sistema empaquetado.
Postcondiciones	El sistema ejecuta correctamente una simulación completa y muestra en la ventana de cmd el progreso y los resultados de la simulación
Pasos	Ejecutar la aplicación haciendo doble click en el fichero *SimDeepPON-RL.exe*. Seleccionar la opción de *Ejecutar Simulador* y dentro de esta, la opción de *Simulador DRL*. Introducir el nombre del modelo entrenado existente en el directorio *models* dentro del sistema empaquetado. Por último hacer click en el botón para comenzar la simulación
Salida esperada	El sistema ejecuta una simulación y nos imprime el progreso en la ventana de cmd. Al terminar nos muestra también en la ventana de cmd las estadísticas de la simulación.

Tabla D.11: CP11 Simulación DRL despliegue

Apéndice E

Documentación de usuario

E.1. Introducción

En este apéndice veremos, de cara a un usuario, todos los pasos que hay que seguir para poder utilizar el sistema desarrollado durante este proyecto.

Veremos primero los requisitos que requiere el sistema para poder ser ejecutado. Relacionado con los requisitos procederé a mostrar los pasos necesarios para instalar el software necesario para el correcto funcionamiento del sistema, y por último tendremos un manual de usuario donde explicaré para un usuario básico, como poder configurar y ejecutar el sistema, tanto en su versión independiente, con el simulador y el agente por separado, como en su versión enlazada.

E.2. Requisitos de usuarios

Para utilizar el sistema SimDeepPON-RL no se requieren conocimientos avanzados de programación ni del funcionamiento interno del agente o del simulador, ya que el uso se realiza a través de una interfaz gráfica sencilla e intuitiva.

Sin embargo, se recomienda que el usuario cumpla los siguientes requisitos mínimos:

- Conocimientos básicos sobre el funcionamiento de redes de telecomunicaciones o sistemas distribuidos (para interpretar los resultados de las simulaciones).

- Familiaridad con el uso de aplicaciones de escritorio y exploración de carpetas en Windows.
- En el caso de usar la versión en código fuente, conocimientos básicos de terminal (cmd, PowerShell) y entorno Python (opcional pero útil).

E.3. Instalación

Existen dos formas de utilizar el sistema:

Opción 1: Uso del archivo ejecutable (*SimDeepPON-RL.exe*)

Esta es la opción más sencilla y recomendada para usuarios no técnicos. Pasos:

1. Descargar la carpeta completa llamada *dist/*. Esta carpeta debe contener una carpeta con el nombre del proyecto, y dentro:
   - El archivo *SimDeepPON-RL.exe*
   - Una subcarpeta llamada *_internal*, que contiene Varias subcarpetas y archivos .dll, .pyd, y .pyc generados por PyInstaller y las carpetas *agente/* y *simulador/* con toda la lógica del sistema
2. Colocar la carpeta en cualquier ubicación del sistema. No es necesaria instalación ni configuración adicional.
3. Para ejecutar el sistema, abrir la carpeta y hacer doble clic sobre el archivo *SimDeepPON-RL.exe*

Se abrirá la interfaz gráfica del sistema, desde la cual el usuario podrá:

- Entrenar un modelo
- Evaluar un modelo existente
- Ejecutar el simulador con distintos modos de toma de decisiones
- Visualizar resultados en forma de gráficas

Nota importante: no eliminar ni mover archivos de la carpeta *dist/*, ya que el ejecutable depende de varios módulos externos ubicados allí. Si se elimina alguno, el programa podría dejar de funcionar.

Opción 2: Ejecución desde código fuente

Esta opción está orientada a usuarios con conocimientos técnicos que deseen trabajar directamente con el código, modificarlo o realizar pruebas avanzadas. Pasos:

1. Descargar el proyecto completo o clonar el repositorio [9] mediante el comando: `git clone 'URL Repositorio'`
2. Acceder a la carpeta del proyecto con la consola de comandos de windows o powershell
3. Instalar las dependencias necesarias con pip install (según los imports del código).
4. Ejecutar la interfaz principal con python main.py

Atención: La ejecución directa desde *agente/main_agente.py* o *simulador/main_simulador.py* puede requerir modificar los imports relativos, ya que estos archivos están diseñados para ejecutarse desde la raíz del proyecto.

