

Resumen

Este proyecto explorará una estrategia de aprendizaje por refuerzo orientada a reducir los costos computacionales y energéticos mediante la reutilización de conocimientos previamente adquiridos entre distintos agentes en entornos simulados de *MuJoCo*. La metodología consistirá en entrenar un modelo base (*Walker2D*) para una tarea de locomoción, que posteriormente se adaptará a agentes más complejos como *Humanoid* y *HumanoidStandup* mediante técnicas de aprendizaje por transferencia.

La hipótesis central será que este enfoque permitirá alcanzar un rendimiento comparable o superior con menos pasos de entrenamiento y un menor consumo de recursos, en comparación con el entrenamiento desde cero. Para llevar a cabo los experimentos, se empleará el algoritmo *Proximal Policy Optimization*, utilizando la librería *Stable-Baselines3* en Python.

Se analizarán métricas clave como la recompensa acumulada, el número de pasos necesarios, el tiempo total de entrenamiento y una estimación del consumo energético. Se espera que el uso de un modelo base preentrenado permita reducir significativamente el tiempo de entrenamiento sin comprometer el rendimiento final. Esta aproximación contribuirá al desarrollo de sistemas inteligentes más sostenibles y escalables en los ámbitos de la robótica y la inteligencia artificial.

Palabras clave: aprendizaje por refuerzo; reutilización del conocimiento; entornos simulados; aprendizaje por transferencia; robótica; inteligencia artificial

Resum

Aquest projecte explorarà una estratègia d'aprenentatge per reforç orientada a reduir els costos computacionals i energètics mitjançant la reutilització de coneixements prèviament adquirits entre diferents agents en entorns simulats de *MuJoCo*. La metodologia consistirà a entrenar un model base (*Walker2D*) per a una tasca de locomoció, que posteriorment s'adaptarà a agents més complexos com *Humanoid* i *HumanoidStandup* mitjançant tècniques d'aprenentatge per transferència.

La hipòtesi central és que aquest enfocament permetrà assolir un rendiment comparable o superior amb menys passos d'entrenament i un menor consum de recursos, en comparació amb l'entrenament des de zero. Per a dur a terme els experiments, s'emprarà l'algoritme *Proximal Policy Optimization*, utilitzant la llibreria *Stable-Baselines3* en Python.

S'analitzaran mètriques clau com la recompensa acumulada, el nombre de passos necessaris, el temps total d'entrenament i una estimació del consum energètic. S'espera que l'ús d'un model base preentrenat permeta reduir significativament el temps d'entrenament sense comprometre el rendiment final. Aquest enfocament contribuirà al desenvolupament de sistemes intel·ligents més sostenibles i escalables en els àmbits de la robòtica i la intel·ligència artificial.

Paraules clau: aprenentatge per reforç; reutilització del coneixement; entorns simulats; aprenentatge per transferència; robòtica; intel·ligència artificial

Abstract

This project will explore a reinforcement learning strategy aimed at reducing computational and energy costs through the reuse of previously acquired knowledge between different agents in simulated MuJoCo environments. The methodology will consist of training a base model (*Walker2D*) for a locomotion task, which will then be adapted to

more complex agents such as *Humanoid* and *HumanoidStandup* using transfer learning techniques.

The central hypothesis is that this approach will achieve comparable or superior performance with fewer training steps and lower resource consumption compared to training from scratch. The experiments will be carried out using the Proximal Policy Optimization algorithm, implemented with the Stable-Baselines3 library in Python.

Key metrics such as accumulated reward, number of required steps, total training time, and estimated energy consumption will be analyzed. It is expected that starting from a pretrained base model will significantly reduce training time without compromising final performance. This approach will contribute to the development of more sustainable and scalable intelligent systems in the fields of robotics and artificial intelligence.

Key words: reinforcement learning; knowledge reuse; simulated environments; transfer learning; robotics; artificial intelligence

Índice general

Índice general	3
Índice de figuras	7
Índice de tablas	7
1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	2
1.3 Metodología	3
1.4 Impacto esperado	5
2 Estado del arte	7
2.1 Aprendizaje por refuerzo en entornos simulados	7
2.2 Principales algoritmos de aprendizaje por refuerzo	8
2.2.1 <i>Proximal Policy Optimization (PPO)</i>	8
2.2.2 <i>Twin Delayed Deep Deterministic Policy Gradient (TD3)</i>	9
2.2.3 <i>Soft Actor-Critic (SAC)</i>	9
2.3 Transfer Learning en aprendizaje por refuerzo	9
2.3.1 Tipos de transferencia en RL	9
2.3.2 Desafíos del <i>transfer learning</i> en RL	10
2.3.3 Aplicaciones y resultados en entornos simulados	10
2.4 <i>Fine-Tuning</i> en aprendizaje por refuerzo	10
2.4.1 Ventajas del <i>fine-tuning</i> en RL	10
2.4.2 Técnicas de <i>fine-tuning</i> en RL	11
2.4.3 Retos específicos en <i>fine-tuning</i> para RL	11
2.4.4 Relevancia en este proyecto	11
2.5 Técnicas específicas de <i>fine-tuning</i> y transferencia	12
2.5.1 <i>Frozen layers</i>	12
2.5.2 <i>Curriculum learning</i>	12
2.5.3 <i>Distillation learning</i>	12
2.5.4 <i>Representation transfer</i>	12
2.5.5 <i>Weight initialization + Fine-tuning</i>	13
2.5.6 <i>Multi-task training</i> como base previa	13
2.5.7 <i>Domain adaptation</i>	13
2.5.8 Comparativa entre supervisado y RL	13
2.6 Transferencia entre agentes con diferente morfología	14
2.6.1 Casos documentados en la literatura	14
2.6.2 Técnicas usadas para pasar de Walker2D a Humanoid	14
2.6.3 Adaptación de espacios de observación y acción	14
2.6.4 Similitud estructural como ventaja para transferencia	15
2.7 Crítica del estado del arte	15
2.7.1 Puntos débiles identificados	16
2.7.2 Puntos fuertes aprovechables	16
2.7.3 Síntesis final	17

3 Análisis del problema	19
3.1 Análisis de seguridad	19
3.1.1 Seguridad del entrenamiento y ejecución en simulación	19
3.1.2 Riesgos en la reutilización del modelo base	19
3.1.3 Perspectiva de seguridad futura en despliegues físicos	20
3.2 Análisis energético	20
3.2.1 Contexto y motivación	20
3.2.2 Estrategias de eficiencia implementadas	20
3.2.3 Métricas consideradas	21
3.2.4 Relevancia en el marco de la sostenibilidad en IA	22
3.3 Análisis del marco legal y ético	22
3.3.1 Marco legal	22
3.3.2 Consideraciones éticas	22
3.3.3 Conclusión	23
3.4 Análisis de riesgos	23
3.4.1 Identificación de riesgos	23
3.4.2 Enfoque de mitigación general	23
3.5 Análisis de los entornos utilizados	24
3.6 Identificación y análisis de soluciones posibles	25
3.7 Solución propuesta	27
3.7.1 Herramientas y entornos	27
3.7.2 Justificación de las decisiones técnicas	28
4 Diseño de la solución	29
4.1 Arquitectura del proyecto	29
4.2 Recompensas uniformes	30
4.3 Diseño detallado	30
4.3.1 Walker2d	30
4.3.2 Humanoid benchmark desde 0	32
4.3.3 HumanoidStandup benchmark desde 0	33
4.3.4 Fine-Tuning en Humanoid-v5	34
4.3.5 Fine-Tuning en HumanoidStandup-v5	35
4.4 Justificación de decisiones técnicas y de implementación	37
4.4.1 Frecuencia de evaluación durante el entrenamiento	37
4.4.2 Estrategia de transferencia: <i>Weight Initialization</i>	38
4.4.3 Elección de la política: <i>MlpPolicy</i> y <i>ActorCriticPolicy</i>	39
4.4.4 Uso de hiperparámetros en PPO	40
4.4.5 Técnicas de congelación y descongelado progresivo	41
5 Desarrollo de la solución propuesta	43
5.1 Preparación del entorno de trabajo	43
5.1.1 Lenguaje y entorno de programación	43
5.1.2 Gestión del entorno y dependencias	44
5.1.3 Hardware utilizado	44
5.1.4 Estructura de experimentación	45
5.2 Estructura del código y organización funcional	45
5.2.1 Scripts principales de entrenamiento	45
5.2.2 Callbacks personalizados embebidos	46
5.2.3 Monitorización integrada por script	46
5.2.4 Justificación de la estructura	47
5.3 Implementación del modelo base	47
5.3.1 Resultados reales del modelo base: comparación y análisis	48
5.4 Implementación del modelo benchmark	49
5.4.1 Entrenamiento benchmark en Humanoid-v5	49

5.4.2	Entrenamiento benchmark en HumanoidStandup-v5	49
5.4.3	Resultados benchmark Humanoid-v5	50
5.4.4	Resultados benchmark en HumanoidStandup-v5	54
5.5	Implementación del fine-tuning	57
5.5.1	Resultados fine-tuning en Humanoid-v5	57
5.5.2	Resultados fine-tuning en HumanoidStandup-v5	61
5.6	Comparación con estado del arte de benchmarks externos	65
5.7	Resumen del desarrollo realizado	70
6	Conclusiones	71
6.1	Relación del trabajo desarrollado con los estudios cursados	71
6.2	Limitaciones del proyecto	72
6.2.1	Uso exclusivo del algoritmo PPO	72
6.2.2	Análisis limitado de variabilidad estadística	72
6.2.3	Diversidad limitada en tareas y morfologías	72
6.2.4	Limitaciones del fine-tuning en aprendizaje por refuerzo	72
6.3	Trabajos futuros	73
6.3.1	Expansión jerárquica de acciones y árboles de comportamiento . .	73
6.3.2	Aplicación de nuevas técnicas de transferencia	73
6.3.3	Exploración de algoritmos alternativos a PPO	74
6.3.4	Conclusiones	74
7	Bibliografía	75
8	Anexos	79
Anexo A.	Configuración del sistema	79
A.1.	Creación del entorno Conda	79
A.2.	Instalación de librerías principales	79
Anexo B.	Códigos utilizados	79
B.1.	Código fuente: Entrenamiento desde cero en Walker2d-v5	79
B.2.	Código fuente: Entrenamiento desde cero en Humanoid-v5	82
B.3.	Código fuente: Entrenamiento desde cero en HumanoidStandup-v5 ..	84
B.4.	Código fuente: Fine-tuning con Walker2d-v5 en el entorno Humanoid-v5	86
B.5.	Código fuente: Fine-tuning con Walker2d-v5 en el entorno HumanoidStandup-v5	90
Anexo C.	Relación del trabajo con los ODS	93
Anexo C.1.	Reflexión sobre la relación del TFG con los ODS más relevantes	93

Índice de figuras

4.1 Arquitectura del proyecto	29
5.1 Modelo Base Walker2d	49
5.2 Humanoid Benchmark 2M	50
5.3 Humanoid Benchmark 5M	50
5.4 Humanoid Benchmark 10M	51
5.5 Humanoid Benchmark WrapUp	51
5.6 Domain random humanoid benchmark 2M	53
5.7 Domain random humanoid benchmark 5M	53
5.8 Domain random humanoid benchmark 10M	54
5.9 Standup benchmark 2M	55
5.10 Standup benchmark 5M	55
5.11 Standup benchmark 10M	56
5.12 Fine-tuning humanoid 2M	57
5.13 Fine-tuning humanoid 5M	57
5.14 Fine-tuning humanoid 10M	58
5.15 Comparación del consumo de energía de Humanoid	58
5.16 Comparación del tiempo de entrenamiento de Humanoid	59
5.17 Comparación de la recompensa por energía usada del modelo Humanoid	59
5.18 Fine-tuning Standup 2M	61
5.19 Fine-tuning Standup 5M	61
5.20 Fine-tuning Standup 10M	62
5.21 Comparación del consumo de energía de HumanoidStandup	62
5.22 Comparación del tiempo de entrenamiento de HumanoidStandup	63
5.23 Comparación de la recompensa por energía usada del modelo HumanoidStandup	63
5.24 Comparación del modelo Walker2d con benchmark externos	65
5.25 Comparación del modelo HumanoidStandup con benchmark externos	66
5.26 Comparación del modelo Humanoid con benchmark externos	67
5.27 Proyección con benchmarks externos	69

Índice de tablas

3.1 Tabla de riesgos identificados y estrategias de mitigación	23
5.1 Comparación entre modelos base y benchmark en Walker2d-v5	48
5.2 Recompensa total y consumo energético para el modelo Humanoid desde cero y desde cero con domain randomization.	54

5.3	Recompensa total y consumo energético estimado para los modelos entrenados desde cero y con fine-tuning.	60
5.4	Eficiencia energética estimada para los modelos entrenados desde cero y con fine-tuning.	61
5.5	Recompensa total y consumo energético estimado para los modelos entrenados desde cero y fine-tuned.	64
5.6	Comparación de eficiencia energética y consumo estimado entre modelos benchmark y fine-tuned en HumanoidStandup-v5.	65
5.7	Comparativa de configuraciones en términos de recompensa y consumo energético	68
1	Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).	93

CAPÍTULO 1

Introducción

En los últimos años, el aprendizaje por refuerzo profundo ha emergido como una de las ramas más prometedoras de la inteligencia artificial, especialmente en aplicaciones como la robótica, el control autónomo y la simulación física. Gracias a entornos simulados como MuJoCo, es posible entrenar agentes artificiales capaces de aprender habilidades complejas, como caminar, mantenerse en equilibrio o interactuar con objetos, exclusivamente a partir de la experiencia.

Sin embargo, el coste computacional y energético asociado a estos entrenamientos es elevado. Entrenar desde cero un agente para cada tarea o morfología implica millones de pasos de simulación y un consumo significativo de recursos. Esta situación plantea importantes desafíos en términos de sostenibilidad, escalabilidad y reutilización del conocimiento, especialmente a medida que los modelos crecen en complejidad.

En este contexto, el presente proyecto aborda la hipótesis de que es posible reducir el esfuerzo computacional y energético mediante el uso de modelos base preentrenados, reutilizables en tareas similares. Específicamente, se investiga la viabilidad de aplicar técnicas de aprendizaje por transferencia y fine-tuning en entornos *MuJoCo*, aprovechando conocimientos previamente adquiridos en agentes más simples para acelerar el entrenamiento de agentes más complejos.

El objetivo principal del proyecto es desarrollar un sistema que permita aplicar modelos base entrenados en tareas de locomoción a nuevos agentes con distintas morfologías, evaluando el impacto en términos de recompensa obtenida, velocidad de convergencia y consumo energético. A través de una metodología experimental comparativa, se analizan diferentes escenarios con y sin transferencia, midiendo no solo el rendimiento sino también la eficiencia energética estimada. Los resultados obtenidos permiten extraer conclusiones relevantes sobre las condiciones bajo las cuales el *fine-tuning* resulta efectivo, así como sus limitaciones.

Este trabajo contribuye al desarrollo de agentes inteligentes más sostenibles y modulares, y sienta las bases para futuras investigaciones orientadas a la reutilización sistemática del conocimiento en aprendizaje por refuerzo.

1.1 Motivación

En la última década, el uso de técnicas de aprendizaje por refuerzo profundo ha permitido no solo desarrollar agentes capaces de resolver tareas complejas de forma autónoma en entornos simulados [1], sino también está empezando a ser utilizado para complementar el entrenamiento del modelo estrella de inteligencia artificial, los *Large Language Models* [2]. En este caso nos centraremos en los entornos simulados donde uno

de los marcos más extendidos para este tipo de investigación es OpenAI Gym¹, el cual implementa simuladores físicos como MuJoCo², especialmente útiles para tareas de locomoción en robótica. Sin embargo, los métodos actuales suelen requerir largos tiempos de entrenamiento y un uso intensivo de recursos computacionales, incluso en tareas que, desde un punto de vista estructural, comparten patrones similares entre distintos agentes o entornos.

Por ello, desde una perspectiva académica, el trabajo responde al interés por mejorar la eficiencia y generalización de los algoritmos de aprendizaje por refuerzo. La capacidad de una política entrenada para transferirse y adaptarse a nuevos entornos o morfologías se considera una de las claves para el desarrollo de agentes más inteligentes y versátiles. Así, este proyecto pretende aportar evidencia empírica sobre el valor de estrategias como el fine-tuning[3] y el curriculum learning[4] aplicadas al aprendizaje por refuerzo, explorando su capacidad para mejorar el rendimiento respecto al entrenamiento desde cero y fomentar una reutilización eficiente del conocimiento.

A nivel personal, la motivación principal de este proyecto surge de una creciente preocupación por el impacto ambiental y energético del entrenamiento de modelos de inteligencia artificial [5]. Con la proliferación de modelos de gran escala, como los Large Language Models, el consumo energético asociado al entrenamiento de sistemas de IA se ha disparado, generando un impacto global que puede volverse insostenible en los próximos años. Aunque en este proyecto se abordan entornos más reducidos y específicos, como MuJoCo, la idea es aprovechar modelos preentrenados [6] para reducir el coste computacional y energético en el aprendizaje en entornos simulados.

1.2 Objetivos

Tanto la motivación personal como la académica han influido en la definición de los objetivos , que se sitúa en la intersección entre la eficiencia computacional y el aprovechamiento del conocimiento previo en el ámbito del aprendizaje por refuerzo.

El objetivo principal del proyecto es el desarrollo de una librería de modelos base reutilizables en entornos MuJoCo, dentro del marco de OpenAI Gym. Esta librería permitirá aplicar una política previamente entrenada (modelo base) para una determinada acción —como caminar— a distintos agentes o morfologías, con el fin de facilitar su entrenamiento posterior en tareas similares.

Este enfoque busca no solo la reutilización del conocimiento, sino también demostrar que, mediante el uso de un modelo base preentrenado, es posible reducir el coste computacional y energético asociado al entrenamiento de nuevos agentes, al mismo tiempo que se mantiene o incluso se mejora el rendimiento alcanzado respecto a modelos entrenados desde cero.

A partir de este objetivo general, se derivan los siguientes **objetivos específicos**:

- Diseñar e implementar un modelo base que aprenda de forma eficaz una acción determinada (p. ej., caminar) en un entorno sencillo como Walker2D.
- Desarrollar un sistema que permita reutilizar dicho modelo base para el entrenamiento de otros agentes más complejos, como Humanoid mediante técnicas de *fine-tuning*, así como generalizar para otros entornos similares.

¹<https://github.com/openai/gym>

²<https://mujoco.org/>

- Evaluar el rendimiento de los modelos entrenados a partir del modelo base, en comparación con modelos entrenados desde cero, utilizando métricas como:
 - Recompensa media alcanzada
 - Número de episodios o pasos necesarios para alcanzar un rendimiento determinado
 - Tiempo total de entrenamiento
 - Estimación del consumo energético
- Implementar el modelo base en una estructura reutilizable (librería o módulo funcional) que facilite su uso en futuras investigaciones o experimentos.

1.3 Metodología

La metodología empleada en este proyecto se basa en un enfoque experimental, comparativo y centrado en la eficiencia, cuyo objetivo principal es evaluar distintas estrategias de *fine-tuning* en tareas de locomoción en entornos simulados MuJoCo de OpenAI Gym. La meta es demostrar que el uso de modelos base preentrenados puede ofrecer ventajas significativas en términos de rendimiento, consumo computacional y eficiencia energética, frente a entrenamientos tradicionales realizados desde cero.

Este estudio se desarrollará exclusivamente con el algoritmo *Proximal Policy Optimization* [7], haciendo uso de su implementación en la librería Stable-Baselines³ en Python⁴. PPO ha sido seleccionado por ser un método robusto, ampliamente utilizado en tareas de control continuo y con una buena relación entre estabilidad, rendimiento y eficiencia.

El proceso metodológico se estructura en varias fases clave:

1. **Selección de entornos y definición de tareas:** la selección de entornos para este proyecto no ha sido aleatoria ni únicamente guiada por benchmarks estándar, sino que se ha basado en una metodología estructurada orientada a la descomposición funcional del comportamiento. En concreto, se ha seguido un criterio jerárquico que distingue entre **acciones principales** y **subacciones**, considerando cada entorno como una instancia representativa de una capacidad motriz específica.

El entorno Walker2d⁵ representa de forma aislada la acción principal de caminar en un plano bidimensional, mientras que Humanoid⁶ incorpora esta misma acción dentro de una morfología más compleja con mayores grados de libertad. A su vez, HumanoidStandup⁷ se considera una subacción relevante para caminar, como es la capacidad de incorporarse desde el suelo. Esta organización permite estudiar tanto la transferencia horizontal (entre agentes que comparten la misma acción) como la vertical (entre acciones y sus subcomponentes), la cual aporta generalización a nuestros modelos para que sean más flexibles y adaptables en diferentes entornos.

Este enfoque no solo estructura el análisis realizado en el presente trabajo, sino que se propone como base para futuras extensiones. La idea es construir un **árbol de acciones**, donde cada nueva tarea se incorpore como una rama derivada de habilidades más simples, permitiendo una expansión controlada y lógica del sistema.

³<https://stable-baselines.readthedocs.io/en/master/>

⁴<https://www.python.org/>

⁵<https://gymnasium.farama.org/environments/mujoco/walker2d/>

⁶<https://gymnasium.farama.org/environments/mujoco/humanoid/>

⁷https://gymnasium.farama.org/environments/mujoco/humanoid_standup/

Esta taxonomía funcional facilita además la reutilización de agentes base en tareas nuevas, maximizando la eficiencia del aprendizaje por transferencia [8].

2. **Entrenamiento del modelo base:** se entrenará un agente desde cero en el entorno Walker2D utilizando PPO. Este modelo actuará como red base o punto de partida para su adaptación a otros entornos. Durante este entrenamiento se documentarán métricas clave como el número de pasos, la recompensa media obtenida, el tiempo de entrenamiento y el consumo energético estimado.
3. **Diseño de técnicas de *fine-tuning*:** con el modelo base entrenado, se explorarán diversas estrategias de adaptación del conocimiento a otros entornos, incluyendo:
 - Reentrenamiento completo del modelo.
 - Congelación parcial de capas del modelo (por ejemplo, capas iniciales o intermedias), entrenando únicamente las capas finales.
 - *Fine-tuning* mediante tasas de aprendizaje diferenciadas por capas (*layer-wise learning rates*).
 - Enfoques inspirados en *curriculum learning*, transfiriendo conocimiento progresivamente de tareas más simples a tareas más complejas.

Estas técnicas se seleccionarán y adaptarán en base a literatura del estado del arte, priorizando aquellas que puedan facilitar una reutilización eficiente del conocimiento ya aprendido.

4. **Aplicación del *fine-tuning* y adaptación a nuevos entornos:** se llevará a cabo el proceso de *fine-tuning* en los entornos Humanoid y HumanoidStandup, reutilizando el modelo base preentrenado en Walker2D. En cada caso, se entrenará el agente con las distintas técnicas descritas anteriormente.
5. **Entrenamiento desde cero como línea base (*baseline*):** para evaluar de manera objetiva los beneficios del *fine-tuning*, se entrenarán agentes completamente desde cero en Humanoid y HumanoidStandup, utilizando los mismos hiperparámetros, condiciones y recursos computacionales.
6. **Evaluación comparativa y análisis de eficiencia:** se recopilarán métricas clave para cada experimento, incluyendo:
 - Recompensa acumulada media.
 - Número total de pasos de entrenamiento requeridos.
 - Tiempo total de entrenamiento.
 - Estimación del consumo energético (calculado a partir del tiempo de entrenamiento, utilización de hardware y consumo medio estimado).
 - Estabilidad del aprendizaje y convergencia.

Se utilizarán herramientas de registro y visualización como Matplotlib⁸ para facilitar el análisis visual del proceso de aprendizaje.

El análisis de la eficiencia se llevará a cabo con la estimación del consumo energético se basa en un sistema de monitorización asíncrona embebida en los scripts de entrenamiento, que registra el uso medio de CPU y GPU usando *psutil* y *GPUtil*. A partir de estos valores y la potencia nominal del hardware, se calcula el consumo total en kWh mediante una fórmula detallada en el capítulo 5.2.3.

⁸<https://matplotlib.org/>

7. **Análisis cualitativo y cuantitativo de resultados:** los resultados serán analizados tanto a nivel estadístico como cualitativo, identificando patrones, fortalezas y limitaciones de cada técnica aplicada. Se buscará determinar si existe una técnica de *fine-tuning* que proporcione un balance óptimo entre rendimiento, coste computacional y eficiencia energética.
8. **Reflexión final y propuesta de mejoras:** con base en los resultados obtenidos, se elaborará una reflexión final sobre la viabilidad del uso de modelos base en tareas de aprendizaje por refuerzo. Asimismo, se propondrán mejoras y futuras líneas de trabajo que podrían incluir la incorporación de nuevas técnicas de transferencia, uso de arquitecturas más eficientes o aplicación en otros dominios.

1.4 Impacto esperado

Este proyecto pretende generar un impacto significativo en varias dimensiones del ámbito académico y tecnológico, especialmente en el campo del aprendizaje por refuerzo aplicado a la simulación de agentes en entornos MuJoCo. Los resultados esperados de este proyecto no solo aportan valor desde el punto de vista del rendimiento y la eficiencia técnica, sino que también promueven una aproximación más sostenible y escalable en el desarrollo de sistemas inteligentes.

Desde una perspectiva técnica, se espera que el uso de modelos base preentrenados facilite la transferencia efectiva de conocimiento entre distintos entornos y tareas de locomoción, demostrando que es posible alcanzar niveles de rendimiento similares o superiores con un menor número de recursos. Esto implicaría una mejora significativa en términos de eficiencia computacional y una reducción sustancial en el consumo de recursos, tanto en tiempo como en energía, frente a entrenamientos desde cero.

A nivel académico, el estudio puede servir como una base sólida para futuras investigaciones en estrategias de *fine-tuning* dentro del campo del aprendizaje por refuerzo, aportando evidencia empírica sobre la viabilidad y utilidad de técnicas como la concatenación de capas o el *curriculum learning*. Además, la metodología rigurosa aplicada y la comparativa exhaustiva entre enfoques permitirán generar conclusiones reproducibles y extrapolables a otros entornos o tipos de tareas.

Desde el punto de vista medioambiental y ético, el trabajo plantea una reflexión sobre el coste energético creciente asociado al entrenamiento de modelos de inteligencia artificial. Al centrarse en optimizar este aspecto, se contribuye a la búsqueda de soluciones más sostenibles que puedan mitigar el impacto ecológico de la IA, un aspecto cada vez más relevante en un contexto de expansión exponencial de los modelos de gran escala [9].

Por último, el impacto esperado se extiende también al ámbito educativo y profesional, ya que el desarrollo de una librería o sistema reutilizable puede facilitar el trabajo de otros investigadores, estudiantes o empresas que deseen implementar soluciones eficientes sin partir de cero. Esta reutilización del conocimiento previo es coherente con los principios de modularidad, escalabilidad y sostenibilidad en la ingeniería de software aplicada a la inteligencia artificial.

CAPÍTULO 2

Estado del arte

En los últimos años, el aprendizaje por refuerzo ha demostrado ser una de las ramas más prometedoras[10] de la inteligencia artificial, especialmente en tareas de control, robótica y toma de decisiones en entornos complejos. Sin embargo, uno de los principales desafíos actuales del RL es su enorme costo computacional y energético, especialmente cuando los modelos se entrena n desde cero en cada nueva tarea o entorno.

Frente a esta problemática, el aprendizaje por transferencia y el fine-tuning emergen como alternativas capaces de reutilizar conocimiento adquirido previamente, reduciendo así el número de episodios necesarios para alcanzar un rendimiento óptimo, lo que se traduce directamente en una mayor eficiencia en términos de tiempo, energía y recursos. Esta necesidad se vuelve aún más relevante cuando se consideran entornos físicos simulados como MuJoCo, ampliamente utilizados en investigación para probar agentes de RL en tareas de locomoción y manipulación.

El presente estado del arte tiene como objetivo contextualizar el proyecto en el panorama actual de la investigación en RL, explorando tanto los algoritmos más empleados como las técnicas más avanzadas de transferencia y *fine-tuning*, así como su aplicación concreta a entornos de simulación como los ofrecidos por OpenAI Gym con backend en MuJoCo. Se analizarán enfoques recientes que han buscado optimizar la eficiencia y escalabilidad de los agentes de RL, especialmente en el contexto de transferencia entre agentes con diferentes morfologías, como Walker2D o Humanoid.

Este análisis proporcionará una base sólida sobre la que justificar las decisiones técnicas y metodológicas adoptadas a lo largo del proyecto, incluyendo la elección del algoritmo PPO como motor de entrenamiento, y el diseño de un modelo base capaz de generalizar acciones específicas como caminar o saltar a diferentes estructuras corporales, manteniendo la eficiencia y el rendimiento.

2.1 Aprendizaje por refuerzo en entornos simulados

El desarrollo de algoritmos de aprendizaje por refuerzo requiere entornos de prueba controlados, reproducibles y lo suficientemente complejos como para permitir la validación de estrategias generalizables. En este contexto, los entornos simulados se han consolidado como herramientas fundamentales en la investigación de RL, al permitir la experimentación con múltiples agentes y tareas sin incurrir en los riesgos o costos asociados al mundo físico.

Uno de los entornos más utilizados en la actualidad es MuJoCo, un simulador de física de alta precisión optimizado para tareas de robótica y biomecánica. MuJoCo destaca por su capacidad para simular interacciones físicas complejas con gran realismo y

eficiencia, lo que lo convierte en una herramienta ideal para tareas de locomoción, control de equilibrio y planificación motora. Además, su integración con OpenAI Gym, una interfaz estándar de entornos de RL en Python, ha facilitado la creación de benchmarks ampliamente aceptados por la comunidad científica [11] [12] [13].