E.4. Manual del usuario

La aplicación incluye una interfaz gráfica desarrollada con la librería Tkinter, que permite al usuario interactuar con el agente de aprendizaje y el simulador sin necesidad de utilizar la línea de comandos. Esta interfaz facilita la configuración de parámetros, la ejecución de pruebas y la visualización de resultados.

Pantalla principal

Al ejecutar el sistema, bien sea mediante comandos con *python main.py* o mediante el ejecutable haciendo doble clic en */dist/SimDeepPON-RL/SimDeepPON-RL.exe*, se abre una ventana principal con los siguientes elementos:

- Un cuadro de texto tipo log, donde se van mostrando mensajes informativos sobre el estado de la aplicación.
- Dos botones de selección inicial: Ejecutar agente y Ejecutar simulador

Además, se nos abrirá también una ventana de consola (cmd) adicional en paralelo, donde se muestran las salidas estándar del sistema, como mensajes de progreso, información de entrenamiento o métricas de evaluación.E.1

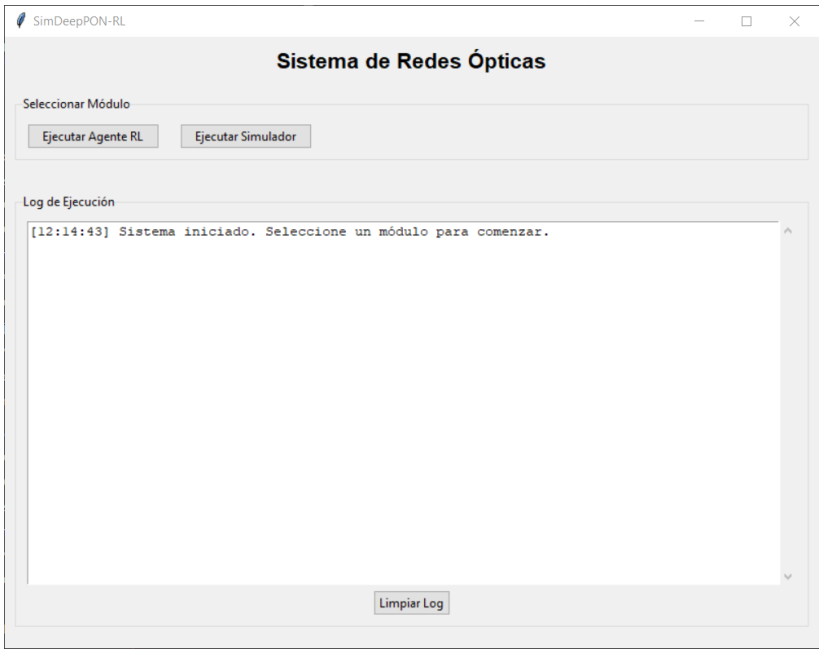

Figura E.1: Pantalla principal de la interfaz gráfica

Ejecución del agente

Al pulsar el botón *Ejecutar agente*, el sistema permite elegir entre tres modos de funcionamiento:

- Entrenamiento
- Evaluación
- Ambas (Entrenamiento + Evaluación)

Figura E.2: Opciones de ejecución del agente

Una vez seleccionada una de estas opciones, se habilita un cuadro con los campos necesarios para introducir los parámetros correspondientes. La interfaz adapta dinámicamente los campos visibles en función del modo elegido. Por ejemplo, si se selecciona entrenamiento, se mostrará un campo para el número de pasos de entrenamiento y otro para el número de entornos paralelos.E.3

Durante la ejecución, los mensajes de progreso se muestran en la consola y los mensajes del estaod del sistema en la ventana de log.

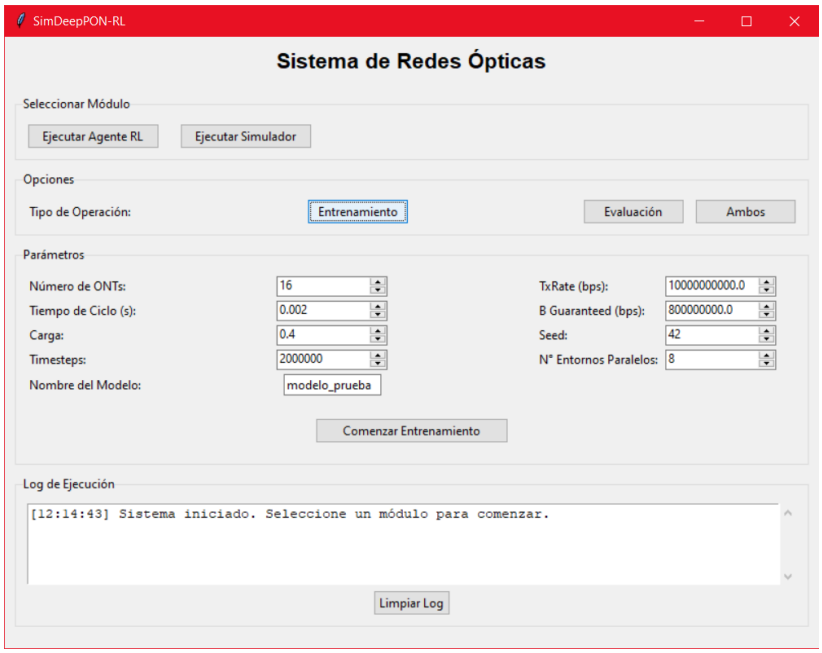

Figura E.3: Parametros para un entrenamiento del agente

Al finalizar la ejecución, se muestra un mensaje de confirmación en la ventana de logs y, si se ha ejecutado una evaluación, aparece una ventana emergente preguntando al usuario si desea visualizar las gráficas generadas.

En caso afirmativo, se abre una segunda ventana E.4 con los siguientes elementos:

- Un menú desplegable para seleccionar la ONT a visualizar.
- Dos botones de radio para elegir entre visualizar la gráfica de tráfico de entrada/salida o la gráfica de evolución de colas.
- Un botón para mostrar la gráfica seleccionada.
- Otro botón para mostrar todas las gráficas.