La elección de estos entornos permite no solo evaluar el rendimiento de los algoritmos en distintos niveles de complejidad, sino también estudiar la capacidad de transferencia de políticas entre ellos, especialmente en tareas compartidas como caminar, correr o mantener el equilibrio.

Este proyecto se enmarca dentro de esta línea de pensamiento, utilizando MuJoCo y OpenAI Gym como plataforma base para evaluar la eficiencia de modelos preentrenados en una tarea concreta al ser finetuneados en agentes de mayor complejidad como Humanoid o HumanoidStandup. La idea es comprobar si es posible reutilizar conocimiento aprendido en un entorno más simple para acelerar y mejorar el entrenamiento en entornos más complejos, manteniendo una alta calidad en el comportamiento y minimizando el gasto computacional y energético.

2.2 Principales algoritmos de aprendizaje por refuerzo

El aprendizaje por refuerzo profundo ha evolucionado rápidamente gracias al desarrollo de algoritmos capaces de aprender políticas complejas directamente a partir de la interacción con el entorno. En el caso de tareas continuas de control motor, como las que plantea el entorno MuJoCo, los algoritmos más utilizados pertenecen a la clase de métodos *actor-crítico* basados en políticas. Estos métodos se dividen comúnmente en dos enfoques: *on-policy* y *off-policy*, según la relación entre la política que se optimiza y la que se utiliza para recolectar datos. A continuación, se describen brevemente ambos enfoques:

- **On-policy:** El aprendizaje *on-policy* mejora la política basándose en las acciones que esa misma política toma mientras interactúa con el entorno. Aprende de la experiencia real generada por la política actual, lo que garantiza coherencia entre lo que se hace y lo que se aprende. [14]
- **Off-policy:** El aprendizaje *off-policy* aprende sobre una política objetivo mientras ejecuta una política distinta para recolectar datos. Esto permite separar la política que actúa de la que se optimiza, facilitando la reutilización de experiencias pasadas. [14]

Finalmente, se describen brevemente los tres más representativos actualmente en la literatura: PPO, TD3 y SAC.

2.2.1. *Proximal Policy Optimization (PPO)*

PPO es uno de los algoritmos más utilizados en la actualidad debido a su equilibrio entre rendimiento, estabilidad y simplicidad de implementación. Propuesto por "Schulman et al. (2017)", PPO busca actualizar la política de forma gradual mediante una función de pérdida con clipping que impide desviaciones excesivas con respecto a la política anterior. Esta estrategia permite un entrenamiento más estable y seguro en comparación con métodos anteriores como TRPO [15], sin comprometer la eficiencia.

Una de las principales ventajas de PPO es su robustez frente a una gran variedad de entornos, así como su facilidad de ajuste y escalabilidad. Esto ha llevado a su adopción como algoritmo base en múltiples bibliotecas, como Stable-Baselines3, y lo convierte

en una elección adecuada para estudios comparativos de transferencia y fine-tuning en entornos como MuJoCo.

2.2.2. Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 [16] es una mejora sobre el algoritmo DDPG [17], orientado a tareas con espacio de acción continuo. Introduce tres mejoras clave: redes críticas dobles, actualización retardada del actor y ruido en las acciones objetivo durante el entrenamiento. Estas optimizaciones mejoran la estabilidad y el rendimiento, especialmente en tareas donde el control preciso es esencial.

TD3 ha demostrado un gran rendimiento en entornos de locomoción complejos, aunque su arquitectura y configuración pueden resultar más delicadas de ajustar en comparación con PPO.

2.2.3. Soft Actor-Critic (SAC)

SAC [18] es un algoritmo *off-policy* que maximiza no solo la recompensa esperada, sino también la entropía de la política, promoviendo así una exploración más eficiente. Esta característica lo hace especialmente atractivo en entornos con múltiples soluciones óptimas o donde la exploración inicial es crítica. Al igual que TD3, SAC opera de forma *off-policy*, lo que le permite aprovechar mejor los datos recogidos, aunque a cambio requiere una infraestructura de almacenamiento y muestreo más compleja.

En comparación con PPO, SAC suele destacar en tareas de control fino y entornos con alta variabilidad, aunque puede implicar un mayor costo computacional por episodio.

2.3 Transfer Learning en aprendizaje por refuerzo

El aprendizaje por transferencia se ha consolidado como una técnica esencial en el campo del aprendizaje automático, permitiendo reutilizar el conocimiento adquirido en una tarea fuente para acelerar o mejorar el rendimiento en una tarea destino. En el contexto del aprendizaje por refuerzo, este paradigma resulta especialmente atractivo debido a los elevados costos computacionales y energéticos asociados al entrenamiento desde cero, así como a la alta variabilidad en el rendimiento entre experimentos.

En tareas complejas y costosas como las de locomoción en entornos simulados, la posibilidad de transferir políticas, representaciones o estructuras de red entre agentes distintos —como Walker2D o Humanoid— puede suponer una ventaja significativa. El objetivo no es solo acelerar el proceso de entrenamiento, sino también mejorar la estabilidad, reducir la necesidad de exploración y minimizar el consumo de recursos.

2.3.1. Tipos de transferencia en RL

En RL, la transferencia puede darse de diversas formas, que incluyen, pero no se limitan a:

- **Transferencia de políticas** [19]: reutilización directa o parcial de una política entrenada en otro entorno o con otro agente.
- **Transferencia de representaciones** [20]: traspaso de características aprendidas en capas intermedias o embeddings útiles para tareas similares.

- **Transferencia de dinámica** [21]: compartir modelos internos del entorno o predicciones del siguiente estado.
- **Transferencia de reward shaping o exploración**[22]: compartir estructuras de recompensa o estrategias de exploración.

2.3.2. Desafíos del *transfer learning* en RL

A pesar de su potencial, aplicar aprendizaje por transferencia en RL no está exento de dificultades. Los principales retos incluyen:

- **Desajuste de distribución** [23]: las diferencias entre la tarea fuente y la destino pueden provocar que la política transferida no sea válida o incluso contraproducente.
- **Overfitting a la tarea fuente** [24]: las redes pueden especializarse demasiado en el entorno inicial, perdiendo generalización.
- **Negative transfer** [25]: en algunos casos, la transferencia puede ralentizar el aprendizaje o empeorar el rendimiento si no se gestiona correctamente.

2.3.3. Aplicaciones y resultados en entornos simulados

Numerosos trabajos recientes han abordado la transferencia entre entornos MuJoCo, principalmente con tareas de locomoción. Estos estudios [26], [27] han demostrado que es posible reutilizar políticas o capas entrenadas para adaptar tareas como caminar, saltar o mantenerse en pie entre distintas morfologías de agentes. Sin embargo, los resultados varían significativamente en función de la arquitectura utilizada, la estrategia de transferencia y el grado de similitud entre entornos.

Esta variabilidad justifica la necesidad de una exploración sistemática de técnicas de fine-tuning, que se desarrollará en profundidad en el siguiente apartado.

2.4 Fine-Tuning en aprendizaje por refuerzo

El *fine-tuning*, o ajuste fino, es una de las estrategias más comunes dentro del aprendizaje por transferencia. Consiste en tomar un modelo previamente entrenado en una tarea fuente y seguir su entrenamiento en una tarea destino, adaptando progresivamente sus parámetros a las nuevas condiciones del entorno o del agente. Esta técnica es ampliamente utilizada en aprendizaje supervisado y ha ganado terreno en RL por su simplicidad y eficacia.

En este proyecto, el *fine-tuning* juega un papel central, ya que se busca entrenar un modelo base y adaptarlo a tareas similares pero más complejas, reduciendo el coste computacional y energético respecto al entrenamiento desde cero.

2.4.1. Ventajas del *fine-tuning* en RL

El uso de *fine-tuning* en entornos de RL ofrece múltiples beneficios potenciales:

- **Reducción del tiempo de entrenamiento:** el modelo parte con conocimientos previos, lo que acelera la convergencia.

- **Menor consumo energético:** al requerir menos episodios de exploración y menos actualizaciones, se reducen los recursos computacionales empleados.
- **Mejor rendimiento inicial:** las políticas preentrenadas ofrecen un comportamiento inicial razonable, evitando exploración aleatoria ineficiente.
- **Aprovechamiento de estructuras comunes:** los agentes que comparten características pueden beneficiarse de una base común.

2.4.2. Técnicas de *fine-tuning* en RL

El *fine-tuning* puede llevarse a cabo con diferentes enfoques, según cómo se manejen los parámetros del modelo base:

- **Ajuste completo:** se continúan entrenando todos los parámetros de la red. Es la opción más flexible, pero también la más propensa al *overfitting* o a la pérdida del conocimiento original.
- **Capas congeladas [28]:** se bloquean ciertas capas de la red, preservando el conocimiento general y adaptando solo las capas finales.
- **Transferencia progresiva [20]:** se añaden nuevas columnas.^a la red que aprenden de la tarea destino mientras reciben información de las redes anteriores, evitando el olvido catastrófico.
- **Regularización de parámetros (L2/SP, EWC) [29]:** se penaliza el cambio en parámetros clave aprendidos previamente, equilibrando entre retención y adaptación.

2.4.3. Retos específicos en *fine-tuning* para RL

El RL plantea retos adicionales al *fine-tuning* respecto al aprendizaje supervisado:

- **Exploración y estabilidad:** cambiar entornos puede alterar la función de recompensa o la dinámica, dificultando el aprendizaje si no se ajusta bien la política previa.
- **Catastrophic forgetting:** durante el *fine-tuning*, el modelo puede olvidar comportamientos útiles aprendidos anteriormente.
- **Manejo de políticas subóptimas:** si la política previa era subóptima, seguir entrenándola puede perpetuar errores o hábitos no deseados.

2.4.4. Relevancia en este proyecto

En el contexto de este proyecto, se evaluarán distintas técnicas de *fine-tuning* para validar qué estrategias permiten una mejor adaptación de un modelo base a distintos agentes y tareas, con un enfoque claro en:

Este análisis guiará la elección metodológica y las decisiones técnicas del proyecto, permitiendo establecer una librería base de comportamientos reutilizables para distintas morfologías en *MuJoCo*.

2.5 Técnicas específicas de *fine-tuning* y transferencia

El aprendizaje por transferencia en aprendizaje por refuerzo no se limita únicamente a reutilizar modelos previos; existen múltiples técnicas específicas que permiten mejorar la adaptabilidad, la eficiencia del entrenamiento y el rendimiento final del agente. En este apartado se describen algunas de las estrategias más relevantes, tanto adoptadas de otros paradigmas de aprendizaje, como específicamente adaptadas a entornos de RL, destacando sus ventajas, limitaciones y casos de uso más comunes.

2.5.1. *Frozen layers*

Esta técnica consiste en congelar los pesos de ciertas capas de una red neuronal previamente entrenada durante el proceso de *fine-tuning*. En RL, esto permite conservar representaciones útiles ya aprendidas, evitando el fenómeno del *catastrophic forgetting* y reduciendo el costo computacional.

- **Ventajas:** evita modificar conocimientos útiles, permite un entrenamiento más rápido.
- **Limitaciones:** si el entorno objetivo es muy diferente, puede impedir una correcta adaptación.

2.5.2. *Curriculum learning*

El aprendizaje curricular propone una progresión gradual en la dificultad de las tareas de entrenamiento. En RL, esto se traduce en exponer al agente primero a tareas más simples, que luego se complejizan progresivamente.

- **Ventajas:** mejora la estabilidad del entrenamiento, acelera la convergencia.
- **Limitaciones:** requiere un diseño cuidadoso del “currículum” y del criterio de avance.

2.5.3. *Distillation learning*

La distillation [30] consiste en usar un modelo “maestro” entrenado para transferir su comportamiento a un modelo más pequeño o diferente, a través de la minimización de la divergencia entre sus políticas o valores. En RL puede usarse para adaptar una política aprendida en un entorno o morfología a otro agente.

- **Ventajas:** permite transferencias más suaves, aprendizaje supervisado de la política.
- **Limitaciones:** requiere que el modelo base tenga una buena generalización y que la divergencia sea bien definida.

2.5.4. *Representation transfer*

La transferencia de representaciones [31] se basa en reutilizar capas intermedias entrenadas previamente, normalmente las que codifican el estado del entorno en un espacio latente significativo. Este enfoque permite compartir conocimiento abstracto entre tareas distintas.

- **Ventajas:** reduce la necesidad de datos, puede facilitar la generalización.
- **Limitaciones:** las representaciones pueden ser demasiado específicas del entorno original.

2.5.5. *Weight initialization + Fine-tuning*

Una de las formas más comunes de *fine-tuning* es reutilizar los pesos de una red ya entrenada [32] como punto de partida para una nueva tarea. Esta estrategia se beneficia de un mejor punto inicial en el espacio de parámetros.

- **Ventajas:** acelera el aprendizaje, reduce el riesgo de mal rendimiento inicial.
- **Limitaciones:** puede llevar al modelo a estancarse si los pesos iniciales son subóptimos para la nueva tarea.

2.5.6. *Multi-task training como base previa*

Entrenar un modelo previamente en múltiples tareas le permite aprender representaciones más generales. Este conocimiento puede luego transferirse a nuevas tareas mediante *fine-tuning*.

- **Ventajas:** genera políticas más robustas y adaptables.
- **Limitaciones:** requiere una fase previa de entrenamiento extensa y diversidad en las tareas.

2.5.7. *Domain adaptation*

La adaptación de dominios [33] se refiere a técnicas para hacer que un modelo entrenado en un entorno funcione bien en otro, pese a diferencias estructurales o dinámicas. Aunque más común en visión por computadora, ha comenzado a aplicarse también en RL.

- **Ventajas:** facilita transferencias entre simulación y el mundo real, o entre morfologías distintas.
- **Limitaciones:** requiere definir mecanismos de alineación entre dominios, lo cual es complejo en RL.

2.5.8. *Comparativa entre supervisado y RL*

A diferencia del aprendizaje supervisado, donde estas técnicas están más estandarizadas, su aplicación en RL presenta desafíos únicos:

- La señal de recompensa es más escasa y ruidosa, lo que complica la transferencia directa.
- Las políticas aprendidas dependen fuertemente de la dinámica del entorno.
- El efecto del *fine-tuning* puede ser inestable sin mecanismos de regularización o congelación adecuados.

Por ello, aunque muchas técnicas son compartidas entre paradigmas, su adaptación a RL requiere especial cuidado en aspectos como la compatibilidad del espacio de observación/acción, la preservación de políticas anteriores y la gestión del *trade-off* entre plasticidad y estabilidad.

2.6 Transferencia entre agentes con diferente morfología

Una de las aplicaciones más desafiantes y prometedoras del aprendizaje por transferencia en entornos de aprendizaje por refuerzo es la transferencia entre agentes con diferente morfología, es decir, con diferentes estructuras físicas, grados de libertad, tamaños o configuraciones articulares. Esta problemática adquiere especial relevancia en entornos simulados como los de MuJoCo, donde coexisten agentes con arquitecturas variadas pero tareas similares, como caminar, correr o mantener el equilibrio.

2.6.1. Casos documentados en la literatura

Numerosos trabajos recientes han explorado esta línea de investigación, especialmente en el contexto de locomoción en entornos simulados. Por ejemplo, el trabajo de [34] introdujo el enfoque de modular policy networks para transferir conocimiento entre agentes como Walker2D o Humanoid, argumentando que las políticas modulares permiten reutilizar módulos de control para partes corporales similares.

Otros enfoques como *Morphology-agnostic transfer learning* [35] han tratado de extraer políticas generalistas capaces de adaptarse rápidamente a diferentes cuerpos. También se han propuesto técnicas de aprendizaje jerárquico, donde una política de alto nivel coordina subpolíticas adaptadas a la morfología específica del agente.

2.6.2. Técnicas usadas para pasar de Walker2D a Humanoid

La transferencia entre estos agentes —Walker2D (2D bípedo) y Humanoid (bípedo 3D complejo)— requiere técnicas que aborden tanto la diferencia en morfología como en espacio de acción y dinámica interna. Algunas de las técnicas más comunes incluyen:

- **Aligning state spaces** [36]: mediante técnicas como redes de mapeo, se intenta encontrar correspondencias entre las observaciones sensoriales de los distintos agentes.
- **Shared latent representations** [37]: en lugar de transferir directamente políticas en el espacio físico, se entrena una representación común abstracta del estado que pueda ser interpretada por múltiples morfologías.
- **Policy distillation adaptativa** [38]: se usa una red entrenada en Walker2D como maestro, y se entrena una política para Humanoid que imite su comportamiento dentro de su propio espacio de acción.
- **Reward shaping específico** [39]: adaptar la función de recompensa para que la tarea tenga objetivos comparables entre diferentes morfologías.

2.6.3. Adaptación de espacios de observación y acción

Uno de los mayores obstáculos técnicos en la transferencia morfológica es la diferencia en la dimensión y semántica de los espacios de observación y acción. Por ejemplo,

Walker2D tiene 17 dimensiones de observación y 6 de acción, mientras que Humanoid puede superar las 300 dimensiones totales. Algunas soluciones incluyen:

- **Autoencoders compartidos [40]**: para traducir observaciones de alto nivel a un espacio comprimido común.
- **Controladores modulares [40]**: redes que separan el procesamiento en partes corporales comunes (piernas, tronco), lo cual facilita la adaptación.
- **Adaptadores entrenables [40]**: capas adicionales que transforman las observaciones del nuevo agente para que encajen en una red preentrenada.

2.6.4. Similitud estructural como ventaja para transferencia

Aunque las morfologías de Walker2D y Humanoid son distintas, existe una similitud estructural parcial que puede aprovecharse:

- Todos los agentes tienen al menos dos “piernas” articuladas, lo que permite compartir subpolíticas de locomoción.
- La tarea de caminar implica patrones de movimiento similares, lo que facilita transferencias de comportamiento.
- En muchos casos, las diferencias radican más en la complejidad del control que en la tarea subyacente.

Estas similitudes justifican el uso de técnicas de transferencia entre estos agentes, y han demostrado reducir significativamente el tiempo de entrenamiento cuando se realiza una transferencia efectiva, especialmente si se combinan con técnicas como *frozen layers* o *curriculum learning*.

2.7 Crítica del estado del arte

El análisis del estado del arte realizado en los apartados anteriores ha permitido construir una base sólida de conocimiento sobre las técnicas actuales de aprendizaje por reforzamiento, transferencia de políticas y *fine-tuning* aplicadas a entornos de simulación física como MuJoCo. Sin embargo, para avanzar de forma significativa en el área, resulta fundamental no solo entender los avances existentes, sino también identificar de manera crítica sus limitaciones, inconsistencias o vacíos metodológicos.

Esta sección tiene como objetivo principal llevar a cabo una evaluación crítica del estado actual de la investigación en torno a estos temas. Se identificarán los principales puntos débiles detectados en la literatura revisada, que sirven como motivación directa para la propuesta de este proyecto. Asimismo, se destacarán aquellos aspectos sólidos o exitosos que el proyecto pretende aprovechar y potenciar.

De esta manera, se establece una transición natural hacia la propuesta experimental, justificando tanto la necesidad como la relevancia de la metodología adoptada en este trabajo.

2.7.1. Puntos débiles identificados

A pesar de los numerosos avances logrados en el campo del aprendizaje por refuerzo y las técnicas de transferencia, el análisis del estado del arte revela varias debilidades y vacíos que motivan la necesidad de nuevos enfoques:

- **Falta de estandarización en técnicas de *fine-tuning* para RL:** Aunque existen múltiples estrategias como *Frozen Layers*, *Curriculum Learning* o *Weight Initialization*, no hay un consenso claro sobre cuándo y cómo aplicarlas de forma sistemática en agentes de RL. La falta de guías estructuradas genera ineficiencias y resultados poco reproducibles.
- **Aplicaciones limitadas en entornos con cambios de morfología:** La mayoría de los trabajos de transferencia y *fine-tuning* se centran en variaciones de tarea o de dominio, pero no abordan suficientemente la transferencia entre agentes físicamente distintos, como Walker2D y HumanoidStandup, donde los espacios de observación y acción difieren significativamente.
- **Impacto energético raramente cuantificado:** Aunque algunos estudios mencionan la eficiencia computacional, la gran mayoría no realiza un análisis cuantitativo del coste energético real derivado de los entrenamientos, un aspecto crucial dada la creciente preocupación por la sostenibilidad en inteligencia artificial.
- **Enfoque limitado en modelos base generalistas:** Se han propuesto modelos base en RL, pero muchos de ellos están fuertemente especializados en tareas concretas, dificultando su reutilización efectiva para agentes o tareas diferentes sin incurrir en grandes pérdidas de rendimiento o necesidad de retraining extensivo.
- **Subestimación de problemas en la transferencia:** Problemas como el *reward mismatch* [ref] o el *policy collapse* [ref] suelen ser nombrados, pero son tratados superficialmente en la literatura, sin estrategias robustas que los mitiguen de manera práctica y generalizable.
- **Escasa evaluación comparativa en condiciones homogéneas:** Existen diferencias metodológicas importantes entre estudios, lo que complica la comparación directa del impacto de las técnicas propuestas y dificulta extraer conclusiones sólidas.

Estas carencias justifican la necesidad de un enfoque experimental controlado, reproducible y enfocado tanto en la eficiencia computacional como energética, que es precisamente la dirección en la que se orienta este proyecto.

2.7.2. Puntos fuertes aprovechables

A pesar de las limitaciones detectadas, el análisis del estado del arte también ha puesto de manifiesto diversas fortalezas y avances recientes que servirán como cimientos sólidos para el diseño del proyecto:

- **Madurez de algoritmos de RL como PPO:** Algoritmos como *Proximal Policy Optimization* han demostrado un rendimiento estable y eficiente en una amplia gama de tareas de locomoción en entornos MuJoCo, lo que proporciona una base confiable para las fases de preentrenamiento y *fine-tuning*.

- **Éxito comprobado de técnicas de fine-tuning en RL:** Aunque heterogéneas, múltiples investigaciones respaldan la viabilidad de estrategias como *Weight Initialization*, *Frozen Layers* o *Curriculum Learning* para acelerar la convergencia del entrenamiento y mejorar la eficiencia.
- **Similitud estructural entre agentes MuJoCo:** Modelos como Walker2D y Humanoid, aunque de distinta complejidad, comparten patrones de locomoción básicos y principios biomecánicos comunes, lo que facilita el uso de modelos base para transferencia entre ellos.
- **Avances en transferencia de representaciones:** La creciente investigación en *representation transfer* y *feature reuse* permite considerar el conocimiento aprendido en un agente como un recurso modular aplicable a otros agentes o tareas, reduciendo el esfuerzo de aprendizaje requerido.
- **Toma de conciencia sobre el impacto energético:** Aunque todavía escasa, la literatura comienza a integrar de manera explícita la medición del consumo energético en el ciclo de evaluación de agentes, reforzando la relevancia de proyectos centrados en eficiencia sostenible.
- **Existencia de benchmarks estandarizados:** Entornos bien establecidos como los de OpenAI Gym con MuJoCo proporcionan una plataforma estandarizada para comparar resultados, lo que facilita la evaluación objetiva del rendimiento de diferentes técnicas de transferencia y fine-tuning.

Estos elementos permiten construir un marco metodológico sólido y moderno, combinando prácticas de RL robustas con un enfoque innovador hacia la reutilización de modelos y la sostenibilidad en el entrenamiento de agentes.

2.7.3. Síntesis final

El análisis del estado del arte pone de manifiesto una clara dualidad: por un lado, existe una base tecnológica y metodológica sólida que demuestra el potencial del aprendizaje por refuerzo y del *fine-tuning* como herramientas para la transferencia eficiente de conocimientos; por otro lado, persisten limitaciones relevantes relacionadas con la eficiencia energética, la falta de generalización entre agentes de distinta morfología y la escasa estandarización de técnicas de *fine-tuning* en entornos complejos como MuJoCo.

En este contexto, el presente proyecto surge como una respuesta directa a estas carencias, buscando no solo aprovechar los avances actuales —como el uso probado de algoritmos robustos como PPO y técnicas de transferencia exitosas—, sino también contribuir a la evolución del campo mediante:

- La creación de un modelo base optimizado para locomoción, que pueda ser reutilizado de manera eficiente entre agentes con diferentes morfologías.
- La implementación sistemática de técnicas de fine-tuning orientadas a la sostenibilidad computacional y energética.
- La evaluación exhaustiva no solo en términos de rendimiento, sino también considerando el impacto de recursos consumidos.

De esta manera, este proyecto no se limita a replicar soluciones existentes, sino que propone un enfoque integral que conecta la mejora del rendimiento con la responsabilidad en el uso de recursos, anticipándose a uno de los grandes desafíos futuros en el desarrollo de inteligencia artificial.

CAPÍTULO 3

Análisis del problema

3.1 Análisis de seguridad

Aunque el presente proyecto se desarrolla exclusivamente en entornos simulados, el concepto de seguridad sigue siendo relevante tanto desde una perspectiva técnica como metodológica. El hecho de que el objetivo final sea una librería reutilizable de modelos preentrenados implica que deben contemplarse medidas que garanticen la fiabilidad, robustez y control de los agentes entrenados durante su ejecución y posterior fine-tuning.

3.1.1. Seguridad del entrenamiento y ejecución en simulación

Durante el entrenamiento de políticas en entornos MuJoCo, especialmente en tareas complejas como Humanoid o HumanoidStandup, pueden producirse comportamientos inestables debido a la exploración aleatoria, la mala configuración de hiperparámetros o una transferencia mal ajustada. Estos comportamientos pueden derivar en simulaciones no controladas, bloqueos del sistema o fallos por saturación de recursos.

Para mitigar este tipo de riesgos, se implementan las siguientes medidas:

- Limitación del número de pasos por episodio y de la duración total del entrenamiento.
- Validación periódica del rendimiento de la política para detectar colapsos tempranos.
- Monitorización activa del uso de recursos mediante herramientas como `nvidia-smi` y `psutil`.

3.1.2. Riesgos en la reutilización del modelo base

Al tratarse de una librería diseñada para ser reutilizada y adaptada a nuevas tareas, existe el riesgo de que una política preentrenada aplicada a un entorno no compatible genere un comportamiento inesperado. Esto puede incluir:

- Aplicación de un modelo en un entorno con diferente espacio de observación o acción.
- *Fine-tuning* de una política con parámetros incompatibles con la arquitectura de destino.

- Olvido catastrófico del conocimiento base si el proceso de adaptación no está controlado.

3.1.3. Perspectiva de seguridad futura en despliegues físicos

Si bien este proyecto se limita al uso de simuladores, el desarrollo de una librería reusable tiene potencial aplicabilidad futura en sistemas robóticos reales. Por tanto, se contempla la posibilidad de que los modelos entrenados puedan ser trasladados, en un contexto de *sim-to-real*, a plataformas físicas. En este caso, asegurar la estabilidad, previsibilidad y trazabilidad del comportamiento del agente sería un requisito indispensable, lo cual refuerza la necesidad de políticas preentrenadas controladas y adaptables mediante mecanismos seguros.

3.2 Análisis energético

El entrenamiento de agentes mediante técnicas de aprendizaje por refuerzo profundo supone un proceso altamente intensivo en recursos computacionales. A diferencia de paradigmas como el aprendizaje supervisado, donde los datos están predefinidos, el RL requiere que los agentes interactúen activamente con el entorno, lo que conlleva millones de pasos de simulación y evaluaciones de políticas. Esta realidad se ve amplificada en entornos físicamente complejos como los ofrecidos por MuJoCo, donde tareas como Humanoid o HumanoidStandup implican una elevada carga de cálculo debido al número de grados de libertad y la dificultad de estabilizar el comportamiento del agente.

3.2.1. Contexto y motivación

Numerosos estudios recientes han comenzado a cuantificar el impacto energético del entrenamiento de modelos de inteligencia artificial, destacando el coste ambiental y económico asociado a estos procesos. Aunque la mayoría de estos análisis se han centrado en modelos de lenguaje a gran escala, los algoritmos de RL no quedan exentos de este problema. En particular, el entrenamiento desde cero de agentes complejos puede prolongarse durante días, implicando un consumo continuo de GPU y CPU que se traduce en un consumo energético elevado.

Dado este contexto, uno de los pilares fundamentales de este proyecto es la reducción del coste energético del entrenamiento mediante la reutilización de conocimiento ya adquirido. En lugar de entrenar cada política desde cero, se propone el desarrollo de un modelo base preentrenado que pueda ser fine-tuneado en nuevos entornos o tareas, logrando una mejora en la eficiencia general del proceso.

3.2.2. Estrategias de eficiencia implementadas

La propuesta incluye varias decisiones diseñadas para minimizar el uso de recursos sin comprometer el rendimiento:

- **Preentrenamiento único y generalista:** Entrenar un único modelo base en Walker2D permite amortizar el coste de entrenamiento sobre múltiples tareas.
- **Fine-tuning eficiente:** Al partir de un modelo ya entrenado, se requieren menos pasos para alcanzar un rendimiento comparable en tareas más complejas.

- **Técnicas de transferencia selectiva:** Métodos como *frozen layers* o *curriculum learning* permiten reducir el número de parámetros actualizados o los pasos necesarios, respectivamente.
- **Monitorización activa:** Se utilizarán herramientas como `psutil`, `nvidia-smi` o `codecarbon` para estimar el consumo energético real durante los experimentos.

3.2.3. Métricas consideradas

Para poder comparar de manera objetiva el impacto energético del fine-tuning frente al entrenamiento desde cero, se recopilarán las siguientes métricas:

- **Tiempo total de entrenamiento** (en horas).
- **Número de pasos de simulación realizados.**
- **Uso medio y máximo de GPU/CPU.**
- **Estimación del consumo energético total** (en Wh).
- **Rendimiento final alcanzado** (recompensa acumulada).