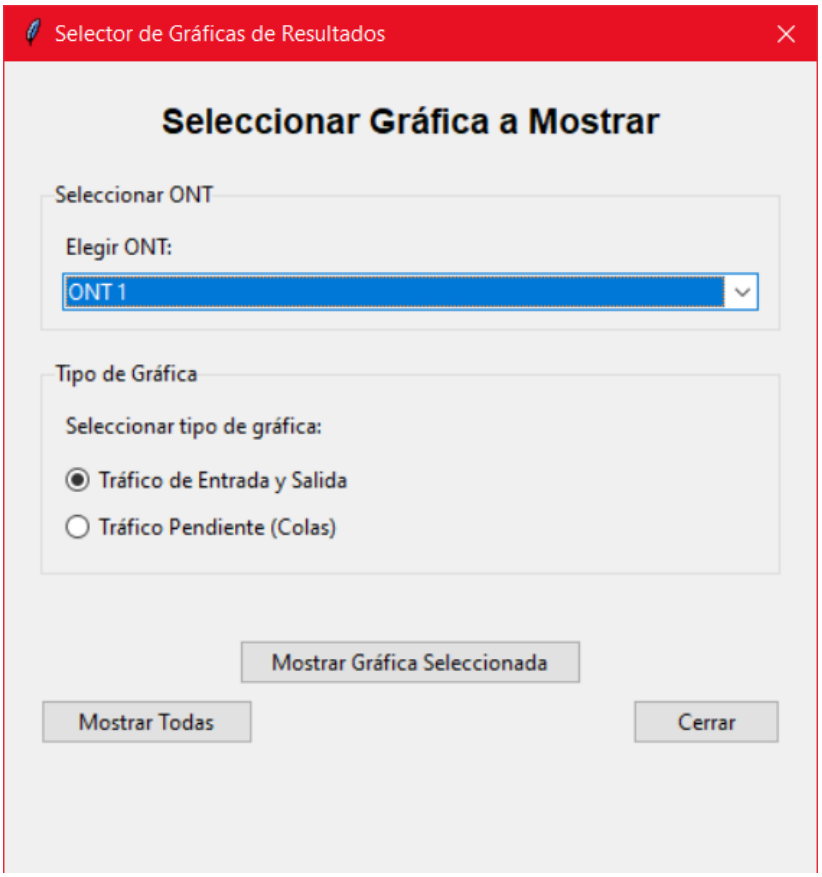

Figura E.4: Ventana para la selección de gráficas

Ejecución del simulador

Si se selecciona la opción Ejecutar simulador, se despliega un nuevo cuadro con tres posibles modos de simulación:

- Ejecución normal (sin inteligencia artificial)
- Ejecución con DNN
- Ejecución con DRL

En el modo DRL, el usuario debe introducir el nombre del fichero que contiene el modelo ya entrenado. Una vez introducidos los datos necesarios, se pulsa el botón de Ejecutar, y el sistema lanza el simulador correspondiente. El estado de la simulación se muestra en tiempo real en la consola de cmd,

con mayor o menor detalle si se activa o no el parámetro Watch dentro del fichero configuration.py (Recordar que cambiar este parámetro implica recompilar el empaquetado de pyinstaller para que surta efecto, en caso de que estemos ejecutando mediante .exe, este proceso se detalla en el apartado 4 del apéndice D).

Al finalizar la ejecución, se imprimen métricas finales, como uso medio de ancho de banda, colas acumuladas, tiempos de espera, etc. Y se muestran un par de tablas con información detallada sobre el comportamiento de la red y la toma de decisiones del sistema E.6.