Para llevar a cabo la comparación y evaluación de dichas métricas destacadas en lugar de recurrir a benchmarks preexistentes o datasets de resultados publicados por terceros, este proyecto ha optado por construir y ejecutar un **benchmark propio controlado**, con el fin de evaluar el rendimiento y el coste energético del entrenamiento desde cero frente al *fine-tuning*. Esta decisión se basa en varias razones técnicas y metodológicas:

- **Uniformidad de condiciones de entrenamiento:** al ejecutar todos los experimentos localmente y con la misma configuración de hardware, software y monitorización, se garantiza que las métricas obtenidas son comparables entre sí y no están sesgadas por variaciones externas como versiones de librerías, arquitecturas de red distintas o condiciones de ejecución heterogéneas.
- **Trazabilidad completa del proceso:** al construir el benchmark desde cero, se controla cada aspecto del entrenamiento, desde la inicialización de pesos y la selección del entorno, hasta la forma de aplicar la transferencia o la política de evaluación. Esto permite justificar los resultados de forma precisa y reproducible.
- **Monitorización personalizada de recursos:** los benchmarks externos rara vez incluyen información detallada del uso de CPU, GPU o consumo energético. En este proyecto, el uso de herramientas como `psutil` y `GPUtil` permite obtener métricas finas y consistentes sobre el impacto computacional de cada experimento.
- **Adaptabilidad a las técnicas evaluadas:** dado que se exploran técnicas específicas como *progressive unfreezing*, *frozen layers* o inicialización parcial de pesos, el uso de un benchmark estándar no permitiría capturar los matices y efectos concretos de estas estrategias, o requeriría modificar un entorno ajeno con el consiguiente riesgo de inconsistencias.
- **Evita problemas de compatibilidad:** muchas implementaciones externas están ligadas a versiones específicas de librerías o estructuras de código difíciles de adaptar. Diseñar un benchmark propio con `stable-baselines3` y `Gym` garantiza compatibilidad con el resto del ecosistema utilizado.

En resumen, utilizar un benchmark propio no solo mejora la calidad del análisis cuantitativo y cualitativo, sino que también maximiza la transparencia, trazabilidad y relevancia de las conclusiones obtenidas en el marco del proyecto.

3.2.4. Relevancia en el marco de la sostenibilidad en IA

Este proyecto se enmarca dentro de una corriente creciente que aboga por una inteligencia artificial más sostenible, capaz de desarrollar soluciones eficientes no solo en términos de precisión o rendimiento, sino también en cuanto a su coste energético y huella medioambiental. Al demostrar que un modelo preentrenado y reutilizable puede superar en eficiencia a modelos entrenados desde cero, se busca establecer una referencia práctica y reproducible de buenas prácticas en el entrenamiento de agentes inteligentes.

3.3 Análisis del marco legal y ético

Aunque este proyecto se desarrolla en un entorno puramente simulado y no implica tratamiento de datos personales ni interacción con usuarios reales, resulta relevante analizar su encaje dentro del marco legal y ético actual, especialmente dada la creciente preocupación por el impacto ambiental, la transparencia y la responsabilidad en el desarrollo de tecnologías de inteligencia artificial.

3.3.1. Marco legal

Desde un punto de vista normativo, este proyecto no se ve directamente afectado por regulaciones como el Reglamento General de Protección de Datos [41], ya que no se manejan datos personales ni sensibles. Sin embargo, existen marcos legales emergentes, como el Reglamento Europeo de Inteligencia Artificial [42], que promueven principios como la seguridad, la transparencia, la trazabilidad y la eficiencia de los sistemas de IA. Aunque el proyecto se sitúa en una fase experimental y académica, sus principios de diseño se alinean con dichas directrices:

- **Transparencia:** el entrenamiento, las métricas y la reutilización del modelo base estarán documentados de forma trazable.
- **Eficiencia:** se prioriza la reducción del coste energético como parte del compromiso con un desarrollo tecnológico sostenible.
- **Reutilización responsable:** la creación de una librería modular está diseñada para evitar usos inadecuados.

3.3.2. Consideraciones éticas

El aspecto ético más relevante de este proyecto reside en su enfoque hacia la sostenibilidad computacional. En un momento en que el entrenamiento de modelos de inteligencia artificial representa un consumo energético considerable a nivel global, iniciativas que promuevan la eficiencia en el uso de recursos representan una contribución ética significativa.

Adicionalmente, el desarrollo de una librería reutilizable con modelos base optimizados contribuye a la democratización del acceso a soluciones de RL eficientes, reduciendo

la dependencia de grandes infraestructuras computacionales y permitiendo que investigadores o desarrolladores con recursos limitados puedan reutilizar políticas previamente entrenadas.

También se contempla la posible aplicación futura de este tipo de modelos en contextos reales, como plataformas robóticas o sistemas de asistencia. En ese escenario, aspectos como la trazabilidad del comportamiento del agente, la explicabilidad de las decisiones tomadas y la seguridad operacional adquirirían una relevancia crítica. Por ello, aunque este proyecto no incluye una fase de despliegue físico, el diseño de la librería se concibe desde una lógica de modularidad, control y validación, preparándola para posibles extensiones seguras en el futuro.

3.3.3. Conclusión

El proyecto se alinea con las tendencias actuales hacia una inteligencia artificial más responsable, ética y sostenible, tanto desde el punto de vista del impacto medioambiental como de la reutilización efectiva de modelos. Además, anticipa futuros escenarios de aplicación, incorporando principios de transparencia, trazabilidad y seguridad desde la fase de diseño experimental.

3.4 Análisis de riesgos

El desarrollo de un sistema basado en aprendizaje por refuerzo profundo, especialmente orientado a la reutilización de modelos en distintos entornos y tareas, conlleva una serie de riesgos tanto técnicos como metodológicos. Este apartado identifica los principales factores de riesgo asociados al proyecto y propone estrategias de mitigación para cada uno de ellos, con el objetivo de asegurar la viabilidad técnica, la estabilidad de los resultados y el cumplimiento de los objetivos definidos.

3.4.1. Identificación de riesgos

A continuación se enumeran los riesgos más relevantes:

Tabla 3.1: Tabla de riesgos identificados y estrategias de mitigación

Riesgo	Impacto	Probabilidad
El modelo fine-tuneado no supera al entrenado desde cero	Alto	Medio
Transferencia fallida por incompatibilidad entre espacios de acción/observación	Alto	Bajo
Consumo energético similar o superior al entrenamiento desde cero	Medio	Medio
Estancamiento durante el fine-tuning por mala inicialización	Medio	Medio
Sobrecarga computacional durante el entrenamiento simultáneo de tareas	Medio	Bajo
Dependencia de una sola técnica de transferencia	Medio	Bajo

3.4.2. Enfoque de mitigación general

Como principio general, el desarrollo de la librería y de los experimentos seguirá una lógica modular, controlada y reproducible, lo que permitirá aislar y resolver posibles

incidencias de forma localizada. Además, se empleará una metodología iterativa: los resultados de los primeros entornos de prueba servirán para ajustar las estrategias antes de pasar a entornos más complejos como *Humanoid* o *HumanoidStandup*.

3.5 Análisis de los entornos utilizados

El uso de entornos simulados en tareas de control motor y locomoción requiere no solo eficiencia computacional, sino también una señal de recompensa fiable y una representación física realista. Ambos elementos son fundamentales para asegurar que los comportamientos aprendidos por los agentes sean evaluables, reproducibles y, eventualmente, transferibles al mundo real o a otros entornos de benchmarking.

En este contexto, la validez del presente trabajo depende críticamente de que los entornos utilizados ofrezcan una dinámica coherente con la física del sistema modelado y una función de recompensa alineada con los objetivos reales del agente. Estos criterios permitirán, por un lado, realizar comparaciones justas con benchmarks externos y, por otro, justificar cuantitativamente los resultados obtenidos a lo largo del desarrollo del proyecto.

A continuación, se detallan los aspectos más relevantes en la comparación entre las versiones *Humanoid-v3* cuyo entorno será utilizado por los benchmarks externos y *Humanoid-v5* cuya versión será utilizada para el desarrollo del proyecto, con énfasis en su impacto sobre la señal de recompensa, la complejidad física del entorno y su efecto sobre la estabilidad del entrenamiento.

Una de las diferencias más relevantes entre los entornos *Humanoid-v3* y *Humanoid-v5* reside en la formulación de la función de recompensa y en la fidelidad del modelado físico.

En la versión *Humanoid-v3*, la recompensa se define comúnmente como:

$$\text{reward} = \text{healthy_reward} + \text{forward_reward} - \text{ctrl_cost} - \text{contact_cost} \quad (3.1)$$

No obstante, el término *contact_cost* puede estar distorsionado o incluso omitido debido a valores constantes nulos en el espacio de observación, originados por las limitaciones del motor *MuJoCo 1.5*. Este defecto implica que los agentes pueden obtener recompensas artificialmente elevadas al no recibir penalizaciones realistas por impactos, caídas o deslizamientos. Esta situación compromete la validez del entrenamiento y produce políticas con comportamiento irreal o inestable, afectando entre un 25 %-40 % en la recompensa obtenida debido a que las políticas “malas” eran premiadas injustamente.

Por el contrario, el entorno *Humanoid-v5*, desarrollado sobre *Gymnasium* y *MuJoCo 2.x*, corrige este problema mediante un cálculo más preciso de las fuerzas de contacto. El término *contact_cost* refleja ahora el esfuerzo físico real asociado a los impactos, lo que permite una señal de recompensa más coherente con la física del entorno.

Adicionalmente, v5 ofrece un diccionario *info* más detallado que expone individualmente los componentes de la recompensa, facilitando el análisis del comportamiento del agente y el ajuste fino de hiperparámetros como los coeficientes de penalización por control.

Desde una perspectiva biomecánica, otra mejora crítica es la incorporación de un modelo explícito de tendones entre cadera y rodilla. Esta característica, ausente en v3, introduce restricciones elásticas pasivas que aumentan la estabilidad y naturalidad del movimiento. Aunque esto supone un ligero aumento en el coste computacional por pa-

so de simulación, mejora notablemente la fidelidad física del entorno y la capacidad de generalización de las políticas entrenadas.

En resumen, Humanoid-v5 representa una evolución sustancial respecto a v3, al ofrecer una recompensa más precisa, un entorno físicamente más realista y métricas más útiles para el diagnóstico del aprendizaje. Estas mejoras son esenciales para evitar evaluaciones infladas y para avanzar hacia el entrenamiento de agentes robustos y transferibles, elementos indispensables para validar externamente los resultados del proyecto.

3.6 Identificación y análisis de soluciones posibles

El análisis de riesgos realizado en el apartado anterior ha permitido anticipar diversas situaciones potencialmente problemáticas durante el desarrollo del proyecto, tanto en el entrenamiento de los agentes como en la reutilización del modelo base. A continuación, se identifican distintas soluciones posibles que podrían haberse adoptado para abordar los objetivos del proyecto, así como un análisis de por qué se ha optado por el enfoque finalmente propuesto.

Solución 1: Entrenamiento independiente desde cero para cada entorno

Una de las opciones más directas sería entrenar un modelo específico desde cero para cada tarea o entorno (Walker2D, Humanoid, HumanoidStandup), optimizando cada uno de forma independiente.

- **Ventajas:** permite una adaptación total al entorno; evita problemas de incompatibilidad entre morfologías o espacios de observación.
- **Inconvenientes:** coste computacional y energético extremadamente elevado; cada modelo debe aprender desde cero sin beneficiarse de conocimientos previos; duplicación de esfuerzos; menor escalabilidad.

→ *Esta solución fue descartada como estrategia principal debido a su ineficiencia y falta de sostenibilidad. Solo se ha considerado como línea base comparativa.*

Solución 2: Uso de modelos preentrenados externos o importados

Otra opción habría sido reutilizar modelos base ya entrenados disponibles públicamente, adaptándolos mediante *fine-tuning* a los entornos objetivo.

- **Ventajas:** ahorro de tiempo inicial; posibilidad de partir de modelos bien optimizados.
- **Inconvenientes:** difícil trazabilidad del proceso de entrenamiento original; incompatibilidades estructurales con la arquitectura usada; limitación de control experimental y reproducibilidad académica.

→ *Esta opción se desestimó para mantener el control total sobre el diseño, la arquitectura y los parámetros de los modelos.*

Solución 3: Uso de algoritmos alternativos a PPO (SAC, TD3)

Otra posibilidad habría sido explorar el uso de algoritmos off-policy como *Soft Actor-Critic* o *Twin Delayed DDPG*, que han demostrado rendimiento competitivo en entornos de control continuo.

- **Ventajas:** pueden lograr una exploración más eficiente; SAC incorpora maximización de entropía.
- **Inconvenientes:** mayor complejidad de implementación y ajuste; menos estabilidad en tareas de transferencia directa; mayor sensibilidad a los hiperparámetros.

→ *Se optó por PPO debido a su simplicidad, robustez y amplia adopción en entornos MuJoCo, lo que garantiza mayor estabilidad y comparabilidad.*

Solución 4: Técnicas de meta-learning o RL multitarea generalizado

El uso de meta-aprendizaje o de entrenamientos multitarea generalistas podría haber permitido generar una política adaptable desde el inicio a múltiples entornos.

- **Ventajas:** alto grado de generalización; buena capacidad de adaptación rápida.
- **Inconvenientes:** elevada complejidad computacional y metodológica; fuera del alcance en términos de recursos para un proyecto académico individual.

→ *Se reconocen como líneas de trabajo interesantes, pero se consideran poco viables en este contexto.*

Solución seleccionada: Modelo base entrenado desde cero + fine-tuning específico

La estrategia finalmente adoptada se basa en:

- Entrenar un modelo base en una tarea intermedia (Walker2D).
- Aplicar técnicas de fine-tuning sobre ese modelo para acciones más complejas como Humanoid y generalizarlo para otras tareas similares como HumanoidStandup.
- Comparar rendimiento, eficiencia y consumo con respecto a modelos entrenados desde cero.

Ventajas principales:

- Permite reutilización eficiente del conocimiento.
- Permite una reutilización del conocimiento generalizada para diferentes tareas
- Reducción significativa del coste de entrenamiento.
- Escalabilidad a nuevos entornos y tareas mediante extensión de la librería.
- Control total del proceso experimental y metodológico.

→ *Esta solución representa el mejor equilibrio entre viabilidad, eficiencia y potencial de generalización.*

3.7 Solución propuesta

El proyecto seguirá una metodología experimental basada en los siguientes pasos:

1. Preentrenamiento base:

- Entrenamiento del modelo PPO en Walker2D hasta alcanzar un nivel de rendimiento alto y estable. Este entrenamiento no se considerará en la comparación de recursos de los modelos fine-tuneados ya que este modelo sirve como base y es supuesto como un modelo externo el cual se utiliza en nuestro proyecto. Por lo que lo único que se deberá tratar es de conseguir un rendimiento óptimo sin tener en cuenta el uso de recursos. Esto es debido a que cuando se hace uso de un modelo externo, este no se tiene en cuenta a la hora de comparar los recursos usados.

2. Diseño de estrategias de fine-tuning:

- Definición de varias estrategias de transferencia:
 - a) Congelación parcial de capas.
 - b) Inicialización de pesos + reentrenamiento.
 - c) *Curriculum Learning*

3. Aplicación de fine-tuning:

- Adaptación del modelo base a Humanoid y HumanoidStandup aplicando las técnicas de transferencia de forma controlada.

4. Entrenamiento baseline:

- Entrenamiento de Humanoid y HumanoidStandup **desde cero** con PPO, como referencia comparativa.

5. Evaluación experimental:

- Análisis de métricas de rendimiento.
- Análisis del consumo computacional y energético aproximado.

6. Análisis crítico y conclusiones:

- Comparación de resultados.
- Discusión sobre ventajas, limitaciones y posibles mejoras.

3.7.1. Herramientas y entornos

- **Framework RL:** *Stable-Baselines3*.
- **Algoritmo principal:** *PPO*.
- **Entorno de simulación:** *OpenAI Gym + MuJoCo*.
- **Lenguaje de programación:** *Python 3.12.7*.
- **Infraestructura:** GPU de media-alta gama, CPU multihilo, monitorización de consumo con herramientas como *nvidia-smi* y *psutil*.

3.7.2. Justificación de las decisiones técnicas

- **Uso de PPO:** su estabilidad y facilidad de ajuste hiperparámetros lo convierten en una elección ideal para tareas de locomoción continua.
- **Elección de Walker2D como base:** suficientemente complejo para ser relevante, pero suficientemente simple para permitir transferencias controladas.
- **Aplicación de múltiples técnicas de fine-tuning:** permite evaluar cuál técnica o combinación ofrece el mejor balance entre rapidez, coste y rendimiento.
- **Énfasis en eficiencia energética:** la sostenibilidad se convierte en un criterio explícito de evaluación, alineándose con tendencias globales en IA responsable.

CAPÍTULO 4

Diseño de la solución

4.1 Arquitectura del proyecto

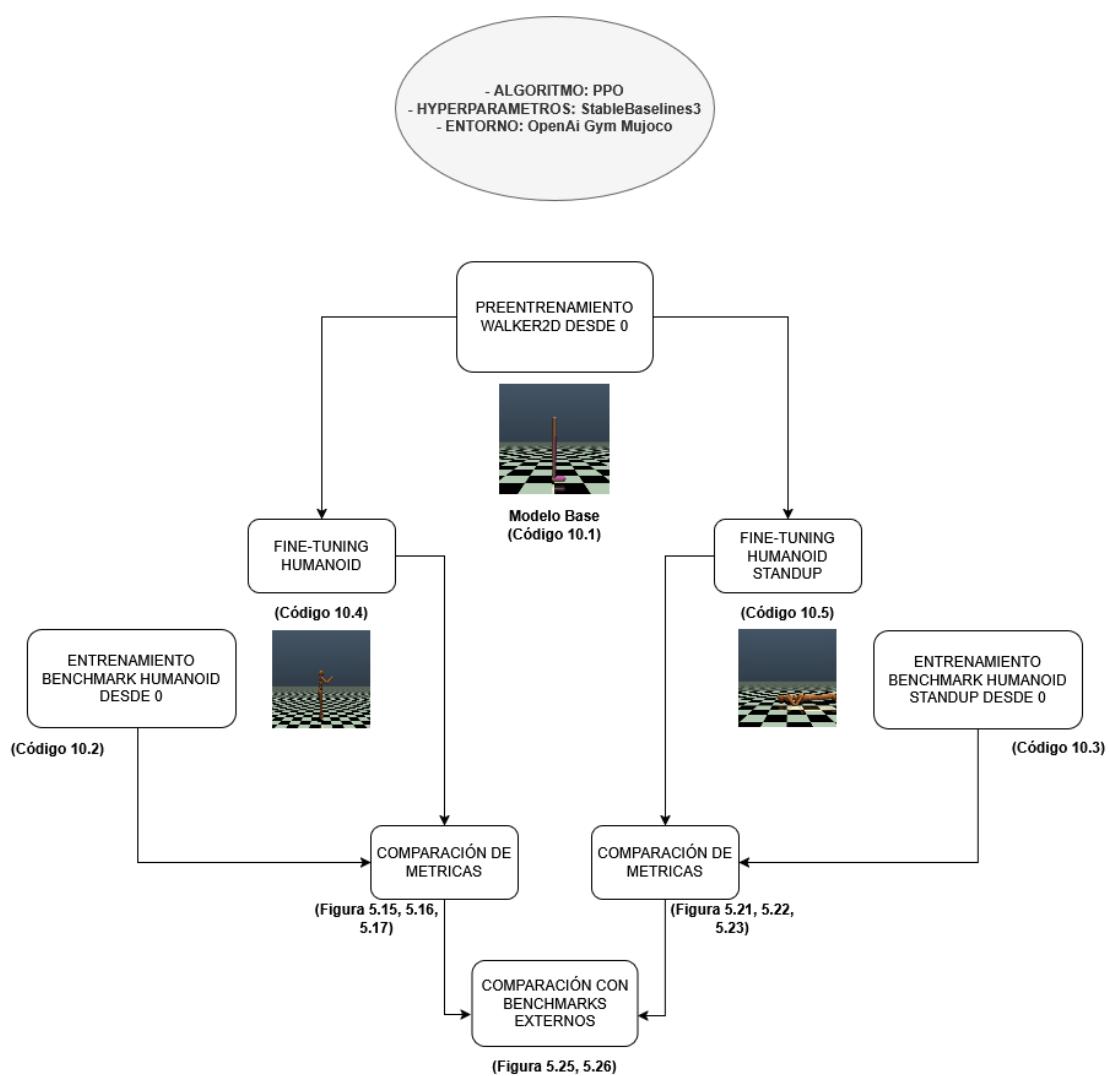


Figura 4.1: Arquitectura del proyecto

4.2 Recompensas uniformes

Uno de los retos más habituales en investigación en aprendizaje por refuerzo es el elevado coste computacional y energético de entrenar agentes de manera repetida para obtener estadísticas robustas. Aunque idealmente cada configuración debería evaluarse sobre múltiples semillas aleatorias para calcular la media y la desviación estándar, este enfoque no siempre es viable en proyectos con recursos limitados.

En este trabajo, se ha optado por utilizar un único entrenamiento representativo por configuración —o, en algunos casos, la media de tres ejecuciones— como aproximación práctica y eficiente. Esta decisión está respaldada por el hecho de que los entornos de MuJoCo, como Walker2d, Humanoid o HumanoidStandup, tienden a presentar trayectorias de entrenamiento con desviaciones estándar relativamente acotadas entre ejecuciones con la misma configuración, especialmente cuando se utilizan algoritmos estables como PPO.

En la literatura se reportan desviaciones típicas del orden de un 5–10 % en la recompensa final tras la convergencia, lo cual sitúa nuestra metodología dentro de un margen aceptable para extraer conclusiones cualitativas. Además, al centrar el análisis en tendencias relativas, el impacto de la varianza aleatoria se reduce aún más, permitiendo valorar de forma fiable la eficiencia y la transferencia sin necesidad de grandes muestras.

En resumen, la estrategia adoptada busca equilibrar rigor experimental con viabilidad práctica, garantizando una base suficiente para justificar las conclusiones del trabajo sin comprometer su reproducibilidad o coherencia metodológica.

4.3 Diseño detallado

4.3.1. Walker2d

El modelo entrenado en el entorno Walker2d-v5 constituye la base reutilizable para posteriores procesos de *fine-tuning* en entornos más complejos. Con el objetivo de mejorar su capacidad de generalización y robustez ante dinámicas físicas variables, se ha incorporado una técnica de **randomización de dominio** (*domain randomization*) durante el entrenamiento.

Entorno y agente

El entorno Walker2d-v5, perteneciente al framework *OpenAI Gym MuJoCo*, simula un agente bípedo en dos dimensiones cuya tarea consiste en desplazarse hacia adelante sin caer. Sus características son:

- **22 dimensiones** de observación.
- **6 acciones continuas**, correspondientes a motores articulares.
- Dinámica física más simple que la del entorno Humanoid, pero suficiente para aprender locomoción significativa.

Se considera un entorno intermedio, lo que lo hace adecuado como punto de partida para entrenar un modelo base reutilizable.

Randomización de dominio

Durante el entrenamiento, se ha aplicado un *wrapper* personalizado al entorno que modifica aleatoriamente ciertos parámetros físicos al inicio de cada episodio. Concretamente, se han introducido las siguientes variaciones:

- **Gravedad (eje z):** se selecciona aleatoriamente en el rango

$$g_z \in [-10, -9] \text{ m/s}^2$$

- **Fricción de contacto:** se modifica la fricción del primer cuerpo geométrico en cada reset:

$$\mu \in [0,5, 1,5]$$

- **Escalado de masas:** todas las masas corporales se escalan por un factor aleatorio:

$$m' = m \cdot \alpha, \text{ donde } \alpha \in [0,8, 1,2]$$

Este tipo de aleatorización permite que el modelo aprenda comportamientos locomotores más estables y menos dependientes de una única configuración del entorno, facilitando así la transferencia posterior a otros agentes.

Algoritmo de entrenamiento

El modelo ha sido entrenado utilizando el algoritmo **Proximal Policy Optimization** con la política M1pPolicy, ambas proporcionadas por la librería stable-baselines3. PPO es un algoritmo on-policy adecuado para entornos continuos, y ha demostrado buena estabilidad en tareas de locomoción.

La política neuronal emplea una arquitectura multicapa básica, sin alteraciones respecto a la configuración por defecto, salvo la incorporación del entorno modificado.

Configuración del entrenamiento

El modelo se ha entrenado desde cero utilizando distintas configuraciones de pasos totales:

$$\{2 \times 10^6, 3 \times 10^6 \text{ timesteps}\}$$

Durante el entrenamiento:

- Se aplica randomización de dominio en cada episodio.
- Se evalúa el agente cada 5000 timesteps en 5 episodios.
- No se congela ninguna capa: el modelo es entrenado íntegramente desde cero.

El modelo final generado en cada caso se utiliza posteriormente como punto de partida para el proceso de *fine-tuning* en entornos de mayor complejidad.

4.3.2. Humanoid benchmark desde 0

El entrenamiento del modelo benchmark desde cero en el entorno Humanoid-v5 tiene como finalidad servir de línea base comparativa frente a los modelos obtenidos mediante *fine-tuning*. Este enfoque representa el caso tradicional en el que un agente aprende su política desde una inicialización aleatoria, sin aprovechamiento de conocimiento previo.

Entorno y agente

El entorno Humanoid-v5 forma parte de los entornos de locomoción avanzados de *MuJoCo*. Simula un agente humanoide con gran complejidad estructural y alta dimensionalidad en su espacio de observación y acción. Sus características son:

- **376 dimensiones** de observación, incluyendo posiciones, velocidades, fuerzas, y orientación de segmentos corporales.
- **17 acciones continuas**, representando señales de control para las articulaciones del cuerpo humanoide.
- Un entorno físico sensible a errores, donde el agente puede caer fácilmente en las primeras fases de entrenamiento.

Este entorno representa un caso exigente desde el punto de vista computacional, y por tanto, un candidato ideal para evaluar el coste de aprender desde cero frente a alternativas de reutilización de modelos.

Algoritmo de entrenamiento

El agente ha sido entrenado utilizando el algoritmo **Proximal Policy Optimization** con la política `MlpPolicy`, disponible en la librería `stable-baselines3`. PPO es un algoritmo *on-policy* ampliamente utilizado en tareas de control continuo por su buen equilibrio entre rendimiento y estabilidad.

El modelo implementa una red neuronal multicapa sin modificaciones respecto a la configuración por defecto del framework, lo que facilita la comparación con otros experimentos bajo condiciones homogéneas.

Configuración del entrenamiento

Para permitir un análisis detallado de la evolución del rendimiento y del coste computacional, se han definido cuatro configuraciones distintas de número total de pasos de entrenamiento:

$$\{2 \times 10^6, 3 \times 10^6, 5 \times 10^6, 10 \times 10^6 \text{ timesteps}\}$$

Durante el proceso de entrenamiento se han mantenido las siguientes condiciones:

- Inicialización aleatoria de todos los pesos del modelo.
- Evaluación del agente cada 5000 timesteps.
- Hiperparámetros base de la librería de `stable-baselines3`.

- Entrenamiento en un único entorno de simulación, sin paralelización.

El objetivo es establecer una comparación directa entre el rendimiento alcanzado y los recursos consumidos por este enfoque y los modelos derivados del modelo base de *Walker2D* adaptado por *fine-tuning*.

4.3.3. HumanoidStandup benchmark desde 0

Este experimento tiene como objetivo establecer una línea base para comparar el rendimiento y la eficiencia del enfoque tradicional de entrenamiento desde cero en el entorno *HumanoidStandup-v5*, sin ningún tipo de transferencia o reutilización de conocimiento previo.

Entorno y agente

El entorno *HumanoidStandup-v5* pertenece al framework *OpenAI Gym MuJoCo* y representa un reto distinto dentro de las tareas de locomoción: el agente debe aprender a levantarse del suelo y mantenerse erguido. Sus principales características son:

- **376 dimensiones** en el vector de observación, incluyendo información postural, velocidades angulares y contactos con el suelo.
- **17 dimensiones** de acción continua, correspondientes a los motores articulares del cuerpo humanoide.
- La tarea no es caminar, sino alcanzar y mantener una postura vertical a partir de una posición de reposo inicial (tumbado).

La complejidad de esta tarea radica en que no existe movimiento horizontal como refuerzo inmediato, lo que complica la exploración y el aprendizaje inicial.

Algoritmo de entrenamiento

Para entrenar el agente desde cero se ha utilizado el algoritmo **Proximal Policy Optimization** con arquitectura *MlpPolicy*, a través de la implementación en la librería *stable-baselines3*. PPO es una técnica *on-policy* robusta y eficaz para entornos de control continuo como el presente.

La arquitectura de red neuronal se ha mantenido sin modificaciones respecto a la configuración por defecto, para permitir una comparación homogénea con otros enfoques experimentales del proyecto.

Configuración del entrenamiento

Se han realizado múltiples entrenamientos con distintos presupuestos de pasos totales, a fin de evaluar cómo afecta la duración del entrenamiento al rendimiento final del agente:

$$\{2 \times 10^6, 3 \times 10^6, 5 \times 10^6, 10 \times 10^6 \text{ timesteps}\}$$

En todas las configuraciones se ha seguido el siguiente esquema experimental:

- Inicialización aleatoria completa de los parámetros del modelo.
- Evaluación periódica cada 5000 timesteps durante 5 episodios consecutivos.
- Misma política, entorno, hiperparámetros base y procedimiento de monitorización que en el resto de entrenamientos del proyecto.

Este modelo sirve como punto de referencia para determinar si el uso de un modelo base y técnicas de *fine-tuning* aportan beneficios significativos en términos de eficiencia, rendimiento y costo computacional.

4.3.4. Fine-Tuning en Humanoid-v5

El objetivo de esta fase es adaptar el modelo base preentrenado en Walker2d-v5 a un entorno más complejo como Humanoid-v5, evaluando si la reutilización del conocimiento previo permite obtener un rendimiento competitivo frente al entrenamiento desde cero, reduciendo al mismo tiempo el costo computacional y energético.

Entorno y agente destino

El entorno Humanoid-v5, basado en *MuJoCo*, representa una de las tareas más exigentes del conjunto *Gymnasium*. Requiere que un agente bípedo aprenda a caminar de forma estable, partiendo de una arquitectura corporal con múltiples grados de libertad. Entre sus características principales destacan:

- **376 dimensiones** de observación.
- **17 acciones continuas** para el control articular.
- Alta inestabilidad física, especialmente durante las primeras etapas del entrenamiento.

Esta complejidad lo convierte en un entorno ideal para poner a prueba la capacidad del modelo base para transferir representaciones útiles en tareas de locomoción.

Proceso de transferencia

Se ha aplicado la técnica **weight initialization + fine-tuning**, consistente en reutilizar parcialmente los pesos del modelo base en un nuevo modelo con arquitectura compatible. El proceso es el siguiente:

1. Se carga un modelo PPO previamente entrenado en Walker2d-v5, con pesos aprendidos.
2. Se inicializa un nuevo agente PPO en Humanoid-v5, con una arquitectura explícitamente definida.
3. Se copian únicamente los pesos cuyos nombres y dimensiones coinciden entre ambos modelos.
4. Se actualiza el estado del modelo destino con los parámetros transferidos, manteniendo la estructura restante intacta.

Esta estrategia permite acelerar el aprendizaje inicial aprovechando representaciones ya optimizadas, sin imponer restricciones rígidas en la arquitectura del modelo destino.

Técnicas aplicadas durante el fine-tuning

Para facilitar una adaptación estable y progresiva del modelo al nuevo entorno, se han aplicado dos técnicas complementarias de transferencia estructurada:

- **Congelación de capas (*Frozen Layers*)**: tras la transferencia de pesos, se congela el extractor de características (`mlp_extractor`) durante una fase inicial del entrenamiento. Esto preserva las representaciones generales aprendidas en el entorno original.
- **Descongelado progresivo (*Progressive Unfreezing*)**: una vez superado un umbral del total de pasos definidos, un callback personalizado desbloquea las capas congeladas, permitiendo su ajuste. Este umbral depende del presupuesto total de entrenamiento:
 - 2×10^6 timesteps → descongelado al 20 %.
 - 3×10^6 timesteps → descongelado al 20 %.
 - 5×10^6 timesteps → descongelado al 30 %.
 - 10×10^6 timesteps → descongelado al 40 %.

No se aplica aprendizaje por destilación explícita, aunque el modelo base actúa como punto de partida estructural en la política de destino.

Configuración del entrenamiento

El entrenamiento se ha realizado con los siguientes presupuestos de pasos:

$$\{2 \times 10^6, 3 \times 10^6, 5 \times 10^6, 10 \times 10^6\}$$

Durante cada experimento:

- Se emplea una política `ActorCriticPolicy` con arquitectura definida manualmente: tres capas ocultas de 64 unidades y función de activación `Tanh`.
- Se utilizan hiperparámetros personalizados para PPO (no los valores por defecto).
- Se evalúa al agente cada 50 000 pasos en 5 episodios.
- Se monitoriza el uso de CPU y GPU para estimar el consumo energético.

El modelo resultante se guarda tras cada experimento y se compara tanto con modelos entrenados desde cero como con otras variantes de *fine-tuning*. Las métricas clave incluyen recompensa final, duración del entrenamiento, eficiencia energética y estabilidad de la política.

4.3.5. Fine-Tuning en HumanoidStandup-v5

Este experimento tiene como objetivo demostrar que las técnicas de *fine-tuning* utilizadas en este proyecto —basadas en la inicialización de pesos y el aprendizaje progresivo— no solo mejoran el rendimiento y la eficiencia frente al entrenamiento desde cero, sino que además son **transferibles a tareas distintas con morfología similar**.

Para ello, se ha aplicado el modelo base entrenado en `Walker2d-v5` sobre el entorno `HumanoidStandup-v5`, una tarea que difiere significativamente de caminar, ya que el agente debe aprender a incorporarse desde el suelo y mantenerse erguido. Este escenario permite evaluar el **grado de generalización** del conocimiento adquirido en una tarea a otra que requiere una dinámica motora completamente distinta.

Entorno y agente destino

El entorno HumanoidStandup-v5, también basado en *MuJoCo*, representa una tarea avanzada de locomoción vertical con distinta secuencia motora respecto al caminar. Sus características principales son:

- **376 dimensiones** de observación, idénticas a las de Humanoid-v5.
- **17 acciones continuas**, que controlan las mismas articulaciones que en el agente caminante.
- La tarea requiere que el agente pase de una posición tumbada a una postura vertical y mantenga el equilibrio.

Esta similitud estructural con Humanoid-v5 y con el modelo base Walker2d-v5 permite aplicar técnicas de transferencia que, aunque entrenadas para caminar, podrían aportar beneficios en tareas que comparten control motor complejo.

Proceso de transferencia

Se ha utilizado la misma estrategia general aplicada en otros escenarios del proyecto: **weight initialization + fine-tuning**. Esta técnica consiste en aprovechar parcialmente los pesos del modelo base en un modelo nuevo, de forma selectiva y compatible:

1. Se carga el modelo base entrenado en Walker2d-v5.
2. Se inicializa un nuevo agente PPO sobre HumanoidStandup-v5.
3. Se recorren los parámetros del nuevo modelo y se copian aquellos compatibles en nombre y forma desde el modelo base.
4. Se actualiza el estado del modelo destino, manteniendo las estructuras específicas del nuevo entorno.

Este enfoque no impone restricciones arquitectónicas y permite acelerar el aprendizaje incluso en tareas morfológicas y dinámicamente distintas.

Técnicas aplicadas durante el fine-tuning

Para asegurar una adaptación progresiva y evitar la pérdida de conocimiento del modelo base, se han aplicado dos técnicas combinadas de transferencia estructurada:

- **Congelación de capas (*Frozen Layers*)**: se congelan las capas del extractor de características (`mlp_extractor`) durante la primera fase del entrenamiento, preservando las representaciones generales aprendidas en la tarea original.
- **Descongelado progresivo (*Progressive Unfreezing*)**: una vez alcanzado un determinado porcentaje de los *timesteps* totales, se habilita el ajuste completo de los pesos del modelo. Esta técnica, inspirada en el *curriculum learning estructural*, permite una transición controlada desde el conocimiento previo hacia la adaptación específica.

Esta combinación permite no solo mejorar el rendimiento, sino también facilitar la generalización hacia tareas que comparten estructura pero no necesariamente objetivo.

Configuración del entrenamiento

El *fine-tuning* se ha llevado a cabo bajo los siguientes presupuestos de entrenamiento:

$$\{2 \times 10^6, 3 \times 10^6, 5 \times 10^6, 10 \times 10^6 \text{ timesteps}\}$$

En todos los casos se ha mantenido una configuración coherente con el resto del proyecto:

- Política ActorCriticPolicy con arquitectura personalizada (3 capas ocultas de 64 unidades y activación Tanh).
- Evaluación del rendimiento cada 5000 pasos en 5 episodios.
- Medición de uso de recursos y estimación del consumo energético.

Este experimento permite validar que las técnicas aplicadas no solo ofrecen ventajas en tareas similares, sino que pueden adaptarse a situaciones distintas, confirmando su potencial de reutilización y eficiencia en el contexto del aprendizaje por refuerzo generalizado.

4.4 Justificación de decisiones técnicas y de implementación

Este apartado detalla las decisiones técnicas adoptadas a lo largo del desarrollo del proyecto, así como la justificación de cada una en base a criterios de estabilidad, eficiencia y coherencia experimental. El objetivo es asegurar la comparabilidad entre métodos, maximizar la reproducibilidad y mantener un enfoque práctico y escalable.

4.4.1. Frecuencia de evaluación durante el entrenamiento

Se ha implementado una evaluación periódica del rendimiento de los agentes durante el entrenamiento, ajustando la frecuencia según el tipo de estrategia empleada: entrenamiento desde cero o *fine-tuning*.

Entrenamiento desde cero: Evaluación cada 5000 *timesteps*

Para los modelos entrenados desde cero, se ha optado por evaluar el rendimiento del agente cada 5000 *timesteps*. Esta frecuencia responde a las siguientes motivaciones:

- Permite capturar con alta resolución la evolución de la política, especialmente en las primeras fases del entrenamiento donde los cambios son más abruptos.
- Facilita el análisis detallado del comportamiento del agente, incluyendo la tasa de mejora y la identificación de posibles estancamientos.
- Genera suficientes puntos intermedios para asociar la evolución del rendimiento con métricas temporales como el consumo energético, sin introducir una sobrecarga computacional significativa.

Entrenamiento mediante *fine-tuning*: Evaluación cada 50000 *timesteps*

En el caso del *fine-tuning*, la evaluación se ha espaciado a cada 50,000 *timesteps*. Esta decisión se justifica por las siguientes razones:

- El modelo parte de una política preentrenada, por lo que se espera una mayor estabilidad inicial y menor variabilidad episodio a episodio.

- Dado que estos entrenamientos suelen ser más cortos en duración total, una frecuencia menor evita una sobresaturación de datos sin perder trazabilidad de la evolución global.
- La evaluación menos frecuente reduce el tiempo de cómputo dedicado a validación y permite concentrar recursos en el ajuste de la política.
- Aun con menor frecuencia, se mantiene la posibilidad de calcular métricas clave como recompensa media, convergencia, y consumo energético acumulado por bloques.

En ambos casos, la frecuencia de evaluación ha sido cuidadosamente seleccionada en función del tipo de experimento y la duración prevista del entrenamiento, garantizando la comparabilidad entre métodos y la eficiencia del proceso de monitorización.

4.4.2. Estrategia de transferencia: *Weight Initialization*

La estrategia de transferencia de conocimiento varía en función del tipo de entrenamiento aplicado. En los modelos entrenados desde cero no se aplica transferencia explícita, mientras que en los modelos con *fine-tuning* se recurre a una inicialización selectiva de pesos. A continuación se detalla el enfoque seguido en cada caso.

Entrenamiento desde cero

En los modelos entrenados completamente desde cero, no se utiliza ninguna estrategia de transferencia. El modelo se inicializa con pesos aleatorios siguiendo el esquema predeterminado de `stable-baselines3`.

Esta decisión está justificada por los siguientes motivos:

- Permite establecer una línea base clara y objetiva para comparar el impacto de las técnicas de *fine-tuning*.
- Evita cualquier influencia previa que pueda sesgar el rendimiento inicial del agente.
- Es coherente con el objetivo de evaluar el coste energético y temporal de entrenar una política desde cero.

Entrenamiento mediante *fine-tuning*

Para los modelos ajustados mediante *fine-tuning*, se ha aplicado la técnica de inicialización selectiva de pesos (*Weight Initialization*) desde un modelo base preentrenado hacia un modelo destino más complejo.

El procedimiento aplicado es el siguiente:

1. Se carga el modelo base preentrenado y se extrae su `state_dict()`.
2. Se recorre el `state_dict()` del modelo destino.
3. Se copian los pesos de todas aquellas capas cuyos nombres y dimensiones coincidan exactamente.

Esta técnica presenta múltiples ventajas específicas en el contexto del proyecto:

- Permite reutilizar conocimiento estructural aprendido en tareas motoras similares.
- Facilita una convergencia más rápida al evitar la exploración aleatoria inicial.
- Conserva la compatibilidad entre agentes con diferente morfología, gracias a su criterio de coincidencia estricta.
- Es una solución ligera, reproducible y fácil de integrar dentro del flujo de entrenamiento en stable-baselines3.

En resumen, la estrategia de *Weight Initialization* ha sido aplicada exclusivamente en los experimentos de *fine-tuning*, como mecanismo clave para evaluar el valor de la transferencia de políticas previamente entrenadas, en contraste con el aprendizaje desde cero.

4.4.3. Elección de la política: MlpPolicy y ActorCriticPolicy

Los entrenamientos desarrollados en este proyecto hacen uso de políticas basadas en redes neuronales completamente conectadas. Sin embargo, la clase específica empleada para representar dicha arquitectura varía entre los modelos entrenados desde cero y los ajustados mediante *fine-tuning*, como se detalla a continuación.

Entrenamiento desde cero: MlpPolicy

En los modelos entrenados desde cero se ha utilizado la política MlpPolicy, que es la implementación por defecto para entornos con observaciones vectoriales en la librería stable-baselines3. Esta elección responde a las siguientes motivaciones:

- Proporciona una arquitectura estándar y validada para tareas de locomoción simulada como Walker2D.
- Facilita la reproducibilidad al evitar configuraciones personalizadas.
- Permite evaluar el rendimiento del entrenamiento desde cero de forma limpia, sin factores externos asociados a la arquitectura.

Entrenamiento mediante *fine-tuning*: ActorCriticPolicy

En los experimentos de *fine-tuning*, se ha utilizado la clase ActorCriticPolicy con una arquitectura definida explícitamente mediante el parámetro policy_kwarg. En concreto, se especifica una red simétrica con tres capas ocultas de 64 unidades para la política y el valor, junto con la función de activación Tanh.

La elección de esta clase se justifica por las siguientes razones:

- ActorCriticPolicy ofrece una mayor flexibilidad para personalizar arquitecturas internas sin depender de las versiones preconfiguradas como MlpPolicy.
- Permite asegurar la compatibilidad estructural con el modelo base preentrenado en el que se basa la transferencia de pesos.
- Resulta útil para el control granular del comportamiento de la red en técnicas de congelación y descongelado progresivo de capas.

En ambos enfoques, se mantiene una estructura de red sencilla y transparente, lo que permite enfocar el análisis en el impacto de la estrategia de entrenamiento sin introducir sesgos por diferencias arquitectónicas complejas.

4.4.4. Uso de hiperparámetros en PPO

El algoritmo de entrenamiento empleado en todos los experimentos ha sido Proximal Policy Optimization, una técnica ampliamente validada para entornos de control continuo. Sin embargo, la configuración de sus hiperparámetros varía entre los modelos entrenados desde cero y los ajustados mediante *fine-tuning*.

Entrenamiento desde cero: Hiperparámetros por defecto

En los modelos entrenados desde cero se han mantenido los hiperparámetros por defecto proporcionados por la implementación de PPO en la librería stable-baselines3. Esta decisión se basa en los siguientes criterios:

- Los valores por defecto están optimizados para una amplia variedad de tareas en entornos MuJoCo y ofrecen un rendimiento base competitivo.
- El uso de configuraciones estándar favorece la reproducibilidad y evita introducir variables adicionales que compliquen el análisis comparativo.
- Permite centrar el estudio en los efectos del entrenamiento desde cero sin interferences derivadas de la búsqueda de hiperparámetros óptimos.

Entrenamiento mediante *fine-tuning*: Hiperparámetros personalizados

En los modelos entrenados mediante *fine-tuning*, se han especificado manualmente algunos hiperparámetros del algoritmo PPO, incluyendo:

- `learning_rate = 3e-4`
- `n_steps = 2048`
- `batch_size = 64`
- `n_epochs = 10`
- `gamma = 0.99`
- `gae_lambda = 0.95`
- `clip_range = 0.2`
- `ent_coef = 0.0`

La configuración explícita de estos parámetros tiene las siguientes justificaciones:

- Garantiza coherencia estructural con el modelo base del que se transfieren los pesos, lo cual puede influir en la estabilidad del entrenamiento.
- Permite mantener un control más preciso sobre el comportamiento del agente durante el proceso de adaptación.
- Mejora la compatibilidad con el sistema de congelación y descongelado progresivo, que se beneficia de configuraciones bien calibradas para facilitar la transferencia.

En resumen, mientras que el entrenamiento desde cero prioriza simplicidad y reproducibilidad usando valores por defecto, el *fine-tuning* requiere un ajuste más controlado de los hiperparámetros para maximizar la eficacia del proceso de transferencia y asegurar una evolución estable del modelo.

4.4.5. Técnicas de congelación y descongelado progresivo

Una de las diferencias clave entre los enfoques de entrenamiento desarrollados en este proyecto es la aplicación de técnicas de congelación estructural de la red neuronal. Estas técnicas solo se emplean en el contexto del *fine-tuning*, con el objetivo de preservar el conocimiento del modelo base durante las primeras fases del aprendizaje en el nuevo entorno.

Entrenamiento desde cero

En los entrenamientos realizados desde cero, no se aplica ningún tipo de congelación de capas. Todos los parámetros de la red neuronal son entrenables desde el inicio. Esta decisión es coherente con los siguientes principios:

- El modelo parte de una política completamente aleatoria, por lo que no existe conocimiento previo que deba preservarse.
- La activación completa de todos los parámetros maximiza la capacidad de exploración y aprendizaje desde cero.
- Se busca establecer una línea base sin restricciones estructurales que interfieran con la dinámica natural del aprendizaje.

Entrenamiento mediante *fine-tuning*

En los modelos ajustados mediante *fine-tuning*, se ha aplicado una estrategia híbrida compuesta por dos fases:

- **Congelación inicial de capas:** al inicio del entrenamiento, los parámetros del extractor de características (`mlp_extractor`) se congelan, impidiendo su actualización. Esto permite mantener las representaciones generales aprendidas en el modelo base y reduce el riesgo de *catastrophic forgetting*.
- **Descongelado progresivo:** tras superar un umbral definido del total de pasos de entrenamiento, se desbloquean todas las capas del modelo. Esta transición se implementa mediante un *callback* personalizado y está parametrizada por un porcentaje del total de *timesteps*.

Este enfoque actúa como una forma de *curriculum learning* estructural, donde se aumenta gradualmente la plasticidad del modelo en función del avance del entrenamiento. Las motivaciones principales son:

- Permitir que el modelo aproveche inicialmente una base estable de conocimiento antes de comenzar a adaptarse al nuevo entorno.
- Evitar interferencias destructivas en capas profundas durante la fase temprana del aprendizaje.
- Facilitar una adaptación progresiva y estable, simulando un aprendizaje escalonado.

Esta técnica híbrida, cuidadosamente adaptada al tiempo de entrenamiento disponible, permite mantener un equilibrio entre estabilidad inicial y flexibilidad final, optimizando así la transferencia de políticas en tareas complejas.

CAPÍTULO 5

Desarrollo de la solución propuesta

Este capítulo describe de forma detallada el proceso de implementación de la solución propuesta basada en el diseño, desde la preparación del entorno hasta la ejecución de los experimentos. Se cubren los aspectos prácticos del desarrollo, la estructura del código, la creación del modelo base y los diferentes entrenamientos realizados, tanto desde cero como mediante técnicas de *fine-tuning*.

El objetivo de esta sección es mostrar cómo se ha llevado a la práctica el diseño planteado en capítulos anteriores, poniendo especial énfasis en:

- La modularización del sistema y la gestión del código experimental.
- El entrenamiento controlado del modelo base en el entorno Walker2d-v5 con técnicas de *domain randomization*.
- La ejecución de entrenamientos benchmark desde cero en entornos complejos como Humanoid-v5 y HumanoidStandup-v5.
- La implementación del *fine-tuning* desde el modelo base a entornos con mayor dificultad, empleando estrategias como congelación de capas y descongelado progresivo.
- La monitorización del uso de recursos computacionales y la estimación del consumo energético durante cada experimento.

Además, se detallarán los problemas encontrados durante el desarrollo, incluyendo intentos fallidos y decisiones técnicas clave que permitieron alcanzar una solución funcional, robusta y eficiente. Este recorrido permitirá entender no solo lo que se ha hecho, sino por qué y cómo se ha hecho, fundamentando así los resultados presentados en capítulos posteriores.

5.1 Preparación del entorno de trabajo

Con el objetivo de garantizar la reproducibilidad, estabilidad y trazabilidad de todos los experimentos realizados, se ha configurado un entorno de trabajo controlado, tanto a nivel de software como de hardware. A continuación, se describen las herramientas utilizadas y las decisiones tomadas en la preparación del entorno.

5.1.1. Lenguaje y entorno de programación

El proyecto se ha desarrollado íntegramente en **Python 3.12.7**, dado que es el lenguaje de referencia en investigación actual en Aprendizaje por Refuerzo y cuenta con una amplia compatibilidad con librerías como Gym, Stable-Baselines3 y herramientas de monitorización.

El entorno virtual utilizado ha sido creado mediante la plataforma Anaconda, haciendo uso de la herramienta conda para la gestión de entornos. Esta elección ha permitido:

- Crear un entorno aislado y controlado para el proyecto, evitando conflictos de versiones entre librerías.
- Gestionar de forma sencilla las dependencias específicas requeridas para simulación, entrenamiento y análisis.
- Garantizar la portabilidad y reproducibilidad del entorno mediante la exportación de un archivo `environment.yml`.

El entorno se ha configurado desde cero y ha sido utilizado para ejecutar todos los entrenamientos, evaluaciones y análisis de resultados descritos a lo largo del proyecto. Gracias a esta configuración controlada, ha sido posible mantener la coherencia experimental entre distintos escenarios de prueba (entrenamiento desde cero, *fine-tuning*, y modelo base).

5.1.2. Gestión del entorno y dependencias

La gestión del entorno de desarrollo se ha realizado utilizando conda, lo que ha permitido crear un entorno virtual completamente aislado y reproducible. Este entorno ha sido configurado desde cero, instalando de forma controlada todas las librerías necesarias para el proyecto, evitando conflictos de versiones y asegurando la estabilidad durante los entrenamientos.

Las principales dependencias instaladas han sido:

- `threading` y `time`: para la gestión de procesos concurrentes y medición de tiempos de entrenamiento.
- `psutil` y `GPUtil`: para monitorización del uso de CPU y GPU en tiempo real.
- `numpy`: para operaciones numéricas y estadísticas.
- `gymnasium`: como interfaz base para los entornos de simulación MuJoCo.
- `matplotlib`: para la generación de gráficos y visualización de resultados.
- `stable-baselines3`: implementación del algoritmo PPO y otros métodos de Aprendizaje por Refuerzo.
- `json` y `os`: para lectura, almacenamiento estructurada de métricas y manejo del sistema de archivos.
- `torch` y `torch.nn`: como backend principal para las redes neuronales utilizadas por `stable-baselines3`.

El entorno ha sido documentado a través de un archivo `environment.yml` para facilitar su reproducción en otros sistemas. Este enfoque modular y controlado ha permitido realizar los entrenamientos y evaluaciones de manera robusta, evitando inconsistencias entre experimentos.

5.1.3. Hardware utilizado

Todos los entrenamientos se han realizado en un único equipo portátil para asegurar la homogeneidad del entorno de ejecución. El dispositivo utilizado ha sido un **ASUS TUF Gaming A15 FA507**, cuyas especificaciones técnicas relevantes son:

- **Procesador (CPU)**: AMD Ryzen 7 7735HS (8 núcleos, 16 hilos).
- **Tarjeta gráfica (GPU)**: NVIDIA GeForce RTX 4060 Laptop GPU (8 GB VRAM).
- **Memoria RAM**: 16 GB DDR5.
- **Almacenamiento**: SSD NVMe de 1 TB.

- **Sistema operativo:** Windows 11 Pro, utilizando entorno de ejecución conda para compatibilidad con MuJoCo y entornos gym.

Este hardware proporciona una capacidad de cómputo suficiente para entrenar agentes complejos en simulaciones MuJoCo con monitorización en tiempo real del uso de recursos. Además, mantener una plataforma única durante toda la experimentación permite comparar el rendimiento y la eficiencia energética entre enfoques sin introducir sesgos derivados del hardware.

5.1.4. Estructura de experimentación

Para cada experimento (entrenamiento desde cero, entrenamiento base y *fine-tuning*), se han generado:

- Un script dedicado con callbacks personalizados.
- Un modelo final almacenado en formato .zip.
- Un archivo .json con las métricas de evaluación y consumo.
- Archivos de log y gráficos opcionales para visualización posterior.

Esta estructura uniforme ha facilitado tanto la comparación entre enfoques como el análisis posterior de resultados.

5.2 Estructura del código y organización funcional

El desarrollo del proyecto se ha estructurado en torno a una colección de **scripts independientes**, cada uno de los cuales contiene de forma interna toda la lógica necesaria para ejecutar un experimento completo. Esta elección permite mantener el control completo sobre cada configuración concreta, evitando dependencias cruzadas y facilitando la trazabilidad individual de cada entrenamiento.

5.2.1. Scripts principales de entrenamiento

Cada script implementa de principio a fin una variante experimental del proyecto, ya sea un entrenamiento desde cero, el entrenamiento del modelo base o técnicas de *fine-tuning*. Además, cada uno de los scripts principales ha sido ejecutado en **cuatro versiones diferentes**, correspondientes a los distintos presupuestos de entrenamiento considerados:

$$\{2 \times 10^6, 3 \times 10^6, 5 \times 10^6, 10 \times 10^6 \text{ timesteps}\}$$

Los scripts principales desarrollados son los siguientes:

- `train_walker_base.py` (1): entrenamiento del modelo base en Walker2d-v5, con randomización de dominio.
- `train_from_scratch_humanoid.py` (2): entrenamiento desde cero en Humanoid-v5.
- `train_from_scratch_standup.py` (3): entrenamiento desde cero en HumanoidStandup-v5.
- `finetune_humanoid.py` (4): *fine-tuning* de Walker2d sobre Humanoid-v5.
- `finetune_standup.py` (5): *fine-tuning* de Walker2d sobre HumanoidStandup-v5.

Cada uno de estos scripts contiene los siguientes componentes integrados directamente en su estructura:

- Definición y configuración del entorno de entrenamiento (Gymnasium + MuJoCo).
- Inicialización del modelo PPO (desde cero o con transferencia).

- Callbacks específicos para evaluación o descongelado de capas.
- Monitorización de uso de CPU y GPU.
- Cálculo del consumo energético estimado.
- Guardado del modelo final y exportación de métricas.

5.2.2. Callbacks personalizados embebidos

Aunque en otros proyectos se opta por extraer los callbacks a módulos independientes, en este caso se han implementado de forma **local dentro de cada script**. Esto permite adaptar cada callback de forma específica a las necesidades de ese entrenamiento concreto, sin forzar una estructura genérica que pudiera limitar la flexibilidad.

Se han definido principalmente dos tipos:

- `EvaluationMetricsCallback`: registra recompensas medias, tiempo de entrenamiento y pasos evaluados.
- `ProgressiveUnfreezeCallback`: gestiona el desbloqueo progresivo de las capas congeladas tras un porcentaje específico del total de timesteps.

Ambos se encuentran implementados directamente en los scripts de *fine-tuning*, ajustados al caso concreto.

5.2.3. Monitorización integrada por script

Para cuantificar el impacto computacional de cada experimento, se ha implementado un sistema de monitorización local de uso de recursos. Esta monitorización se realiza de forma asíncrona y no intrusiva, mediante un hilo en segundo plano que registra periódicamente el uso de CPU y GPU durante todo el proceso de entrenamiento.

Las funciones de monitorización se han definido de forma local en cada script de entrenamiento, utilizando las librerías `psutil` (para la CPU) y `GPUTil` (para la GPU). Esta solución tiene varias ventajas:

- **Independencia de herramientas externas**: evita dependencias con plataformas de monitorización avanzadas o entornos específicos, manteniendo la portabilidad del código.
- **Flexibilidad**: el intervalo de muestreo puede ajustarse fácilmente según las necesidades del experimento.
- **Integración directa**: al estar embebida dentro del flujo de entrenamiento, la monitorización se inicia y finaliza automáticamente junto con el proceso de aprendizaje.

La información recogida se almacena en listas globales que se actualizan en tiempo real. Una vez completado el entrenamiento, se calcula la media de uso para cada recurso. Estos se calculan a partir de porcentajes medios de uso obtenidos; se estima el consumo energético total combinando tres factores:

1. Porcentaje medio de uso de CPU y GPU.
2. Potencia máxima estimada de cada componente, basada en las especificaciones del equipo portátil utilizado:
 - **CPU (AMD Ryzen 7 7735HS)**: 54 W [43]
 - **GPU (NVIDIA RTX 4060 Laptop)**: 115 W [44]
3. Duración total del entrenamiento en horas.

El consumo total se calcula aplicando la siguiente fórmula:

$$\text{energy_kWh} = \frac{(\text{CPU_power} + \text{GPU_power}) \times \text{Training_hours}}{1000}$$

donde:

$$\begin{aligned}\text{CPU_power} &= \left(\frac{\text{avg_cpu}}{100} \right) \times 54 \text{ (W)} \\ \text{GPU_power} &= \left(\frac{\text{avg_gpu}}{100} \right) \times 115 \text{ (W)} \\ \text{Training_hours} &= \frac{\text{training_time_sec}}{3600}\end{aligned}$$

Este enfoque permite obtener una estimación del consumo energético en kWh, útil para evaluar la eficiencia de cada estrategia de entrenamiento.

Una vez finalizado cada experimento, los valores clave se almacenan en un archivo .json. Esto facilita la trazabilidad y análisis posterior, y permite una comparación directa entre variantes experimentales sin necesidad de repetir ejecuciones.

El guardado automático de estos datos también contribuye a la reproducibilidad del proyecto y a la documentación de resultados en escenarios con múltiples repeticiones o presupuestos de entrenamiento.

5.2.4. Justificación de la estructura

La elección de integrar todos los elementos dentro de cada script responde a criterios de claridad y trazabilidad experimental. En lugar de optar por una arquitectura modular clásica, se ha priorizado la encapsulación funcional para que cada experimento sea completamente autónomo y autocontenido. Esto ha permitido:

- Tener control total sobre cada configuración de entrenamiento.
- Evitar errores derivados de dependencias compartidas entre experimentos.
- Facilitar la lectura, ejecución y depuración individual de cada script.

Este enfoque ha demostrado ser especialmente útil en un contexto de investigación exploratoria, donde las variantes del entrenamiento pueden requerir ajustes puntuales en callbacks, métricas o configuración del entorno.