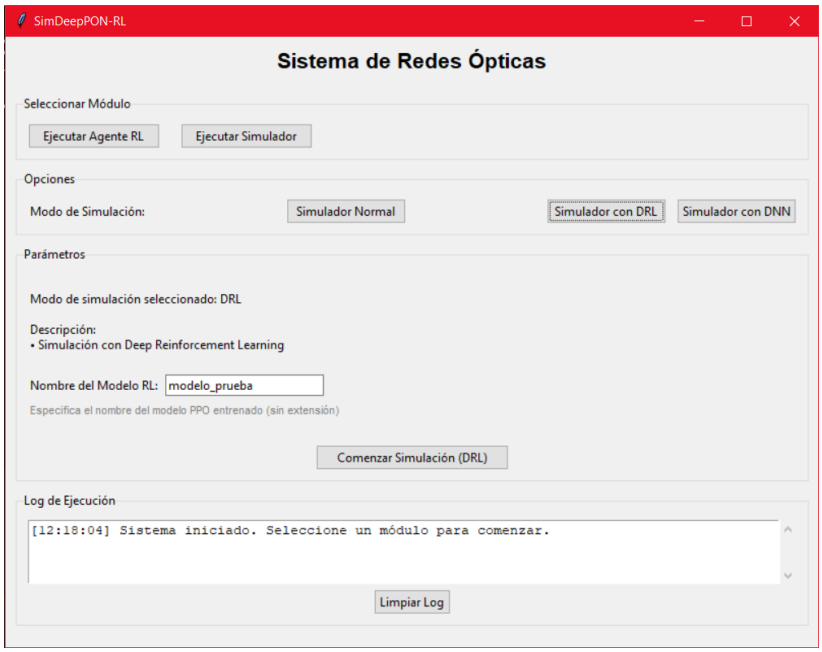

Figura E.5: Opciones de ejecución del simulador y datos entrada DRL

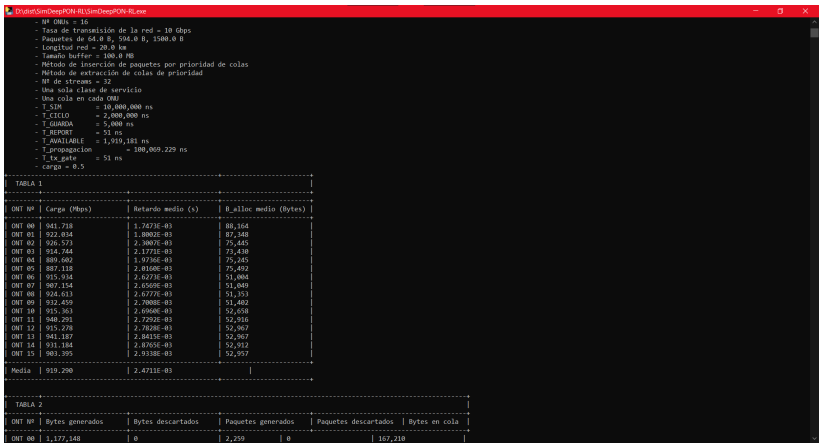

Figura E.6: Resultados del simulador

Apéndice F

Anexo de sostenibilización curricular

F.1. Introducción

En este apartado se analiza la relación entre el proyecto desarrollado y los Objetivos de Desarrollo Sostenible (ODS) definidos por la Organización de las Naciones Unidas en la Agenda 2030. Aunque el presente trabajo se enmarca principalmente en el ámbito tecnológico, su aplicación e impacto pueden alinearse con varios de estos objetivos, contribuyendo de forma indirecta al desarrollo de infraestructuras más eficientes, sostenibles e inclusivas.

A continuación se detallan aquellos ODS que guardan una mayor relación con el proyecto, junto con una breve justificación de dicha vinculación.

F.2. Objetivos de Desarrollo Sostenible

ODS 9: Industria, innovación e infraestructura

El ODS 9 promueve la construcción de infraestructuras resilientes, la industrialización sostenible y el fomento de la innovación[10]. En este sentido, considero que mi proyecto se alinea directamente con este objetivo, ya que contribuye a la mejora de infraestructuras digitales a través de la aplicación de técnicas de inteligencia artificial.

Figura F.1: Objetivo de Desarrollo Sostenible 9

Mi trabajo se ha centrado en mejorar la eficiencia y la inteligencia de las redes ópticas pasivas (PON), una tecnología ampliamente utilizada en telecomunicaciones. Al integrar un agente de aprendizaje por refuerzo profundo (DRL) capaz de tomar decisiones optimizadas sobre el reparto de ancho de banda, estoy proponiendo una solución innovadora que permite aprovechar mejor los recursos disponibles y anticiparse dinámicamente a las condiciones de red. Esto supone una clara mejora sobre los métodos tradicionales de asignación, que suelen ser estáticos o poco adaptativos.

Por tanto, el proyecto no solo mejora el rendimiento de una infraestructura clave, sino que lo hace mediante el uso de tecnologías emergentes, fomentando así la innovación tecnológica en el sector de las telecomunicaciones.

ODS 11: Ciudades y comunidades sostenibles

Este objetivo de desarrollo sostenible persigue lograr asentamientos humanos inclusivos, seguros, resilientes y sostenibles, promoviendo infraestructuras más eficientes e inteligentes.[11] Aunque este objetivo está más enfocado al urbanismo y la planificación de ciudades, las redes de telecomunicaciones son un componente esencial para que las ciudades sean realmente inteligentes, inclusivas y sostenibles. En este contexto, mi proyecto puede tener un impacto indirecto pero relevante.

Figura F.2: Objetivo de Desarrollo Sostenible 11

La gestión eficiente del ancho de banda en redes PON es clave en entornos urbanos densamente conectados, donde la demanda de tráfico es alta y varía constantemente. La capacidad de adaptar dinámicamente los recursos de red,

gracias a la intervención del agente inteligente, puede contribuir a mejorar la calidad del servicio, reducir la latencia y evitar congestiones en horas punta.

Esto favorece el desarrollo de ciudades más conectadas y resilientes, donde los servicios digitales (educación, administración electrónica, salud, transporte, etc.) pueden operar de forma fluida y fiable. En definitiva, mi propuesta ayuda a fortalecer la infraestructura digital sobre la que se apoyan muchas funciones urbanas modernas.

ODS 12: Producción y consumo responsables

Este objetivo busca garantizar el uso eficiente de los recursos naturales y reducir el impacto ambiental asociado al desarrollo tecnológico y económico. [12] Uno de los pilares de este proyecto ha sido precisamente la optimización del uso de los recursos de red. En lugar de aplicar políticas fijas o conservadoras de asignación de ancho de banda, el agente DRL toma decisiones informadas que tienen en cuenta el estado actual de la red y las necesidades reales de cada punto de acceso.

Figura F.3: Objetivo de Desarrollo Sostenible 12

Este enfoque contribuye a evitar el despilfarro de recursos técnicos y permite adaptar la infraestructura existente a una demanda cambiante, sin necesidad de ampliaciones innecesarias. Además, el uso eficiente del ancho de banda también puede implicar un menor consumo energético, tanto en los equipos de red como en los nodos centrales.

De esta forma, considero que el proyecto promueve una forma más responsable y sostenible de gestionar recursos tecnológicos, en línea con los principios del ODS 12, que aboga por modelos de producción y consumo que minimicen el impacto ambiental y optimicen el uso de los medios disponibles.

ODS 13: Acción por el clima

El ODS 13 de la ONU, Acción por el Clima, promueve medidas urgentes para combatir el cambio climático y sus impactos. [13] Aunque no ha sido un objetivo explícito del proyecto, soy consciente de que las tecnologías de la información y las telecomunicaciones tienen un impacto directo en el consumo energético global. En este sentido, cualquier mejora en la eficiencia de las redes, por pequeña que sea, puede contribuir a una reducción del consumo eléctrico asociado a la infraestructura digital.

Figura F.4: Objetivo de Desarrollo Sostenible 13

El hecho de que el sistema propuesto permita una asignación de recursos más ajustada a la demanda y evite cargas innecesarias en la red implica, potencialmente, una menor utilización de energía, especialmente si se aplica a gran escala. Este tipo de optimizaciones, cuando se aplican en redes comerciales o entornos empresariales, pueden tener un impacto acumulativo significativo.

Por tanto, aunque mi trabajo no aborda directamente la sostenibilidad ambiental, considero que sí contribuye de forma indirecta al ODS 13, en tanto que fomenta el desarrollo de soluciones tecnológicas más eficientes y conscientes del uso de recursos.

Bibliografía

[1] Scrum. https://www.scrum.org/.

[2] Tkinter documentation. https://docs.python.org/es/3.13/library/tkinter.html.

[3] Pyinstaller manual. https://pyinstaller.org/en/stable/.

[4] Glassdoor. https://www.glassdoor.es/Sueldos/programador-junior-sueldo-SRCH_KO0,18.htm.

[5] Gobierno de España. Ley de propiedad intelectual. https://www.boe.es/buscar/act.php?id=BOE-A-1996-8930.

[6] Python history and license. https://docs.python.org/3/license.html.

[7] Unión Europea. Reglamento general de protección de datos. https://eur-lex.europa.eu/ES/legal-content/summary/general-data-protection-regulation-gdpr.html.

[8] Gobierno de España. Ley orgánica de protección de datos personales y garantía de derechos digitales. https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673.

[9] 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.

[10] Orgnaización de Naciones Unidas. Objetivo de desarrollo sostenible 9. https://www.un.org/sustainabledevelopment/es/infrastructure/.

[11] Orgnaización de Naciones Unidas. Objetivo de desarrollo sostenible 9. https://www.un.org/sustainabledevelopment/es/cities/.

[12] Orgnaización de Naciones Unidas. Objetivo de desarrollo sostenible 9. http://www.un.org/sustainabledevelopment/es/sustainable-consumption-production/.

[13] Orgnaización de Naciones Unidas. Objetivo de desarrollo sostenible 9. http://www.un.org/sustainabledevelopment/es/climate-change-2/.