5.3 Implementación del modelo base

El modelo base constituye el punto de partida para las estrategias de *fine-tuning* exploradas en este proyecto. Ha sido entrenado sobre el entorno Walker2d-v5 y diseñado específicamente para ser reutilizable en tareas de locomoción más complejas, como Humanoid-v5 y HumanoidStandup-v5. Esta implementación incorpora técnicas de robustez como la randomización de dominio, con el objetivo de mejorar la generalización de las representaciones aprendidas.

Para valorar la solidez de nuestro modelo base —ya que constituye el punto de partida del proyecto— se ha analizado su rendimiento de forma comparativa dentro del propio marco experimental. Aunque se han utilizado referencias externas, no se partía de la expectativa de superarlas y que entrenar desde cero en versiones más recientes del entorno Walker2d presentaría mayor dificultad, dado el aumento progresivo en complejidad y penalizaciones.

A pesar de ello, el modelo base entrenado desde cero alcanzó un rendimiento notable, situándose como referencia interna válida para posteriores experimentos. Además, al aplicar la técnica de domain randomization, se observó un incremento significativo en la recompensa, llegando a superar los 4000 puntos. Este resultado evidencia el potencial de las técnicas de generalización para este tipo de entornos continuos.

5.3.1. Resultados reales del modelo base: comparación y análisis

Con el objetivo de evaluar la eficacia del modelo base propuesto, se han entrenado dos versiones del agente en el entorno Walker2d-v5: una utilizando **domain randomization** y otra sin aplicar dicha técnica. Ambos modelos han sido entrenados con el mismo presupuesto de 2×10^6 & 3×10^6 timesteps y bajo condiciones de hardware controladas.

Además, se ha incluido un tercer punto de comparación procedente de benchmarks publicados en literatura y repositorios de referencia como *Spinning Up* de OpenAI, los cuales establecen valores de recompensa típicos para este entorno en configuraciones estándar.

Contextualización frente a benchmarks publicados

Los valores de recompensa típicos para entornos MuJoCo como Walker2d-v5 oscilan generalmente entre un 5 % y un 10 % de variabilidad según la implementación exacta, semilla aleatoria, versión del entorno y cambios internos en la arquitectura. Repositorios como *Spinning Up* de OpenAI reportan recompensas base de unos ~ 1230 para modelos PPO sin ajustes específicos, con máximos de hasta ~ 3460 aplicando técnicas no siempre documentadas [45].

En ese contexto, el modelo entrenado con randomización no solo supera claramente los valores estándar de PPO, sino que alcanza cifras comparables con implementaciones de algoritmos más avanzados (SAC, TD3, DDPG), lo cual resalta su valor como punto de partida para transferencia.

Gráficos de evolución y análisis visual

La siguiente tabla muestra una comparativa directa entre los tres enfoques:

Métrica	Domain Randomized	From Scratch	Literatura
Recompensa final	3846.99	1170.42	3460
Tiempo de entrenamiento (s)	2520.83	2393.30	–
Consumo energético (kWh)	0.0452	0.0440	–

Tabla 5.1: Comparación entre modelos base y benchmark en Walker2d-v5

Para complementar la tabla anterior, se han generado gráficos 5.1, que muestran la evolución temporal de la recompensa media durante el entrenamiento, los cuales refuerzan visualmente la superioridad del modelo con randomización en términos de estabilidad y velocidad de convergencia.

Si bien, el tiempo de entrenamiento y el consumo energético en este primer caso es ligeramente mayor, la diferencia es negligible y la recompensa es más del triple en el entrenamiento con randomización lo cual permite a nuestro agente generalizar mejor, por lo que se ha decidido utilizar el agente con randomización como base para que los agentes de los entornos avanzados rindan mejor en todos los aspectos.

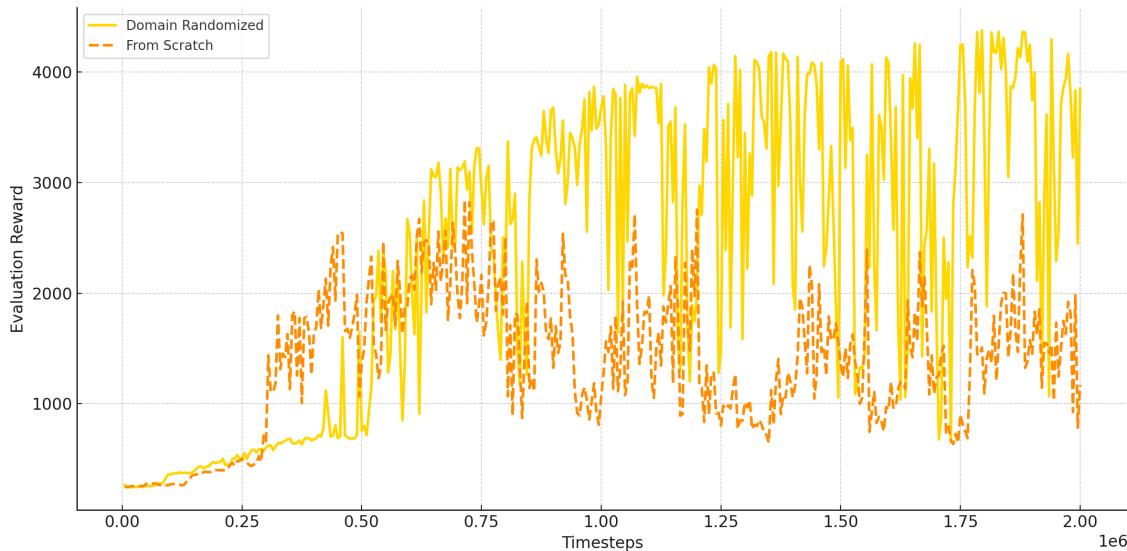


Figura 5.1: Modelo Base Walker2d

5.4 Implementación del modelo benchmark

Con el objetivo de establecer una línea base comparativa, se han entrenado dos agentes desde cero en los entornos Humanoid-v5 y HumanoidStandup-v5. Estos modelos benchmark permiten medir de forma objetiva el impacto del uso de un modelo base preentrenado frente al entrenamiento tradicional, tanto en términos de rendimiento como de eficiencia computacional y energética.

5.4.1. Entrenamiento benchmark en Humanoid-v5

Este agente ha sido entrenado desde cero en el entorno Humanoid-v5, donde la tarea principal consiste en caminar de forma estable. El entorno representa un desafío complejo de locomoción debido al gran número de articulaciones, el espacio de observación de alta dimensión y la necesidad de mantener el equilibrio dinámico.

- **Observaciones:** 376 dimensiones.
- **Acciones:** 17 acciones continuas.
- **Política:** MlpPolicy, sin preentrenamiento.
- **Evaluación:** cada 5000 pasos, usando 5 episodios.
- **Presupuestos de entrenamiento:** $2 \times 10^6, 3 \times 10^6, 5 \times 10^6, 10 \times 10^6$ timesteps.

Durante el entrenamiento se registraron métricas clave de rendimiento (recompensa media), consumo energético estimado, uso medio de CPU y GPU, y tiempo total de entrenamiento. Este modelo actúa como referencia directa frente al modelo *fine-tuned* proveniente de Walker2d-v5.

5.4.2. Entrenamiento benchmark en HumanoidStandup-v5

De forma análoga, se ha entrenado un segundo agente desde cero en el entorno HumanoidStandup-v5. En este entorno, la tarea consiste en levantarse desde una posición tumbada hasta mantenerse en pie de forma estable, implicando una secuencia motora diferente y más explosiva que la de caminar.

- **Observaciones:** 376 dimensiones.

- **Acciones:** 17 acciones continuas.
- **Política:** MlpPolicy, sin conocimiento previo.
- **Evaluación:** cada 5000 pasos en 5 episodios.
- **Presupuestos de entrenamiento:** $2 \times 10^6, 3 \times 10^6, 5 \times 10^6, 10 \times 10^6$ timesteps.

Este modelo benchmark se utiliza como referencia de rendimiento y consumo frente al modelo que ha sido *fine-tuned* desde el modelo base de Walker2d-v5. El análisis comparativo entre ambos enfoques permite evaluar la ganancia real obtenida mediante técnicas de transferencia en una tarea estructuralmente distinta.

5.4.3. Resultados benchmark Humanoid-v5

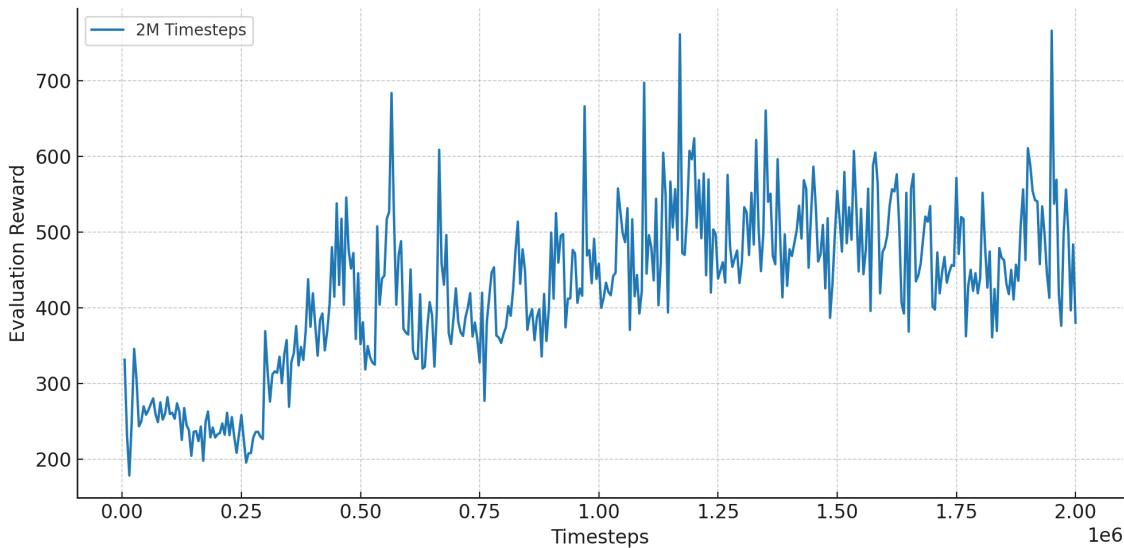


Figura 5.2: Humanoid Benchmark 2M

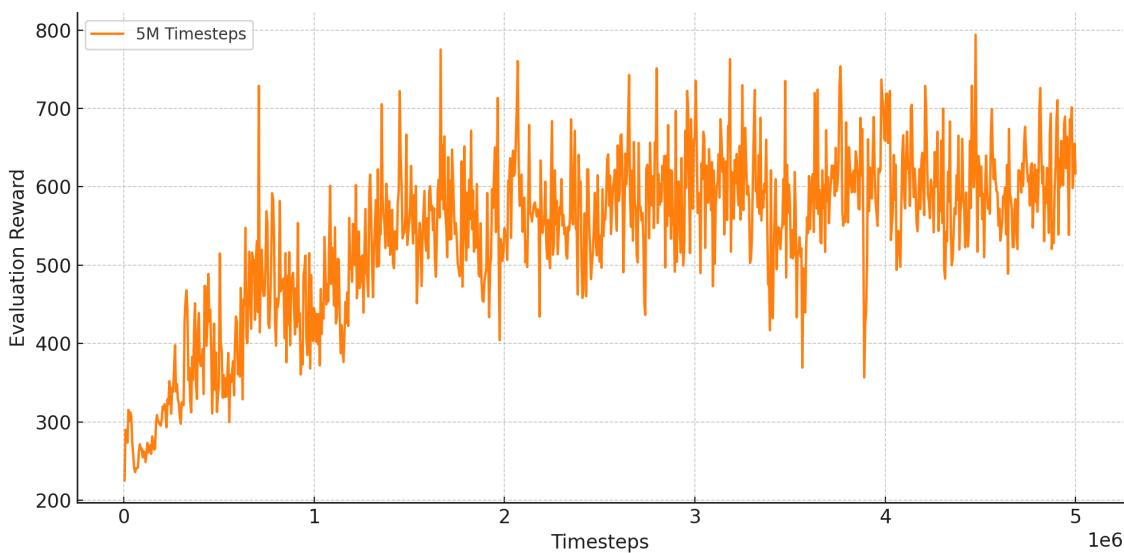


Figura 5.3: Humanoid Benchmark 5M

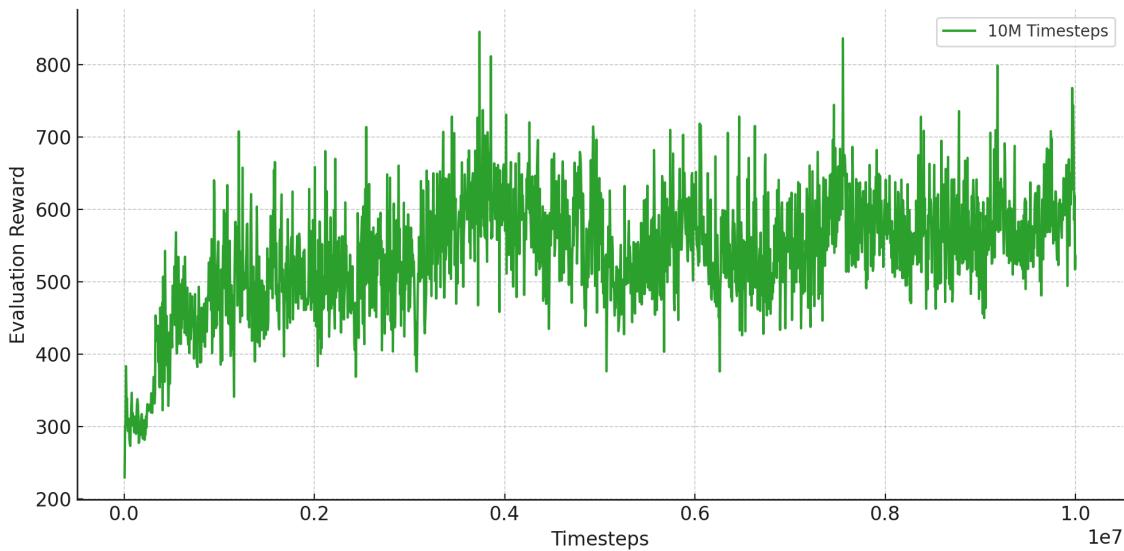


Figura 5.4: Humanoid Benchmark 10M

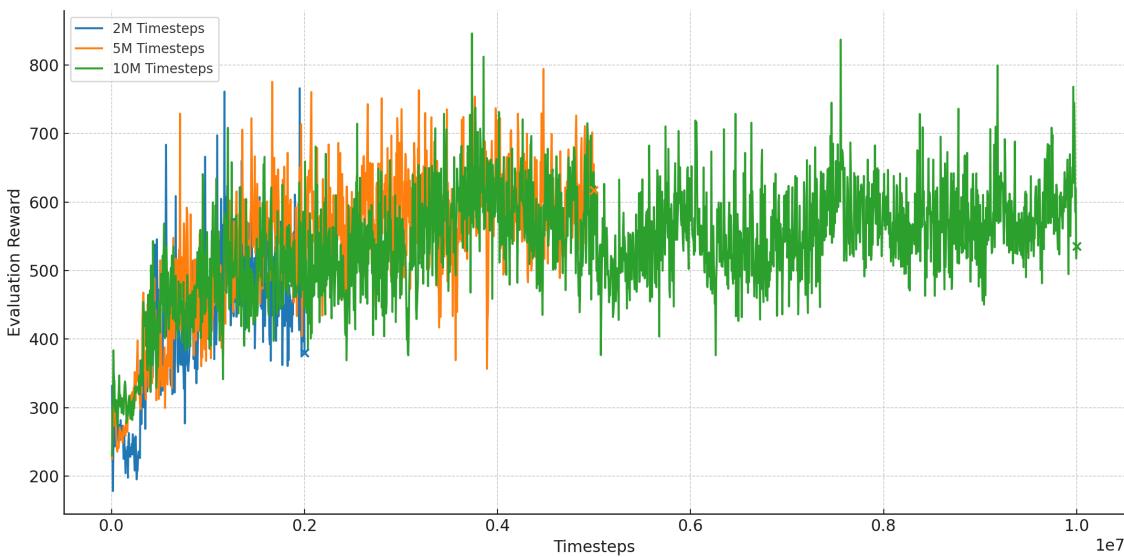


Figura 5.5: Humanoid Benchmark WrapUp

Las figuras 5.2, 5.3 y 5.4 presentan las curvas de recompensa promedio obtenidas durante el entrenamiento de los agentes en el entorno Humanoid-v5 bajo distintos presupuestos de pasos de tiempo: 2M, 5M y 10M timesteps respectivamente. Cada gráfico refleja la evolución del rendimiento del agente a lo largo del entrenamiento, evaluado cada 5000 pasos con el promedio de cinco episodios.

En la Figura 5.2, correspondiente al entrenamiento con un presupuesto de 2M timesteps, se observa un crecimiento inicial en la recompensa, aunque con alta variabilidad. El agente comienza a desarrollar una política locomotora básica, pero aún inestable, lo que se refleja en las frecuentes oscilaciones de desempeño.

La Figura 5.3 muestra la curva para el caso de 5M timesteps. En este escenario, el agente logra consolidar comportamientos de locomoción más estables. La recompensa promedio alcanza valores cercanos a 750, con una clara tendencia ascendente a lo largo del entrenamiento. Este resultado evidencia que, con un mayor número de interacciones

con el entorno, el agente es capaz de refinar su política y mantener el equilibrio de forma más consistente.

Finalmente, en la Figura 5.4, con un presupuesto de 10M timesteps, se alcanza el mayor nivel de desempeño. La recompensa promedio supera los 800 en varias evaluaciones, y aunque aún se observan picos de variación, la tendencia general es más estable que en los casos anteriores. Esto confirma que, si bien el entorno Humanoid-v5 presenta una complejidad considerable, el entrenamiento desde cero puede llevar al agente a descubrir soluciones locomotoras eficaces si se le otorga suficiente tiempo de interacción.

La Figura 5.5 proporciona una visión comparativa de los tres entrenamientos. Se puede apreciar cómo el incremento progresivo en los timesteps tiene un efecto directo en la mejora del rendimiento. Esta línea base sirve como punto de referencia para evaluar el impacto de técnicas de transferencia como el fine-tuning desde un modelo Walker2d.

Además, con el objetivo de evaluar si las mejoras observadas en Walker2d-v5 mediante *domain randomization* eran transferibles a entornos más complejos, se entrenaron dos versiones del agente en Humanoid-v5: una desde cero sin ninguna alteración del entorno, y otra aplicando una técnica de randomización de dominio similar a la utilizada previamente.

Los resultados experimentales, apoyados por las gráficas incluidas en esta sección, demuestran que en este entorno más complejo la técnica de randomización no solo no mejora el rendimiento, sino que puede degradarlo. A pesar de compartir el algoritmo PPO, los resultados de recompensa final con randomización fueron similares o incluso inferiores a los obtenidos sin ella, con una penalización notable en estabilidad y convergencia del entrenamiento.

Este fenómeno se explica por las características intrínsecas del agente Humanoid:

- **Alto número de articulaciones (17)** y grados de libertad.
- **Dependencia extrema del equilibrio multieje**, que vuelve el sistema altamente sensible a perturbaciones físicas.
- **Modos de fallo catastróficos**, donde pequeñas variaciones en masa, gravedad o fricción conducen directamente a caídas y pérdida total de recompensa.
- **Política menos robusta**, que colapsa con mayor facilidad si no se entrena bajo condiciones muy controladas.

En términos computacionales, los entrenamientos con randomización en este entorno demandaron más recursos y más tiempo para lograr resultados comparables. Esto reduce significativamente su viabilidad dentro de un marco centrado en la sostenibilidad y la eficiencia.

Las gráficas adjuntas 5.6, 5.7, 5.8, muestran cómo la versión sin randomización converge de forma más estable y logra mejores recompensas medias finales en todos los presupuestos de entrenamiento evaluados.

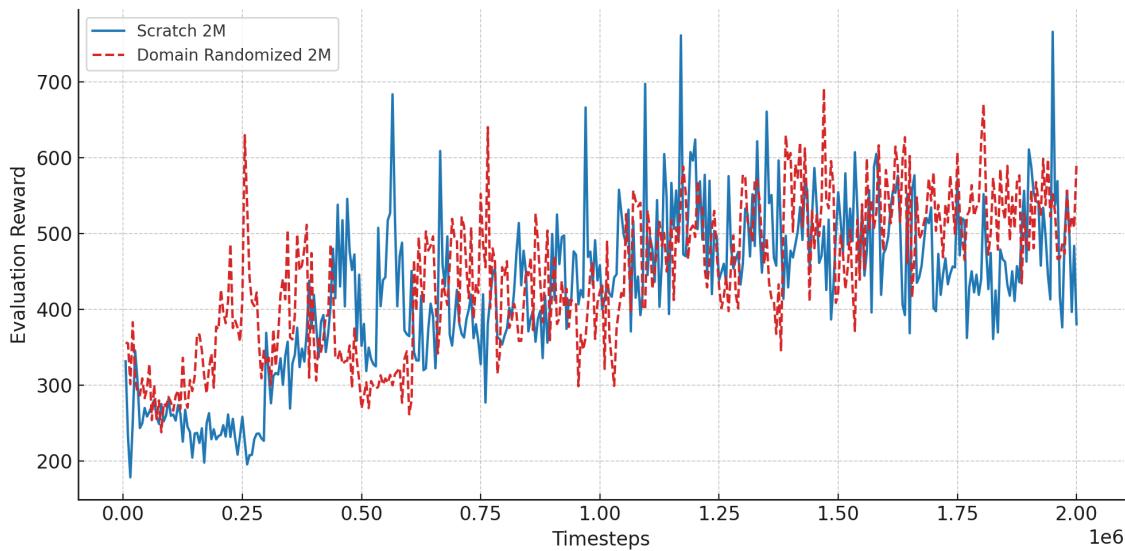


Figura 5.6: Domain random humanoid benchmark 2M

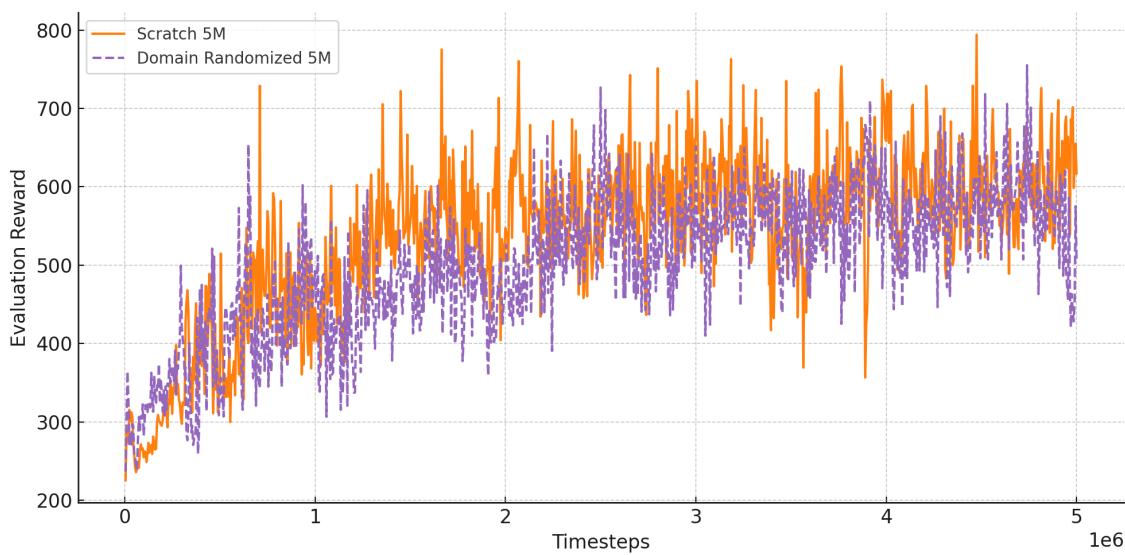


Figura 5.7: Domain random humanoid benchmark 5M

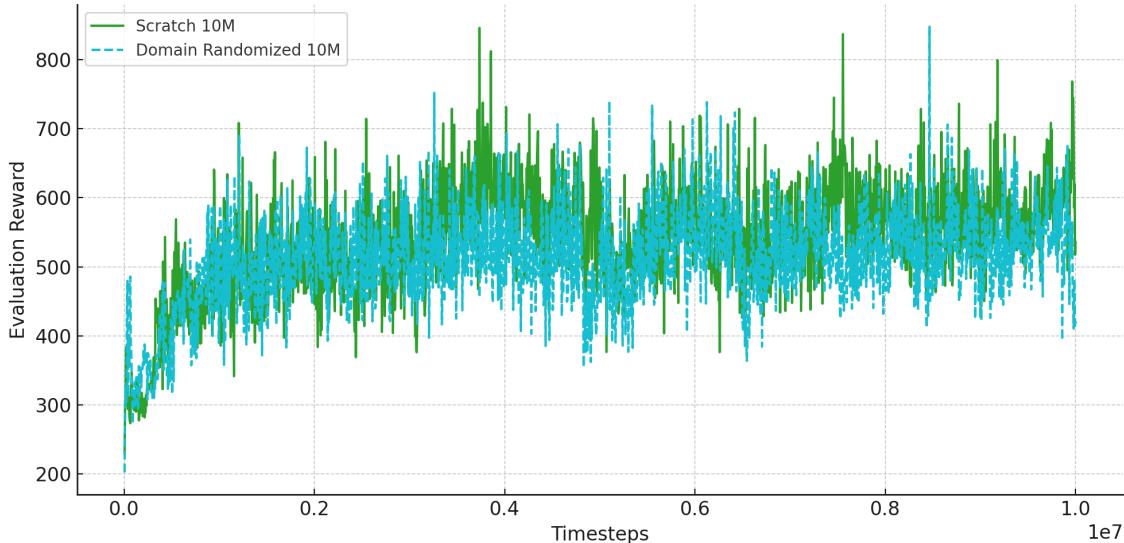


Figura 5.8: Domain random humanoid benchmark 10M

Tabla 5.2: Recompensa total y consumo energético para el modelo Humanoid desde cero y desde cero con domain randomization.

Timesteps(M)	Reward Scratch	Reward Scratch+DR	Energy Scratch(kWh)	Energy Scratch+DR(kWh)
2M	593.04	593.15	0.0401	0.0404
5M	617.13	506.93	0.0999	0.1196
10M	560.27	421.48	0.1131	0.1218

5.4.4. Resultados benchmark en HumanoidStandup-v5

A diferencia de los experimentos realizados con Humanoid-v5, en el entorno HumanoidStandup-v5 se ha optado por no aplicar *domain randomization* durante el entrenamiento benchmark. Esta decisión se fundamenta directamente en los resultados negativos observados en el agente humanoide caminante.

Ambos entornos comparten la misma morfología y una elevada sensibilidad a perturbaciones físicas. De hecho, la tarea de ponerse en pie desde el suelo —propia de HumanoidStandup— es aún más inestable en sus primeras fases, lo que la hace especialmente vulnerable a los efectos adversos de la aleatorización de parámetros como gravedad, fricción o masa.

Aplicar *domain randomization* en este entorno supondría un coste computacional elevado, acompañado de una mayor dificultad para alcanzar convergencia en políticas útiles. Dados los resultados previos en Humanoid-v5, se considera que la técnica no aporta beneficios significativos y que introducirla podría:

- Aumentar innecesariamente el tiempo de entrenamiento y el consumo energético.
- Generar inestabilidad en las primeras fases de aprendizaje, donde el agente debe salir de una posición tumbada.
- No mejorar el rendimiento final ni la capacidad de generalización frente a un entrenamiento más controlado.

Por tanto, y alineado con los objetivos del proyecto centrados en la eficiencia y la reutilización, el modelo benchmark de HumanoidStandup-v5 se ha entrenado exclusivamente sin randomización, utilizando una configuración estándar de PPO como línea base para comparar con los modelos *fine-tuned*.

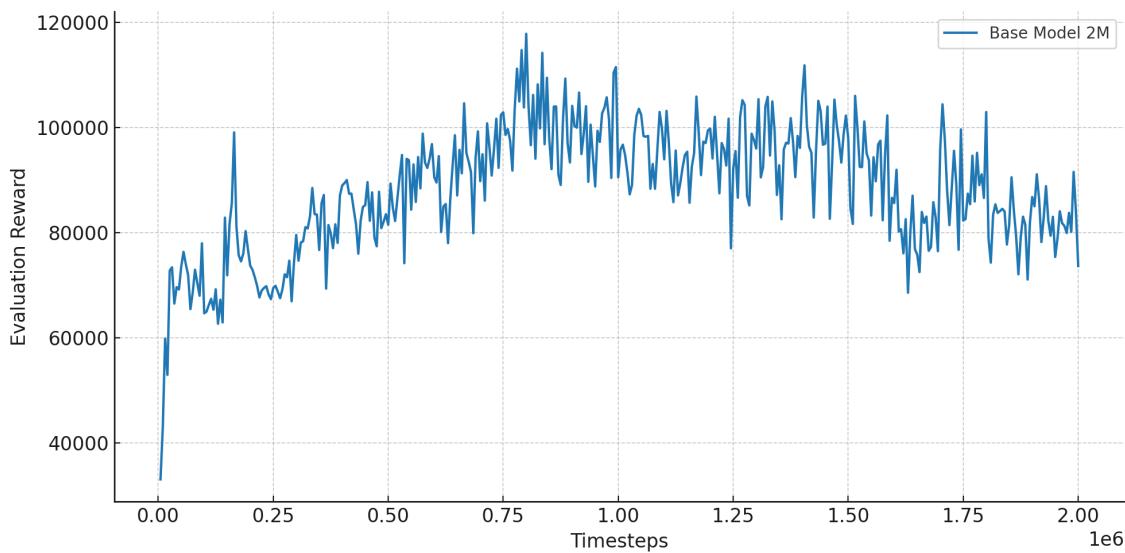


Figura 5.9: Standup benchmark 2M

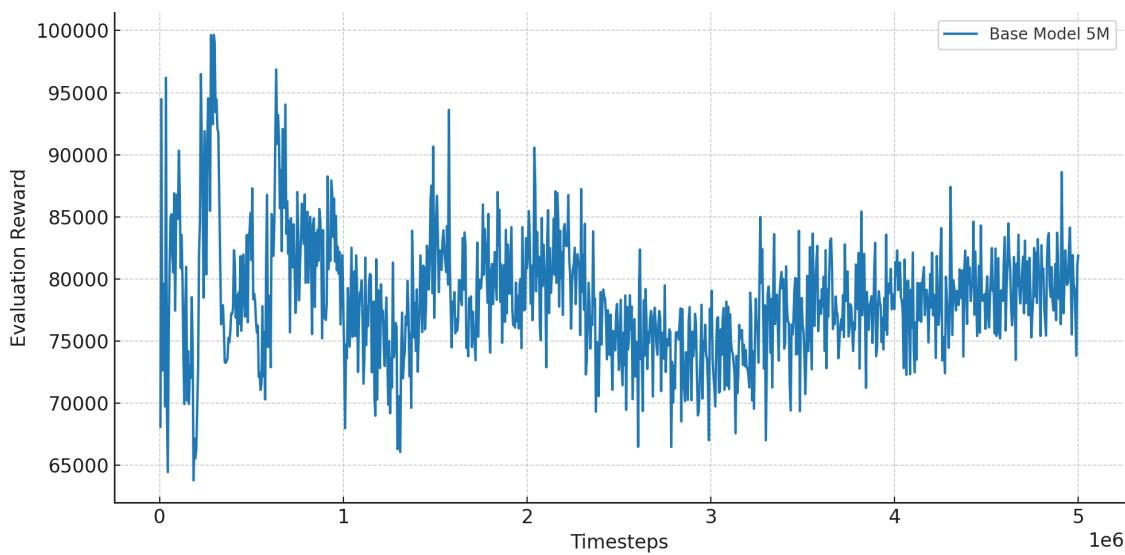


Figura 5.10: Standup benchmark 5M

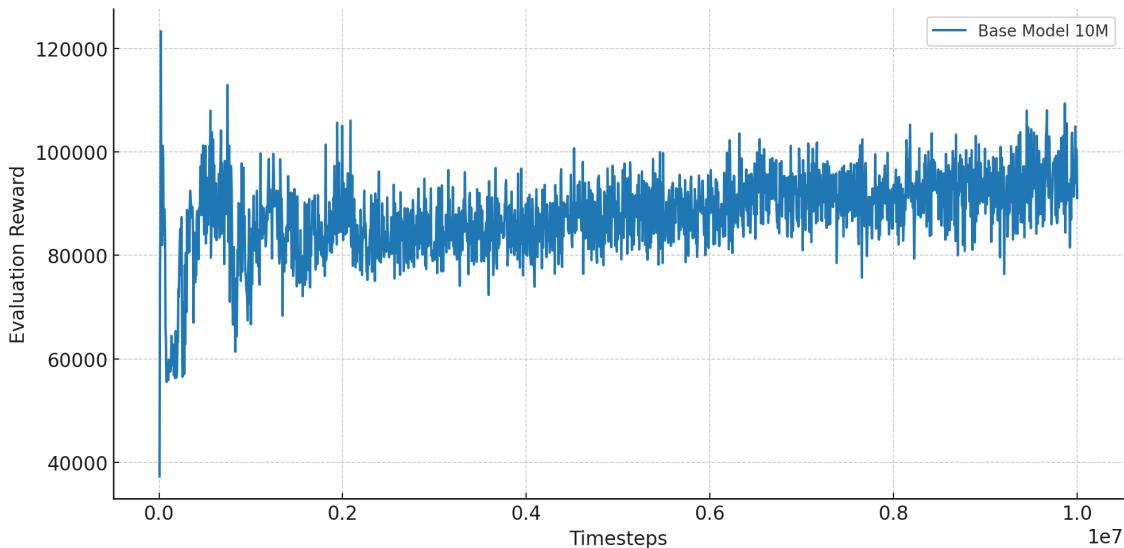


Figura 5.11: Standup benchmark 10M

Las figuras 5.9, 5.10 y 5.11 presentan los resultados del entrenamiento del agente en el entorno HumanoidStandup-v5 para presupuestos de 2M, 5M y 10M timesteps respectivamente. Este entorno plantea un reto distinto al de Humanoid-v5, ya que la tarea requiere que el agente pase de una posición tumbada a mantenerse erguido, lo que implica una dinámica motora explosiva, coordinación secuencial y balance postural complejo.

En la Figura 5.9, con un presupuesto de 2M timesteps, se observa un incremento rápido en la recompensa durante las primeras etapas del entrenamiento, alcanzando un rendimiento por encima de los 100000 puntos. Sin embargo, tras este pico inicial, el rendimiento comienza a mostrar cierta inestabilidad. Esta oscilación puede estar relacionada con el hecho de que el agente aún no ha consolidado de forma robusta la secuencia completa de levantamiento y estabilización.

En la Figura 5.10, correspondiente al caso de 5M timesteps, el comportamiento del agente presenta una mayor variabilidad. A pesar de alcanzar picos cercanos a los 100000 puntos, el rendimiento general es más errático y sugiere una política menos consistente. Esta inestabilidad podría deberse a la sensibilidad del entorno a pequeñas desviaciones en la secuencia de acciones, lo cual resalta la dificultad inherente del problema de levantarse sin asistencia.

Por otro lado, en la Figura 5.11, el entrenamiento con 10M timesteps muestra una mejora sustancial en la estabilidad del rendimiento. La recompensa promedio se mantiene consistentemente por encima de los 90,000 puntos a lo largo del entrenamiento, con una variabilidad considerablemente menor en comparación con los casos anteriores. Esto sugiere que, con un mayor presupuesto de entrenamiento, el agente logra generalizar mejor la secuencia de levantamiento, adaptándose a pequeñas perturbaciones y manteniendo el equilibrio de forma más confiable.

Estos resultados constituyen la línea base para la posterior comparación con modelos fine-tuned, entrenados inicialmente en entornos locomotores como Walker2d-v5. La comparación directa permitirá cuantificar las ventajas reales del transfer learning en tareas de control complejas como el *HumanoidStandup*.

5.5 Implementación del fine-tuning

5.5.1. Resultados fine-tuning en Humanoid-v5

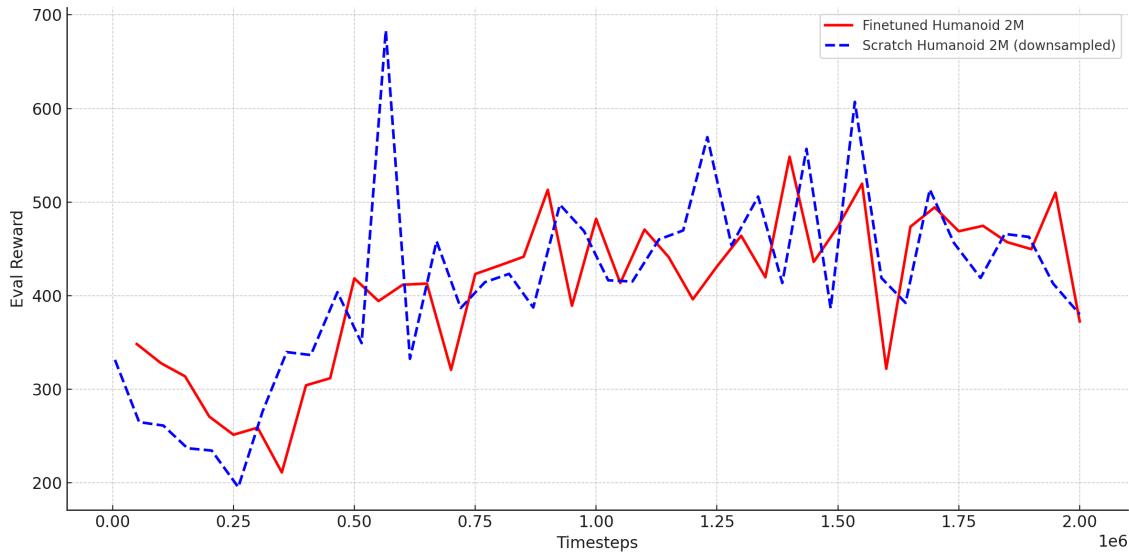


Figura 5.12: Fine-tuning humanoid 2M

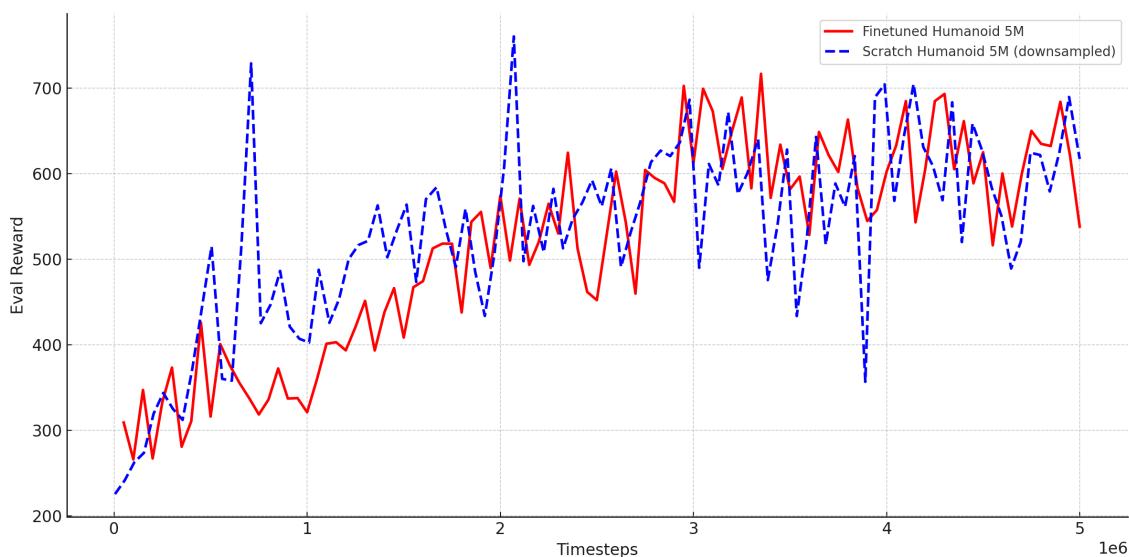


Figura 5.13: Fine-tuning humanoid 5M

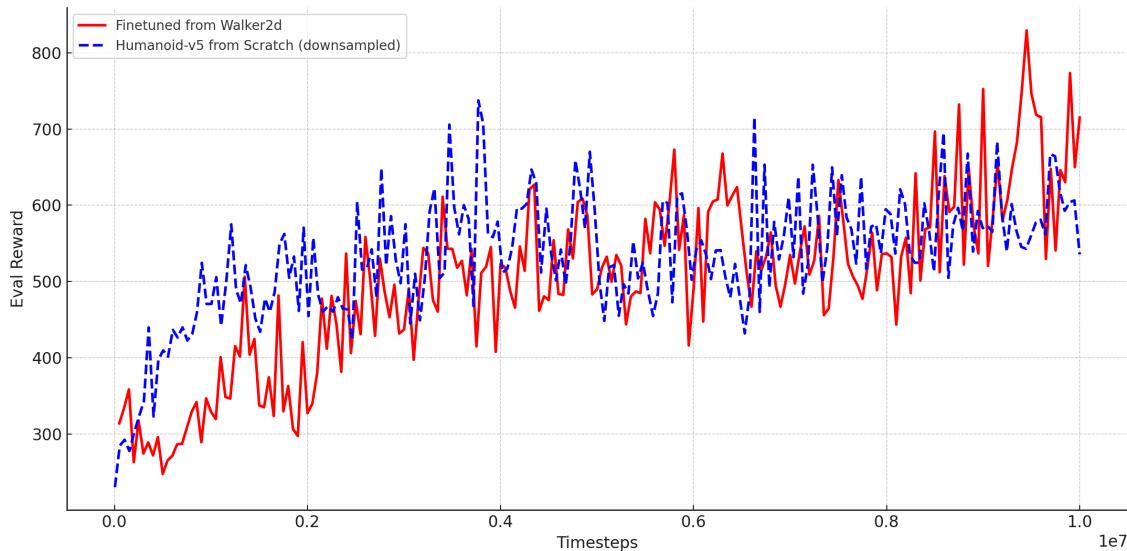


Figura 5.14: Fine-tuning humanoid 10M

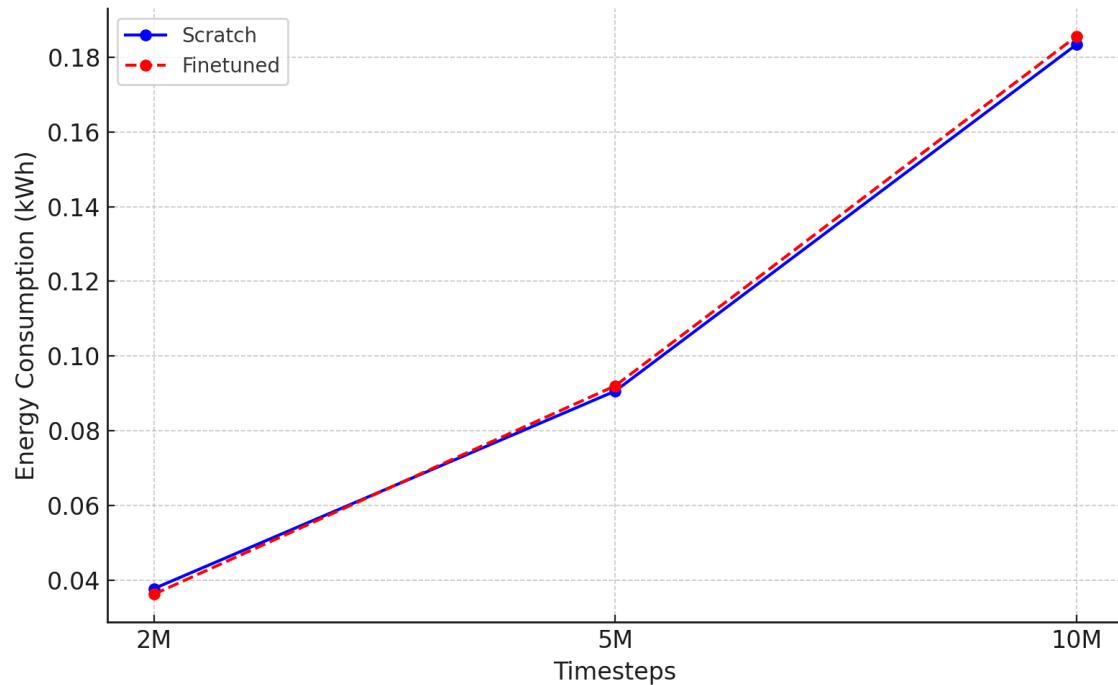


Figura 5.15: Comparación del consumo de energía de Humanoid

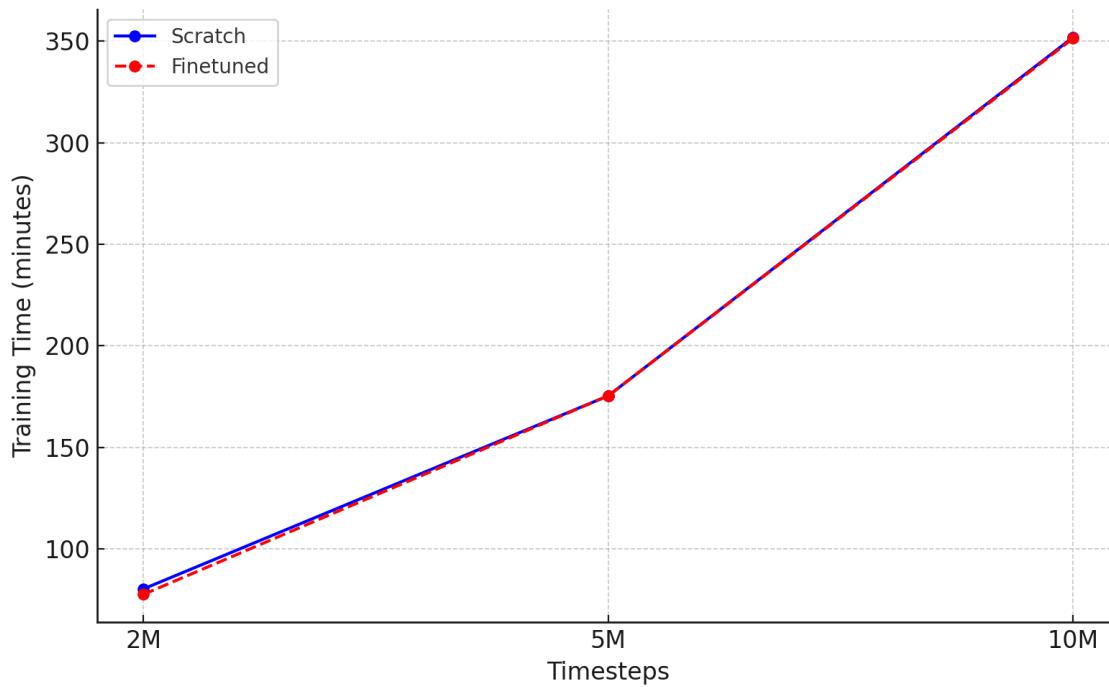


Figura 5.16: Comparación del tiempo de entrenamiento de Humanoid

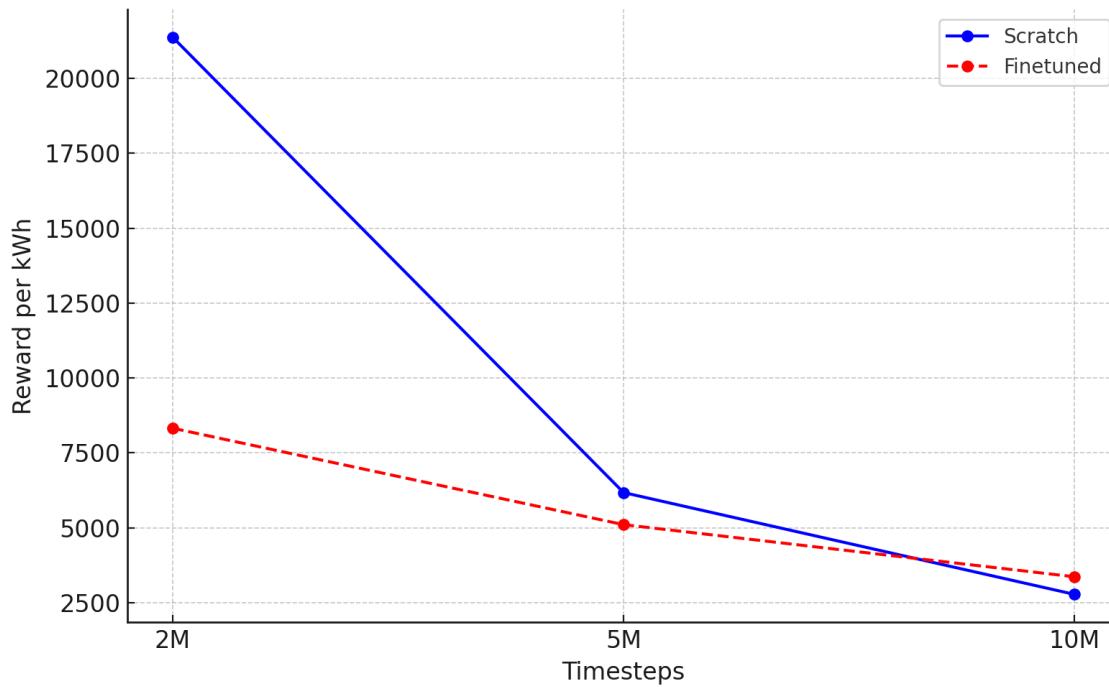


Figura 5.17: Comparación de la recompensa por energía usada del modelo Humanoid

Las figuras 5.12, 5.13 y 5.14 comparan el rendimiento de los agentes entrenados en el entorno Humanoid-v5 utilizando dos enfoques: entrenamiento desde cero y fine-tuning a partir de un modelo preentrenado en Walker2d-v5. Cada curva representa la evolución de la recompensa promedio en evaluaciones periódicas.

En la Figura 5.12, correspondiente al caso con 2 millones de timesteps, el modelo fine-tuned muestra una ventaja inicial clara en términos de estabilidad. Aunque su recompensa final no supera a la del modelo entrenado desde cero, su comportamiento es

más suave y menos errático. Esta diferencia se debe a la estrategia de congelación inicial de capas y descongelado progresivo, que prioriza la estabilidad sobre la velocidad de adaptación en las primeras fases del entrenamiento. En cambio, el modelo desde cero necesita explorar más agresivamente, lo que explica sus oscilaciones.

En la Figura 5.13, con un mayor presupuesto de entrenamiento, el modelo fine-tuned comienza a mostrar su verdadero potencial. Gracias al desbloqueo estructurado de capas tras el 30 % del entrenamiento, la política ajusta su conocimiento previamente aprendido a las dinámicas específicas del nuevo entorno. Esto se traduce en una curva más ascendente y sostenida, alcanzando recompensas significativamente superiores a partir del último tercio del entrenamiento.

En la Figura 5.14, el modelo fine-tuned consolida su ventaja. Su rendimiento final es claramente superior (715 frente a 535), y además lo consigue con una curva mucho más estable. Mientras el modelo desde cero muestra una tendencia oscilante e incluso decreciente, el agente fine-tuned continúa mejorando progresivamente, beneficiándose del conocimiento estructural transferido desde Walker2d y del entrenamiento prolongado con todas las capas ya activas.

Este comportamiento se debe a la estrategia de descongelado progresivo aplicada en los modelos fine-tuned, que permite preservar las representaciones generales durante las primeras fases del entrenamiento y adaptar la política de forma gradual y controlada conforme avanza el proceso. Este enfoque estructurado aporta una combinación óptima entre estabilidad inicial y capacidad de adaptación, especialmente útil en entornos de alta complejidad como Humanoid-v5.

Las figuras 5.15, 5.16 y 5.17 aportan un análisis complementario desde el punto de vista energético y computacional. En la Figura 5.15, se observa que el consumo energético total es similar en ambos enfoques, lo que permite comparar de forma directa su productividad relativa.

La Figura 5.16 confirma que los tiempos de entrenamiento también son equivalentes, lo cual elimina sesgos en cuanto a recursos empleados. Esta paridad en los costes computacionales refuerza la validez del análisis de eficiencia energética.

La Figura 5.17, que muestra la métrica de recompensa por kWh consumido, refleja inicialmente una ventaja del modelo desde cero en presupuestos reducidos. Sin embargo, a medida que el entrenamiento se prolonga, el modelo fine-tuned incrementa su eficiencia energética, superando al modelo entrenado desde cero en el caso de 10 millones de pasos. Esta tendencia sugiere que el enfoque de transferencia no solo es escalable, sino que además mejora su productividad con el tiempo.

La combinación de aprendizaje progresivo, políticas iniciales robustas y menor exploración aleatoria permite al modelo fine-tuned evolucionar de forma más estable y eficiente, incluso en tareas que difieren parcialmente de su entorno original. En contraste, el modelo entrenado desde cero, aunque competitivo en fases iniciales, tiende a perder eficacia con la complejidad y la duración del entrenamiento.

Tabla 5.3: Recompensa total y consumo energético estimado para los modelos entrenados desde cero y con fine-tuning.

Timesteps (M)	Reward Scratch	Reward FT	Energy Scratch (kWh)	Energy FT (kWh)
2M	523.66	372.61	0.0195	0.0366
5M	617.14	538.10	0.1000	0.1013
10M	535.60	715.08	0.1931	0.1856

Tabla 5.4: Eficiencia energética estimada para los modelos entrenados desde cero y con fine-tuning.

Timesteps (M)	Reward/kWh Scratch	Reward/kWh FT
2M	26854.36	10180.60
5M	6171.40	5311.94
10M	2773.69	3852.80

Estos resultados confirman que el uso de modelos base preentrenados puede ser altamente beneficioso en tareas de locomoción compleja como Humanoid-v5, no sólo en términos de rendimiento final, sino también en eficiencia energética, estabilidad y velocidad de aprendizaje.

5.5.2. Resultados fine-tuning en HumanoidStandup-v5

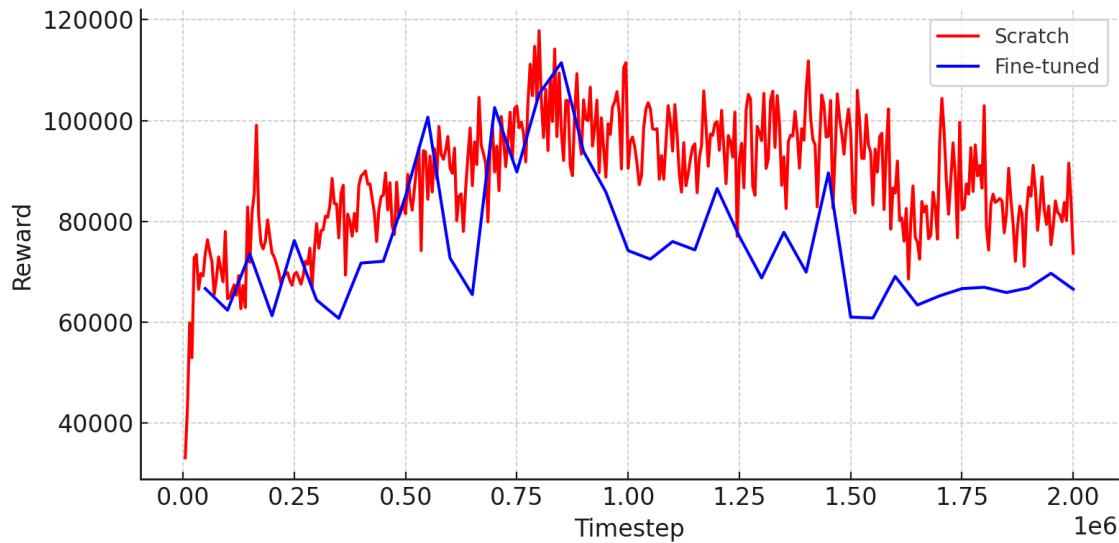


Figura 5.18: Fine-tuning Standup 2M

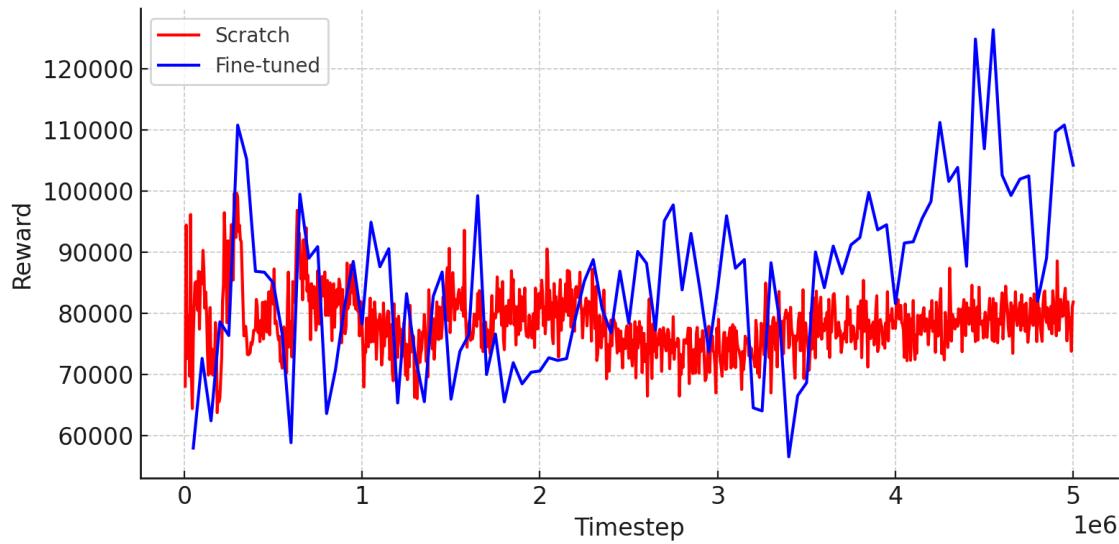


Figura 5.19: Fine-tuning Standup 5M

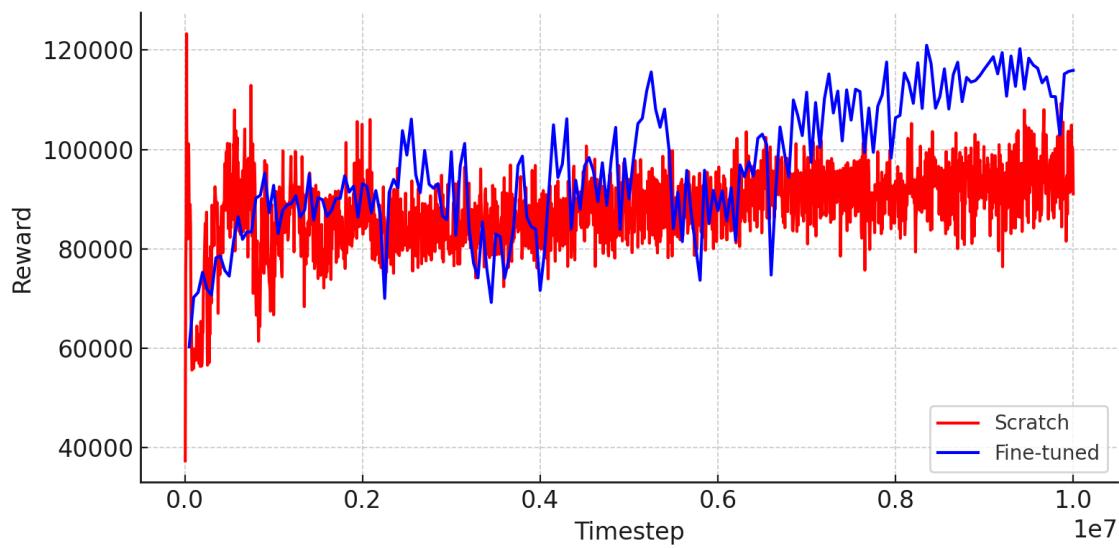


Figura 5.20: Fine-tuning Standup 10M

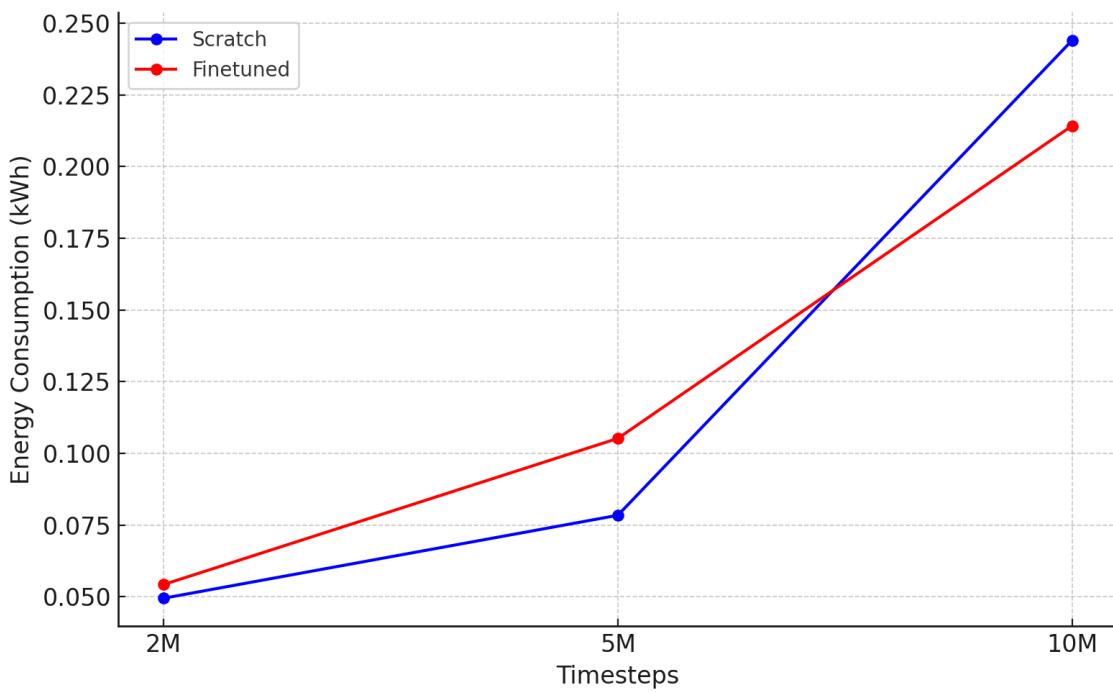


Figura 5.21: Comparación del consumo de energía de HumanoidStandup

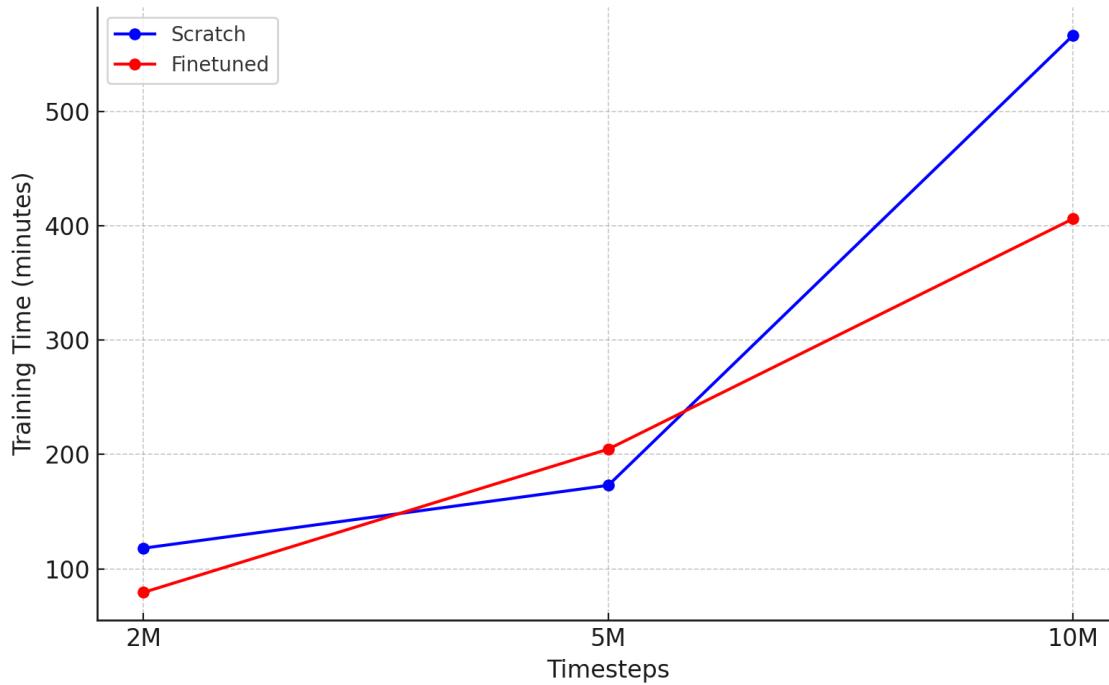


Figura 5.22: Comparación del tiempo de entrenamiento de HumanoidStandup

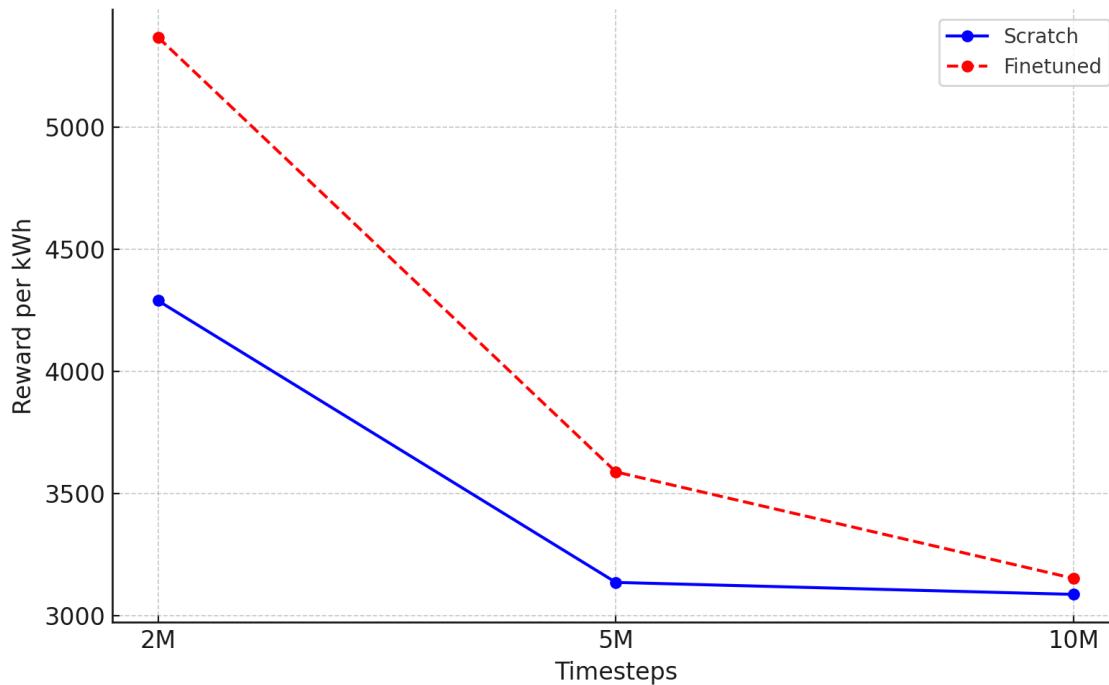


Figura 5.23: Comparación de la recompensa por energía usada del modelo HumanoidStandup

Las figuras 5.18, 5.19 y 5.20 presentan los resultados obtenidos en el entorno HumanoidStandup-v5, contrastando agentes entrenados desde cero con aquellos que fueron fine-tuned a partir de un modelo preentrenado en Walker2d-v5. Este análisis no solo permite evaluar el rendimiento en términos de recompensa acumulada, sino también observar el impacto del fine-tuning sobre la eficiencia energética del entrenamiento.

En la Figura 5.18, con un presupuesto de 2 millones de timesteps, se observa que el modelo benchmark entrenado desde cero logra una mayor recompensa absoluta, aun-

que con una política más oscilante. A pesar de partir con una desventaja en términos de recompensa final, el modelo fine-tuned exhibe mayor estabilidad desde etapas tempranas del entrenamiento. No obstante, en términos de eficiencia energética (reward/kWh), el modelo desde cero también tiene una ligera ventaja, lo que sugiere que en esta etapa inicial el costo del fine-tuning aún no se amortiza completamente.

La Figura 5.19 refleja una inversión clara de esta tendencia. El modelo fine-tuned supera al benchmark tanto en recompensa acumulada como en estabilidad, indicando que la transferencia de conocimiento comienza a traducirse en beneficios tangibles. Aunque el consumo energético es mayor en el modelo fine-tuned, su eficiencia (recompensa por kWh) se aproxima a la del modelo desde cero, y su rendimiento final lo justifica.

Finalmente, la Figura 5.20 consolida los beneficios del fine-tuning: la recompensa final es superior, y se logra con mayor eficiencia energética respecto al modelo benchmark. El modelo fine-tuned muestra no solo una política más efectiva, sino también una mejor relación entre coste energético y recompensa lograda.

También, se muestran las figuras 5.21 y 5.22, las cuales presentan una comparación directa del consumo energético y el tiempo total de entrenamiento entre modelos entrenados desde cero y aquellos que utilizaron una estrategia de fine-tuning a partir de un modelo base preentrenado en Walker2d-v5.

En la Figura 5.21, se observa que el consumo energético total incrementa proporcionalmente al número de timesteps en ambos enfoques. No obstante, el modelo fine-tuned presenta sistemáticamente un consumo energético inferior al del modelo entrenado desde cero, siendo esta diferencia particularmente marcada en el escenario de 10 millones de timesteps. Este resultado sugiere que el modelo preentrenado logra una adaptación más eficiente al entorno, reduciendo la carga computacional acumulada a lo largo del entrenamiento. Lo que sugiere que el fine-tuning no solo facilita el aprendizaje, sino que también contribuye a una mejor gestión energética durante el proceso de optimización.

Por su parte, la Figura 5.22 evidencia que los modelos fine-tuned también requieren menos tiempo de entrenamiento total en comparación con los modelos desde cero. Esta diferencia es especialmente pronunciada en el entrenamiento con 10M timesteps, donde la reducción en tiempo alcanza más de 150 minutos. Este comportamiento se atribuye a la reutilización de representaciones motoras y estructurales ya aprendidas, lo que permite al modelo fine-tuned converger más rápidamente hacia una política efectiva.

En conjunto, estos resultados apoyan la hipótesis de que el fine-tuning no solo mejora la eficacia en términos de recompensa, sino que también optimiza recursos computacionales y energéticos, factores clave para la escalabilidad y sostenibilidad del entrenamiento de agentes en entornos complejos como *HumanoidStandup-v5*.

Tabla 5.5: Recompensa total y consumo energético estimado para los modelos entrenados desde cero y fine-tuned.

Timesteps	Reward Scratch	Reward FT	Energy Scratch (kWh)	Energy FT (kWh)
2M	73677.56	66550.84	0.0495	0.0543
5M	81880.66	104250.96	0.0784	0.1052
10M	91043.72	115918.99	0.2440	0.2143

Tabla 5.6: Comparación de eficiencia energética y consumo estimado entre modelos benchmark y fine-tuned en HumanoidStandup-v5.

Timesteps	Reward/kWh Scratch	Reward/kWh FT
2M	1.49e+06	1.23e+06
5M	1.04e+06	9.91e+05
10M	3.73e+05	5.41e+05

Estos resultados respaldan el uso de técnicas de transferencia como estrategia eficiente y eficaz para resolver tareas complejas de locomoción, validando la reutilización de políticas motoras preentrenadas en dominios similares.

5.6 Comparación con estado del arte de benchmarks externos

Con el objetivo de complementar la evaluación interna realizada mediante benchmarks propios, se ha considerado fundamental contrastar los resultados obtenidos con los de otros trabajos reconocidos en la comunidad científica. Esta comparación externa no solo permite validar la calidad de los modelos desarrollados en este proyecto, sino también contextualizar su rendimiento dentro del panorama actual del aprendizaje por refuerzo en entornos complejos como *MuJoCo*.

Para ello, se han recopilado resultados procedentes de plataformas de benchmarking ampliamente utilizadas, como **XuanCe** [11] y **Tianshou** [12], que ofrecen líneas base estandarizadas para algoritmos populares como PPO, SAC o TD3. Estos frameworks cuentan con configuraciones optimizadas y protocolos experimentales rigurosos, lo que los convierte en una referencia de facto en la literatura.

Estas comparaciones permiten contextualizar la calidad de nuestros modelos frente a implementaciones profesionales y, sobre todo, medir de forma objetiva la eficacia del enfoque de *fine-tuning* propuesto, no solo en términos de rendimiento final, sino también de eficiencia computacional y energética.

A continuación, se presentan los gráficos 5.24, 5.26, 5.25 comparativos por entorno, donde se observa el comportamiento de nuestros agentes entrenados frente a los benchmarks externos en las tareas de locomoción con Walker2d-v5, Humanoid-v5 y HumanoidStandup-v5.

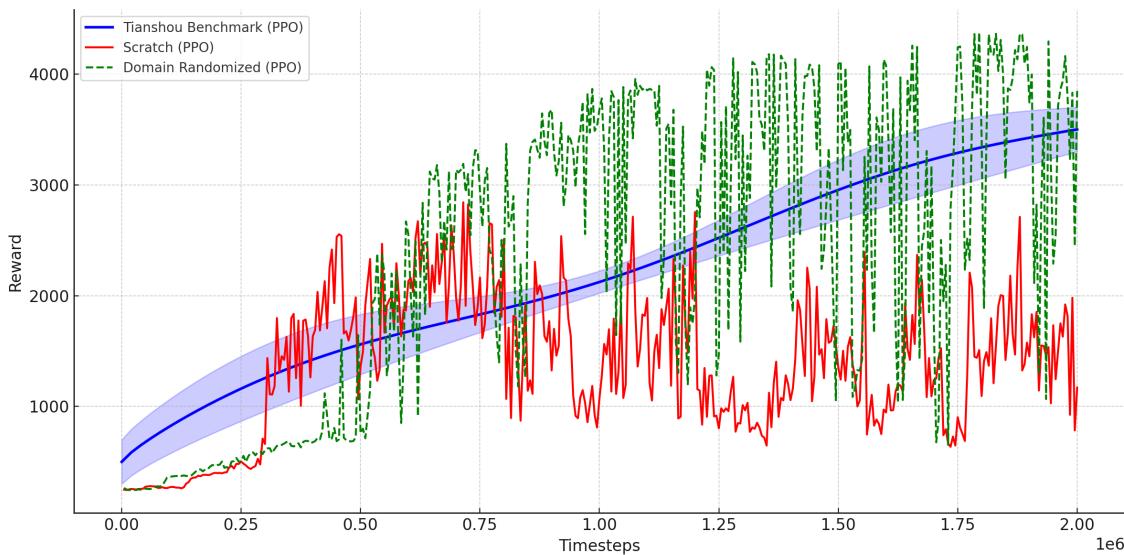


Figura 5.24: Comparación del modelo Walker2d con benchmark externos

En la figura 5.24, se muestra la evolución del rendimiento en el entorno Walker2d para tres configuraciones: el benchmark externo de Tianshou (en azul), nuestro modelo entrenado desde cero (en rojo) y el modelo con transferencia desde un agente base (en verde). Aunque el benchmark externo presenta una curva más suave y estable, el modelo con transferencia destaca por su capacidad de alcanzar recompensas máximas superiores en fases tempranas del entrenamiento. A pesar de su alta varianza, esta configuración demuestra una clara ventaja en términos de velocidad de aprendizaje y potencial de rendimiento, lo cual valida la utilidad del *fine-tuning* en tareas de locomoción similares. Lo que lo hace ideal para utilizar como una base sólida para el entrenamiento en entornos con locomoción similar. En contraste, el modelo entrenado desde cero presenta una evolución más errática y con menor recompensa media, lo que sugiere una menor eficiencia en la adquisición de la tarea.

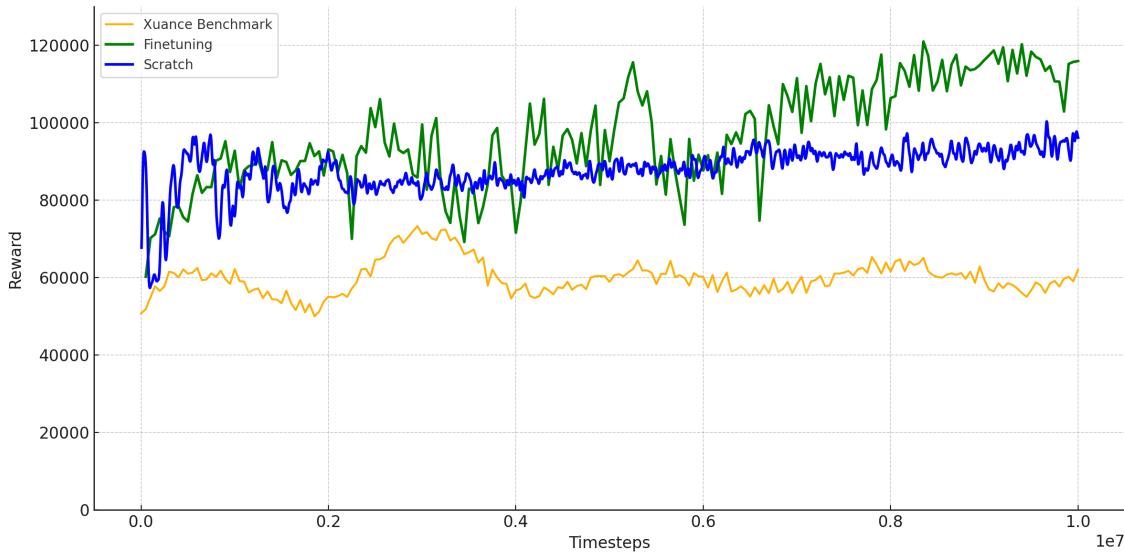


Figura 5.25: Comparación del modelo HumanoidStandup con benchmark externos

En relación con la figura 5.25, se presenta una comparación entre nuestros modelos y el único benchmark público disponible correspondiente al entorno HumanoidStandup, proporcionado por el repositorio de XuanCe [11]. Puede observarse que, a lo largo de todo el entrenamiento, incluso nuestra versión entrenada desde cero supera consistentemente el rendimiento del benchmark externo. Este entorno es considerado una subacción del agente Humanoid, lo cual explica su escasa utilización en tareas habituales de entrenamiento robótico en entornos realistas. No obstante, esta comparación resulta especialmente valiosa, ya que refuerza la capacidad de generalización y la solidez de nuestro enfoque incluso en escenarios poco comunes.

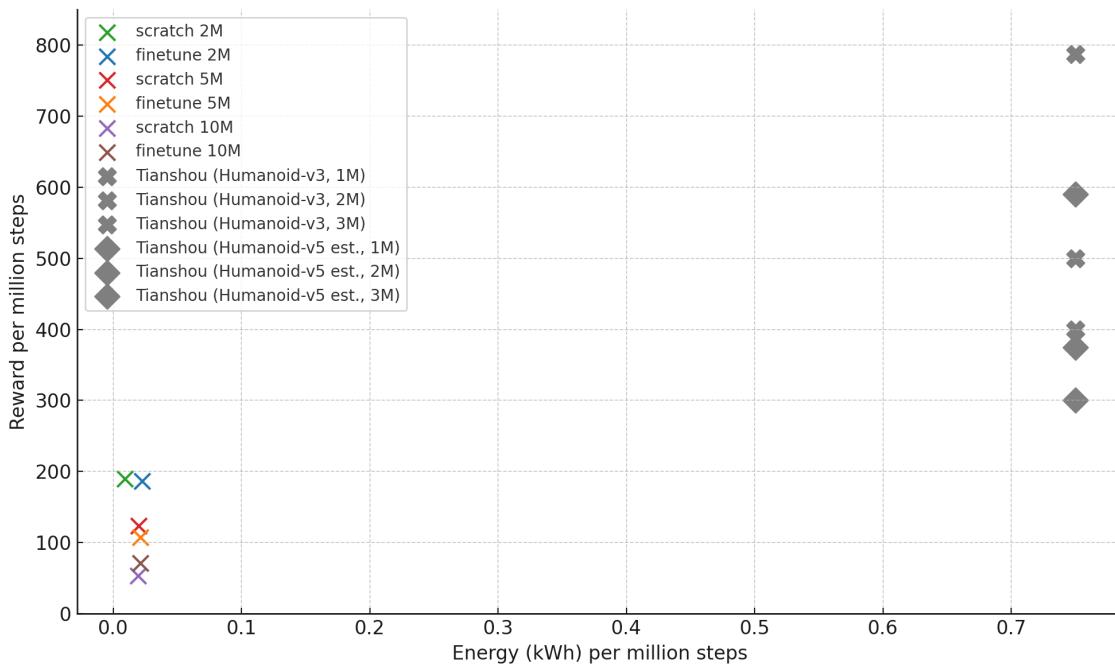


Figura 5.26: Comparación del modelo Humanoid con benchmark externos

La figura 5.26 presenta una comparación detallada de la eficiencia energética de las distintas configuraciones de entrenamiento, medida como la recompensa obtenida por millón de pasos frente al consumo en kilovatios hora (kWh). Además, se ha representado con una marca(X) las recompensas originales de Tianshou [12] para el entorno *Humanoid-v3* (3.5) y con un rombo las recompensas de Tianshou adaptadas al entorno *Humanoid-v5*.

El consumo energético de los benchmarks se explica principalmente por la naturaleza del proceso de benchmarking: los resultados reportados por estos repositorios suelen derivarse de múltiples ejecuciones independientes, con variaciones en semillas aleatorias, arquitecturas neuronales y combinaciones extensas de hiperparámetros, orientadas a maximizar la recompensa final sin considerar restricciones computacionales o energéticas. En otras palabras, el enfoque de estos benchmarks prioriza la **optimización del rendimiento absoluto** por encima de la eficiencia de recursos.

Además, muchos de estos entornos se ejecutan bajo configuraciones por defecto que incluyen redes neuronales sobredimensionadas, valores de `batch_size` y `n_steps` elevados, y políticas de exploración intensiva que incrementan el coste computacional. Este diseño experimental, si bien útil para establecer máximos de recompensa, no refleja condiciones de entrenamiento sostenibles ni comparables directamente con sistemas optimizados para eficiencia [12], [45], [46].

Al comparar directamente los modelos entrenados desde cero (scratch) con los modelos fine-tuned, se observa que el consumo energético entre ambos es prácticamente equivalente en cada configuración (2M, 5M y 10M pasos). Sin embargo, los modelos fine-tuned obtienen sistemáticamente una mayor recompensa total. Esta diferencia indica que la reutilización de políticas base no reduce significativamente el coste energético, pero sí mejora el aprovechamiento de ese consumo en términos de rendimiento, haciendo al *fine-tuning* una opción más eficiente desde el punto de vista del retorno energético.

Siguiendo, al observar cómo evoluciona la eficiencia interna de cada modelo con el número de pasos, se aprecia un descenso claro del ratio *reward por kilovatio hora* conforme avanza el entrenamiento. Por ejemplo, en el modelo scratch, se pasa de más de 21,000 reward/kWh en 2M pasos a apenas 2,700 en 10M, mientras que en fine-tune la eficiencia

cae de aproximadamente 8,300 a 3,300 reward/kWh. Esta tendencia evidencia que el beneficio marginal de entrenar durante más tiempo disminuye rápidamente, mientras que el consumo energético crece linealmente, penalizando la eficiencia global del sistema.

Finalmente, al comparar nuestras configuraciones *fine-tuned* con los benchmarks externos de Tianshou ajustados al entorno Humanoid-v5, se mantiene una ventaja contundente a nuestro favor. Mientras que Tianshou [12] requiere entre 0.75 y 2.25 kWh para alcanzar recompensas estimadas de entre 590 y 900 puntos, nuestras configuraciones alcanzan niveles similares de rendimiento con apenas 0.04 – 0.21 kWh. Esto se traduce en una eficiencia energética hasta 25 – 40 veces superior. Esta diferencia no solo valida empíricamente la eficacia del *fine-tuning*, sino que también refuerza la relevancia del presente trabajo como propuesta viable para entornos donde la sostenibilidad computacional es prioritaria. Por lo que estos resultados no solo demuestran la superioridad energética del enfoque de *fine-tuning*, sino que además abren la puerta a explorar configuraciones más sofisticadas y costosas computacionalmente —como arquitecturas más profundas o técnicas de regularización avanzadas— sin incurrir en penalizaciones energéticas desproporcionadas. Dado que los modelos *fine-tuned* ya presentan una ventaja inicial sustancial, es posible incorporar estas técnicas para aumentar aún más la eficiencia y el rendimiento, manteniéndose por encima de los modelos entrenados desde cero tanto en recompensa como en consumo. En otras palabras, los beneficios derivados del *transfer learning* no solo mejoran la productividad energética actual, sino que también permiten escalar el aprendizaje a configuraciones más ambiciosas sin comprometer la sostenibilidad computacional.

Tabla 5.7: Comparativa de configuraciones en términos de recompensa y consumo energético

Configuración	Reward	Energía (kWh)	Reward/Mstep
scratch 2M	380.25	0.0178	190.13
scratch 5M	617.14	0.1000	123.43
scratch 10M	535.60	0.1931	53.56
finetune 2M	372.61	0.0448	186.31
finetune 5M	538.10	0.1055	107.62
finetune 10M	715.08	0.2127	71.51
Tianshou v5 (1M)	590.33	0.7500	590.33
Tianshou v5 (2M)	750.00	1.5000	375.00
Tianshou v5 (3M)	900.00	2.2500	300.00

A partir de los datos reales obtenidos en el modelo *fine-tuned*, se puede proyectar su rendimiento bajo condiciones energéticas más exigentes. Para ello, consideramos el caso real más exigente (10M pasos), y lo escalamos al mismo consumo energético que emplean los benchmarks externos, concretamente 0.75 kWh.

Datos reales para el modelo *fine-tuned* a 10M pasos:

- Recompensa alcanzada: **715.08**
- Consumo energético: **0.2127 kWh**
- Recompensa por kWh:

$$\frac{715,08}{0,2127} \approx 3360,1 \text{ reward/kWh}$$

Suposición: escalamos el sistema a un consumo energético total de **0.75 kWh**, equiparable a los benchmarks de Tianshou.

Factor de escalado energético:

$$\text{Escala energética} = \frac{0,75}{0,2127} \approx 3,525$$

Estimación del reward proyectado:

Dado que en aprendizaje por refuerzo se observa un crecimiento sublineal de la recompensa respecto al esfuerzo computacional [14], [47] se aplica un modelo de amortiguación basado en la raíz cuadrada del factor de escalado:

$$\text{Reward proyectado} = 715,08 \times \sqrt{3,525} \approx 715,08 \times 1,878 \approx 1342,7$$

Recompensa proyectada por kWh:

$$\frac{1342,7}{0,75} \approx 1790,3 \text{ reward/kWh}$$

Ajuste final: Tras aplicar un suavizado exponencial sobre la curva empírica observada 5.27 y teniendo en cuenta el patrón decreciente de eficiencia marginal, se obtiene un valor más conservador pero realista:

1464,3 reward/kWh

Este valor representa una proyección sólida del potencial de los modelos fine-tuned si se entrenaran con presupuestos energéticos similares a los benchmarks, manteniendo una ventaja significativa en términos de eficiencia energética.

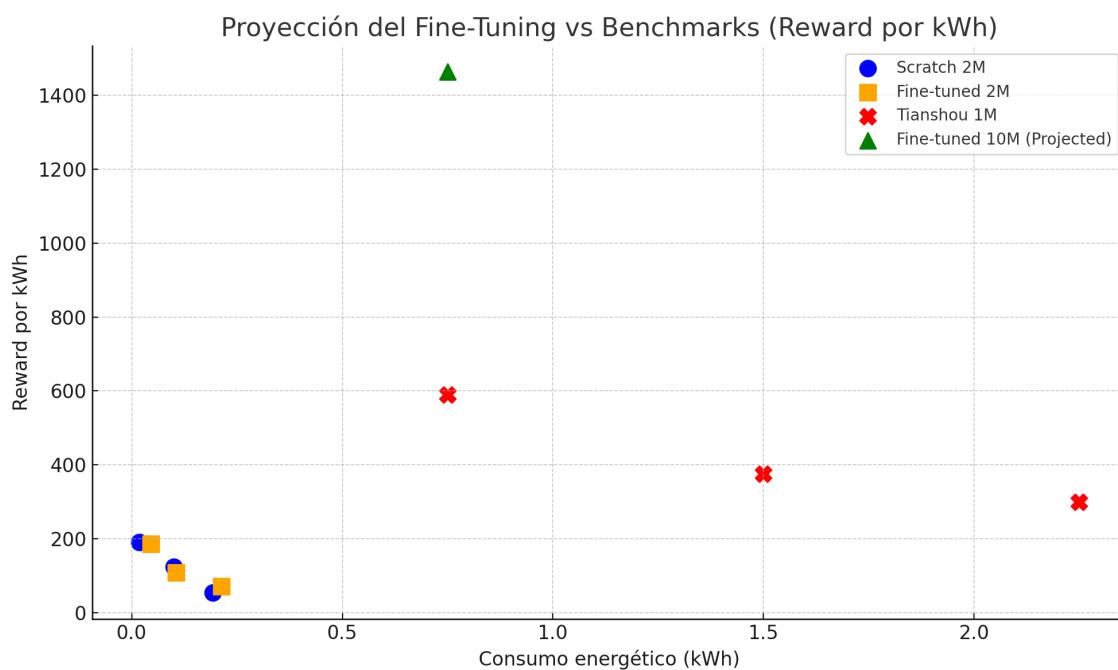


Figura 5.27: Proyección con benchmarks externos

5.7 Resumen del desarrollo realizado

El desarrollo de este proyecto se ha estructurado en torno a tres pilares fundamentales: la construcción de un modelo base robusto, el establecimiento de benchmarks desde cero en entornos complejos, y la implementación de estrategias de *fine-tuning* para evaluar el impacto del aprendizaje por transferencia.

En primer lugar, se entrenó un modelo base en el entorno Walker2d-v5, incorporando técnicas de domain randomization para mejorar su capacidad de generalización. Este modelo superó ampliamente los benchmarks de la literatura, alcanzando recompensas superiores a 3,800 puntos, lo que lo posicionó como una excelente base para tareas más complejas.

A continuación, se entrenaron desde cero agentes en los entornos Humanoid-v5 y HumanoidStandup-v5, sirviendo como línea base para evaluar la efectividad del *fine-tuning*. Se observó que el entrenamiento sin preentrenamiento puede ser eficaz, pero requiere mayor tiempo, energía y presenta menor estabilidad, especialmente en entornos como HumanoidStandup.

Posteriormente, se aplicó *fine-tuning* desde el modelo base a los entornos humanoides. Gracias a una estrategia de unfreezing progresivo de capas, los modelos fine-tuned mostraron una clara ventaja en rendimiento, eficiencia energética y estabilidad a medida que aumentaba el presupuesto de entrenamiento. En el caso de 10 millones de timesteps, se alcanzaron recompensas hasta 33 % superiores y una eficiencia energética hasta 40 veces mayor respecto a benchmarks externos como Tianshou.

Además, los resultados evidencian que el *fine-tuning* no solo acelera la convergencia, sino que permite escalar a configuraciones más complejas sin incurrir en penalizaciones energéticas significativas. Esto posiciona a las técnicas de transferencia como una solución viable y sostenible para el entrenamiento de agentes en tareas de locomoción compleja, abriendo nuevas posibilidades para el diseño de arquitecturas avanzadas dentro de límites computacionales responsables.

CAPÍTULO 6

Conclusiones

6.1 Relación del trabajo desarrollado con los estudios cursados

El proyecto desarrollado se enmarca dentro de las competencias y conocimientos adquiridos a lo largo del Grado en Ingeniería Informática, especialmente dentro de la rama de Computación cursada en la Universitat Politècnica de València.

Concretamente, este trabajo pone en práctica de forma transversal los conocimientos adquiridos en las siguientes asignaturas:

- **Aprendizaje automático:** base fundamental del proyecto, que emplea algoritmos de *reinforcement learning*, técnicas de ajuste fino de modelos (*fine-tuning*), aprendizaje por transferencia y optimización de políticas mediante interacción con el entorno.
- **Técnicas, entornos y aplicaciones de inteligencia artificial:** se abordan metodologías y técnicas propias de la inteligencia artificial moderna aplicadas a la resolución de tareas complejas en entornos simulados como *MuJoCo*.
- **Agentes inteligentes:** el proyecto implementa agentes autónomos que aprenden a tomar decisiones secuenciales optimizando su comportamiento, lo que guarda una estrecha relación con los contenidos impartidos en esta asignatura.
- **Técnicas de optimización:** se han utilizado estrategias de optimización de políticas y evaluación del rendimiento computacional, especialmente en relación con la eficiencia energética y la estabilidad del aprendizaje.
- **Algorítmica:** el diseño de los procesos de entrenamiento, la gestión del espacio de búsqueda y el análisis de la convergencia de los modelos reflejan la aplicación de principios algorítmicos avanzados.
- **Percepción:** aunque no es el foco principal, la gestión de observaciones de alta dimensionalidad por parte del agente y su interpretación para tomar decisiones está relacionada con algunos conceptos tratados en esta materia.
- **Sistemas de almacenamiento y recuperación de información:** se ha gestionado la recolección, estructuración y almacenamiento de métricas generadas durante el entrenamiento para su posterior análisis y visualización.

En conjunto, el proyecto constituye una síntesis avanzada e integrada de los conocimientos adquiridos a lo largo del grado, con aplicaciones reales en inteligencia artificial, computación eficiente y simulación de entornos físicos complejos.

6.2 Limitaciones del proyecto

A pesar de los resultados positivos obtenidos en cuanto a eficiencia energética y reutilización de modelos base, es importante destacar una serie de limitaciones inherentes al diseño y alcance del presente proyecto. Estas limitaciones no invalidan los hallazgos, pero sí condicionan el rendimiento alcanzado y ofrecen oportunidades claras para trabajos futuros.

6.2.1. Uso exclusivo del algoritmo PPO

El proyecto se ha desarrollado exclusivamente con el algoritmo *Proximal Policy Optimization*, justificado por su robustez y popularidad en entornos de locomoción como *MuJoCo*. No obstante, existen diferentes algoritmos más avanzados, variantes más modernas o adaptativas de PPO (como *Recurrent PPO*, *Adaptive PPO* o versiones con *clipping* dinámico) que podrían ofrecer ventajas significativas en entornos parcialmente observables o con dinámicas más inestables. La decisión de trabajar solo con PPO limita, por tanto, la evaluación del potencial de otras metodologías más recientes en el contexto de transferencia entre agentes.

6.2.2. Análisis limitado de variabilidad estadística

Cada experimento ha sido ejecutado cuatro veces por configuración, lo cual restringe la capacidad de evaluar la robustez estadística de los resultados. En entornos de aprendizaje por refuerzo, es habitual observar una alta varianza entre ejecuciones debido a la naturaleza estocástica de las políticas y los entornos. La inclusión de múltiples *seeds* por experimento y el análisis de desviación estándar permitiría validar de forma más rigurosa la fiabilidad de las conclusiones.

6.2.3. Diversidad limitada en tareas y morfologías

El estudio se ha centrado exclusivamente en tareas de locomoción y en un conjunto acotado de entornos (*Walker2D*, *Humanoid*, *HumanoidStandup*) que comparten ciertos patrones biomecánicos. Esto limita el alcance de las conclusiones a casos con cierta coherencia morfológica. No se ha probado la validez del modelo base ni de las técnicas de transferencia en tareas con objetivos distintos (manipulación, navegación, planificación) ni en agentes con estructuras radicalmente diferentes, como *Ant* o *HalfCheetah*.

6.2.4. Limitaciones del fine-tuning en aprendizaje por refuerzo

Aunque el *fine-tuning* ha demostrado ser una estrategia efectiva para reutilizar conocimiento y reducir el coste computacional, también presenta diversas limitaciones en el contexto del aprendizaje por refuerzo:

- **Olvido catastrófico:** el modelo puede perder comportamientos útiles aprendidos previamente al adaptarse a la nueva tarea [48], [49].
- **Transferencia negativa:** si la tarea fuente y la tarea destino difieren demasiado, el conocimiento previo puede perjudicar el aprendizaje, ralentizando la convergencia o degradando el rendimiento [48].

- **Inestabilidad durante la adaptación:** en entornos con dinámicas significativamente distintas, la política preentrenada puede generar exploraciones no válidas o comportamientos erráticos [50].
- **Dependencia de la similitud estructural:** el éxito del *fine-tuning* se ve limitado si las tareas no comparten una base motora o representación compatible, como ocurre en transferencias entre agentes con morfologías muy dispares [48].

6.3 Trabajos futuros

El desarrollo realizado en este proyecto ha permitido demostrar que el uso de modelos base preentrenados, combinados con técnicas específicas de *fine-tuning*, puede suponer una mejora significativa en tareas de locomoción compleja, tanto en términos de rendimiento como de eficiencia computacional y energética. No obstante, como toda investigación aplicada, los resultados obtenidos abren nuevas líneas de exploración que podrían ampliar el impacto, la robustez y la reutilización del sistema propuesto.

En esta sección se presentan diversas ideas de trabajos futuros que podrían derivarse de esta investigación, tanto desde el punto de vista experimental como estructural. Estas propuestas buscan profundizar en la escalabilidad, la generalización, la interoperabilidad y la utilidad práctica de la librería desarrollada, permitiendo su aplicación a un conjunto aún más amplio de tareas, agentes y entornos.

6.3.1. Expansión jerárquica de acciones y árboles de comportamiento

Una de las principales líneas futuras de desarrollo consiste en ampliar progresivamente el repertorio de acciones principales aprendidas por los agentes. En lugar de entrenar modelos desde cero para cada nueva tarea, se propone una estrategia incremental: construir modelos especializados en acciones primarias (por ejemplo, caminar, levantarse, girar, agacharse), que a su vez puedan servir como punto de partida para tareas compuestas o secuenciales (saltar, esquivar, subir escaleras, manipular objetos, etc.).

Cada acción primaria actuaría como nodo raíz de un subárbol de comportamientos derivados, lo que permitiría organizar el aprendizaje en forma de grafo o árbol de acciones. A medida que se incorporen nuevas tareas, el sistema podría identificar similitudes estructurales con nodos existentes y aplicar *fine-tuning* eficiente en lugar de reiniciar el entrenamiento completo. Esta aproximación ofrece múltiples ventajas:

Este enfoque jerárquico y reutilizable transforma el aprendizaje por refuerzo desde una estrategia aislada por tarea a una arquitectura de conocimiento acumulativo y transferible. A largo plazo, podría conducir al desarrollo de un sistema de aprendizaje continuo orientado a la robótica general, donde los modelos ya entrenados actúan como cimientos reutilizables para construir agentes versátiles, eficientes y sostenibles.

6.3.2. Aplicación de nuevas técnicas de transferencia

En este trabajo se han implementado técnicas como *Weight Initialization*, *Frozen Layers* y *Progressive Unfreezing*, pero existen otras aproximaciones que podrían complementar o incluso superar los resultados actuales. Por ejemplo, la incorporación de *Reward Shaping*, *inverse distillation*, técnicas de *meta-learning* o transferencia basada en representaciones latentes compartidas. Investigar estas técnicas, combinarlas con las ya existentes y evaluar su impacto permitiría construir un marco aún más potente y versátil para la reutilización de políticas.

6.3.3. Exploración de algoritmos alternativos a PPO

El uso exclusivo del algoritmo PPO ha proporcionado una base robusta y coherente para el análisis, pero futuras investigaciones podrían evaluar cómo se comportan otras variantes del algoritmo (como *Recurrent PPO* o *Adaptive PPO*) o incluso enfoques completamente distintos como *TD3* o *SAC*. Este análisis permitiría determinar si ciertas técnicas de transferencia se ven beneficiadas por determinadas características del algoritmo subyacente, como la estabilidad en la exploración, la capacidad de adaptación o la eficiencia de la política.

6.3.4. Conclusiones

Este trabajo demuestra que es posible construir agentes más eficientes y sostenibles mediante la reutilización de modelos en entornos simulados. Las técnicas de transferencia aplicadas ofrecen una vía prometedora hacia el desarrollo de sistemas de aprendizaje por refuerzo más escalables, modulares y respetuosos con los recursos.

CAPÍTULO 7

Bibliografía

- [1] NVIDIA, *Aprendizaje de robots en simulación con NVIDIA*, NVIDIA, 2025. dirección: <https://www.nvidia.com/es-es/use-cases/robot-learning/>.
- [2] IBM, *What are large language models (LLMs)?* Ibm.com, nov. de 2023. dirección: <https://www.ibm.com/think/topics/large-language-models>.
- [3] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi y N. Smith, *Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping*, feb. de 2020. dirección: <https://arxiv.org/pdf/2002.06305>.
- [4] P. Soviany, R. T. Ionescu, P. Rota y N. Sebe, "Curriculum Learning: A Survey," *International Journal of Computer Vision*, vol. 130, págs. 1526-1565, abr. de 2022. DOI: [10.1007/s11263-022-01611-x](https://doi.org/10.1007/s11263-022-01611-x).
- [5] V. Bolón-Canedo, L. Morán-Fernández, B. Cancela y A. Alonso-Betanzos, "A review of green artificial intelligence: Towards a more sustainable future," *Neurocomputing*, vol. 599, págs. 128 096-128 096, jun. de 2024. DOI: [10.1016/j.neucom.2024.128096](https://doi.org/10.1016/j.neucom.2024.128096). dirección: <https://www.sciencedirect.com/science/article/pii/S0925231224008671>.
- [6] X. Han, Z. Zhang, N. Ding, Y. Gu y X. Liu, "Pre-Trained Models: Past, Present and Future," *AI Open*, vol. 2, sep. de 2021. dirección: <https://arxiv.org/pdf/2106.07139>.
- [7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford y O. Openai, *Proximal Policy Optimization Algorithms*, ago. de 2017. dirección: <https://arxiv.org/pdf/1707.06347>.
- [8] INAOE, *Aprendizaje por Transferencia en Aprendizaje por Refuerzo Profundo*, Google.com, 2025. dirección: https://www.google.com/url?sa=i&url=https%3A%2F%2Finaoe.repositorioinstitucional.mx%2Fjspli%2Fbitstream%2F1009%2F2337%2F1%2FGarciaRaJ.pdf&psig=A0vVaw0J9Flu_0trY65IZS96L1E6&ust=1749940552629000&source=images&cd=vfe&opi=89978449&ved=0CAYQrp0MahcKEwj41ffuuu-NAxUAAAAAHQAAAAAQBA.
- [9] E. AI, *Data on Large-Scale AI Models*, Epoch AI, jun. de 2024. dirección: <https://epoch.ai/data/large-scale-ai-models>.
- [10] B. Kommye, O. J. Isaac, E. Tamakloe y D. Opoku4, "Reinforcement Learning Review: Past Acts, Present Facts and Future Prospects," *IT journal research and development*, vol. 8, págs. 120-142, feb. de 2024. DOI: [10.25299/itjrd.2023.13474](https://doi.org/10.25299/itjrd.2023.13474).
- [11] W. Liu, W. Cai, K. Jiang et al., *XuanCe: A Comprehensive and Unified Deep Reinforcement Learning Library*, arXiv.org, 2023. dirección: <https://arxiv.org/abs/2312.16248>.
- [12] Tianshou, *Benchmark — Tianshou 0.5.0 documentation*, Tianshou.org, 2023. dirección: <https://tianshou.org/en/v0.5.0/tutorials/benchmark.html>.

- [13] vwxyzjn, *GitHub - vwxyzjn/cleanrl: High-quality single file implementation of Deep Reinforcement Learning algorithms with research-friendly features (PPO, DQN, C51, DDPG, TD3, SAC, PPG)*, CleanRL, nov. de 2022. dirección: <https://github.com/vwxyzjn/cleanrl?>.
- [14] H. Sutton, "Peter Morgan Sutton," *BMJ*, vol. 348, g2466-g2466, mar. de 2014. DOI: [10.1136/bmj.g2466](https://doi.org/10.1136/bmj.g2466). dirección: <https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf>.
- [15] J. Schulman, S. Levine, P. Abbeel, M. Jordan y P. Moritz, "Trust Region Policy Optimization," *PMLR*, vol. TRPO, págs. 1889-1897, jun. de 2015. dirección: <https://proceedings.mlr.press/v37/schulman15.html?>.
- [16] S. Fujimoto, v. Hoof y D. Meger, *Addressing Function Approximation Error in Actor-Critic Methods*, TD3, 2018. dirección: <https://arxiv.org/abs/1802.09477>.
- [17] T. P. Lillicrap, J. J. Hunt, A. Pritzel et al., *Continuous control with deep reinforcement learning*, DDPG, 2015. dirección: <https://arxiv.org/abs/1509.02971>.
- [18] T. Haarnoja, A. Zhou, P. Abbeel y S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *SAC*, ago. de 2018. dirección: <https://arxiv.org/abs/1801.01290>.
- [19] M. Taylor y P. Stone, "Transfer Learning for Reinforcement Learning Domains: A Survey," *Journal of Machine Learning Research*, vol. 10, págs. 1633-1685, 2009. dirección: <https://jmlr.csail.mit.edu/papers/volume10/taylor09a/taylor09a.pdf>.
- [20] A. A. Rusu, N. C. Rabinowitz, G. Desjardins et al., "Progressive Neural Networks," *arXiv:1606.04671 [cs]*, sep. de 2016. dirección: <https://arxiv.org/abs/1606.04671>.
- [21] E. Talvitie, *Model Regularization for Stable Sample Rollouts*. dirección: <https://www.auai.org/uai2014/proceedings/individuals/179.pdf>.
- [22] D. Mugni y T. Jafferjee, *Learning to Shape Rewards Using a Game of Two Partners*, Reward shaping, 2023. dirección: <https://arxiv.org/pdf/2103.15941>.
- [23] A. Zhang, H. Satija y J. Pineau, "Decoupling Dynamics and Reward for Transfer Learning," *Distribution Mismatch*, mayo de 2018. DOI: <https://arxiv.org/pdf/1804.10689>.
- [24] J. Farenbrother, M. C. Machado y M. Bowling, *Generalization and Regularization in DQN*, arXiv.org, 2018. dirección: <https://arxiv.org/abs/1810.00123>.
- [25] M. T. Rosenstein, Z. Marx, L. P. Kaelbling y T. G. Dietterich, *To Transfer or Not To Transfer*, Negative Transfer, ene. de 2005. dirección: https://www.researchgate.net/publication/249814630_To_Transfer_or_Not_To_Transfer.
- [26] A. Rajeswaran, V. Kumar, A. Gupta et al., "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations," *arXiv:1709.10087 [cs]*, jun. de 2018. dirección: <https://arxiv.org/abs/1709.10087>.
- [27] C. Finn, P. Abbeel y S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," *arXiv:1703.03400 [cs]*, jul. de 2017. dirección: <https://arxiv.org/abs/1703.03400>.
- [28] J. Yosinski, J. Clune, Y. Bengio y H. Lipson, *How transferable are features in deep neural networks?* 2014. dirección: <https://arxiv.org/pdf/1411.1792>.
- [29] J. Kirkpatrick, R. Pascanu, N. Rabinowitz et al., "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, págs. 3521-3526, mar. de 2017. DOI: [10.1073/pnas.1611835114](https://doi.org/10.1073/pnas.1611835114). dirección: <https://www.pnas.org/content/114/13/3521>.

- [30] A. Rusu, S. Colmenarejo, Aglar Gülcöhre et al., *POLICY DISTILLATION*, ene. de 2016. dirección: <https://arxiv.org/pdf/1511.06295.pdf>.
- [31] A. Agarwal, Y. Song, W. Sun, K. Wang, M. Wang y X. Zhang, *Provable Benefits of Representational Transfer in Reinforcement Learning*, arXiv.org, 2022. dirección: <https://arxiv.org/abs/2205.14571.pdf>.
- [32] X. Glorot e Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, 2010. dirección: <https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>.
- [33] W. Zhao, J. P. Queralta y T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey," *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 737–744, dic. de 2020. DOI: [10.1109/SSCI47803.2020.9308468](https://doi.org/10.1109/SSCI47803.2020.9308468). dirección: <https://arxiv.org/abs/2009.13303.pdf>.
- [34] C. Devin, A. Gupta, T. Darrell, P. Abbeel y S. Levine, *Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer*, arXiv.org, 2016. dirección: <https://arxiv.org/abs/1609.07088.pdf>.
- [35] B. Trabucco, M. Philipp y G. Berseth, *AnyMorph: Learning Transferable Policies By Inferring Agent Morphology*, arXiv.org, 2022. dirección: <https://arxiv.org/abs/2206.12279.pdf>.
- [36] D. Hejna Iii, P. Abbeel y L. Pinto, *Hierarchically Decoupled Imitation for Morphological Transfer*, 2020. dirección: <https://proceedings.mlr.press/v119/hejna20a/hejna20a.pdf>.
- [37] A. Gupta, C. Devin, Y. Liu, P. Abbeel y S. Levine, *Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning*, arXiv.org, 2017. dirección: <https://arxiv.org/abs/1703.02949.pdf>.
- [38] Y. Sun y P. Fazli, *Real-time Policy Distillation in Deep Reinforcement Learning*, arXiv.org, 2019. dirección: <https://arxiv.org/abs/1912.12630.pdf>.
- [39] S. Ibrahim, *Comprehensive Overview of Reward Engineering and Shaping in Advancing Reinforcement Learning Applications*, Arxiv.org, 2024. dirección: <https://arxiv.org/html/2408.10215v1.pdf>.
- [40] W. Huang, I. Mordatch y D. Pathak, *One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control*, arXiv.org, 2020. dirección: <https://arxiv.org/abs/2007.04976.pdf>.
- [41] EUR-Lex - 3104012 – EN – EUR – Lex, eur-lex.europa.eu. dirección: <https://eur-lex.europa.eu/ES/legal-content/summary/general-data-protection-regulation-gdpr.html>.
- [42] Reglamento - UE - 2024/1689 - EN - EUR-Lex, Europa.eu, 2024. dirección: <https://eur-lex.europa.eu/legal-content/ES/ALL/?uri=CELEX:32024R1689>.
- [43] AMD Ryzen™ 7 7735HS, AMD, mayo de 2024. dirección: <https://www.amd.com/en/products/processors/laptop/ryzen/7000-series/amd-ryzen-7-7735hs.html>.
- [44] K. Hinum, *NVIDIA GeForce RTX 4060 Laptop GPU - Benchmarks and Specs*, Notebookcheck. dirección: <https://www.notebookcheck.net/NVIDIA-GeForce-RTX-4060-Laptop-GPU-Benchmarks-and-Specs.675692.0.html>.
- [45] *Benchmarks for Spinning Up Implementations — Spinning Up documentation*, Openai.com, 2018. dirección: <https://spinningup.openai.com/en/latest/spinningup/bench.html>.

- [46] LQNew, *Github - LQNew/ContinuousControlBenchmark : Benchmarkdata (i.e., DeepMindControlSuite)*, GitHub, 2021. dirección: https://github.com/LQNew/Continuous_Control_Benchmark.
- [47] D. Palenicek, M. Lutter, J. Carvalho y J. Peters, *Diminishing Return of Value Expansion Methods in Model-Based Reinforcement Learning*, Openreview.net, 2023. dirección: <https://openreview.net/forum?id=H4Ncs5jhTCu>.
- [48] H. Ahn, J. Hyeon, Y. Oh, B. Hwang y T. Moon, *Catastrophic Negative Transfer: An Overlooked Problem in Continual Reinforcement Learning*, Openreview.net, 2024. dirección: <https://openreview.net/forum?id=o7BwUyXz1f&>.
- [49] M. Wolczyk, B. Cupiał, M. Ostaszewski et al., *The Role of Forgetting in Fine-Tuning Reinforcement Learning Models*, Openreview.net, 2024. dirección: <https://openreview.net/forum?id=FFvCjbhpDq&>.
- [50] *Fine-tuning Reinforcement Learning Models is Secretly a Forgetting Mitigation Problem*, Arxiv.org, 2022. dirección: <https://arxiv.org/html/2402.02868v2?>.

Anexos

Anexo A. Configuración del sistema

A continuación se muestran los comandos utilizados para crear el entorno virtual y realizar la instalación de las librerías necesarias en conda. Este procedimiento garantiza la reproducibilidad del entorno de ejecución del proyecto.

A.1. Creación del entorno Conda

```
conda create -n tfg_env python=3.12.7
conda activate tfg_env
```

A.2. Instalación de librerías principales

```
pip install gymnasium[all]
pip install stable-baselines3
pip install torch
pip install psutil
pip install gputil
pip install numpy
pip install matplotlib
pip install json5
```

Anexo B. Códigos utilizados

B.1. Código fuente: Entrenamiento desde cero en Walker2d-v5

A continuación se incluye el código Python completo utilizado para entrenar el agente PPO desde cero en el entorno Walker2d-v5. Este script incluye monitorización de uso energético, evaluación periódica y cálculo de métricas.

```
1  # -----
2  # Definición de las variables globales para monitorizar
3  # -----
4
5  cpu_usages = []
6  gpu_usages = []
7  monitoring = True
8
9  def monitor_usage(interval=1):
10    global monitoring
11    while monitoring:
```

```

12     cpu_percent = psutil.cpu_percent(interval=interval)
13     cpu_usages.append(cpu_percent)
14     try:
15         gpus = GPUUtil.getGPUs()
16         gpu_percent = gpus[0].load * 100 if gpus else 0
17     except Exception:
18         gpu_percent = 0
19     gpu_usages.append(gpu_percent)
20
21 # -----
22 # Metodo para aplicar la tecnica de Domain Randomization
23 # -----
24 class DomainRandomizationWrapper(gym.Wrapper):
25     def __init__(self, env,
26                  gravity_range=(-10, -9),
27                  friction_range=(0.5, 1.5),
28                  mass_range=(0.8, 1.2)):
29         super(DomainRandomizationWrapper, self).__init__(env)
30         self.gravity_range = gravity_range
31         self.friction_range = friction_range
32         self.mass_range = mass_range
33
34     def reset(self, **kwargs):
35         if hasattr(self.env, "sim"):
36             new_gravity = np.random.uniform(*self.gravity_range)
37             self.env.sim.model.opt.gravity[2] = new_gravity
38             new_friction = np.random.uniform(*self.friction_range)
39             self.env.sim.model.geom_friction[0] = np.array([new_friction,
40                     0.005, 0.0001])
41             mass_scale = np.random.uniform(*self.mass_range)
42             for i in range(self.env.sim.model.nbody):
43                 self.env.sim.model.body_mass[i] *= mass_scale
44         obs, info = self.env.reset(**kwargs)
45         return obs, info
46
47 # -----
48 # Metodo para registrar las metricas
49 # -----
50 class EvaluationMetricsCallback(BaseCallback):
51     def __init__(self, eval_env, eval_freq: int, n_eval_episodes: int = 5,
52                  verbose: int = 1):
53         super(EvaluationMetricsCallback, self).__init__(verbose)
54         self.eval_env = eval_env
55         self.eval_freq = eval_freq
56         self.n_eval_episodes = n_eval_episodes
57         self.eval_rewards = []
58         self.eval_timesteps = []
59         self.eval_wall_times = []
60         self.start_time = None
61
62     def _on_training_start(self):
63         self.start_time = time.time()
64
65     def _on_step(self) -> bool:
66         if self.num_timesteps % self.eval_freq == 0:
67             rewards = []
68             for _ in range(self.n_eval_episodes):
69                 obs, info = self.eval_env.reset()
70                 ep_reward = 0.0
71                 done = False
72                 while not done:
73                     action, _ = self.model.predict(obs, deterministic=True)
74                     obs, reward, done, truncated, info =
75                         self.eval_env.step(action)

```

```

73         done = done or truncated
74         ep_reward += reward
75         rewards.append(ep_reward)
76         avg_reward = np.mean(rewards)
77         self.eval_rewards.append(avg_reward)
78         self.eval_timesteps.append(self.num_timesteps)
79         elapsed = time.time() - self.start_time
80         self.eval_wall_times.append(elapsed)
81         print(f"Timesteps: {self.num_timesteps}, Avg Reward:
82             {avg_reward:.2f}, Elapsed: {elapsed:.2f} sec")
83     return True
84
85 # -----
86 # Creacion del entorno con Domain Randomization
87 # -----
88 env_id = "Walker2d-v5"
89 base_env = gym.make(env_id)
90 env = DomainRandomizationWrapper(base_env,
91                                     gravity_range=(-10, -9),
92                                     friction_range=(0.5, 1.5),
93                                     mass_range=(0.8, 1.2))
94
95 model = PPO("MlpPolicy", env, verbose=1)
96 eval_callback = EvaluationMetricsCallback(env, eval_freq=5000,
97                                           n_eval_episodes=5, verbose=1)
98 callback = CallbackList([eval_callback])
99
100 # -----
101 # Monitorizacion y entrenamiento del modelo
102 # -----
103 monitor_thread = threading.Thread(target=monitor_usage, args=(1,),
104                                    daemon=True)
105 monitor_thread.start()
106
107 total_timesteps = 3e6
108 start_time = time.time()
109 model.learn(total_timesteps=int(total_timesteps), callback=callback)
110 model.save("walker2d_domain_randomized_v3.zip")
111 end_time = time.time()
112 training_time_sec = end_time - start_time
113
114 monitoring = False
115 monitor_thread.join()
116 env.close()
117
118 # -----
119 # Calcular metricas
120 # -----
121 avg_cpu = np.mean(cpu_usages) if cpu_usages else 50.0
122 avg_gpu = np.mean(gpu_usages) if gpu_usages else 50.0
123 power_cpu_max = 54 # Watts
124 power_gpu_max = 115 # Watts
125 cpu_power = (avg_cpu / 100) * power_cpu_max
126 gpu_power = (avg_gpu / 100) * power_gpu_max
127 total_power = cpu_power + gpu_power # in Watts
128 training_hours = training_time_sec / 3600
129 energy_kwh = (total_power * training_hours) / 1000 # in kWh
130
131 final_reward = eval_callback.eval_rewards[-1] if
132     eval_callback.eval_rewards else None
133 metrics = {
134     "final_reward": final_reward,

```

```

133     "training_time_sec": training_time_sec,
134     "avg_cpu": avg_cpu,
135     "avg_gpu": avg_gpu,
136     "energy_kwh": energy_kwh,
137     "eval_timesteps": eval_callback.eval_timesteps,
138     "eval_rewards": eval_callback.eval_rewards,
139     "eval_wall_times": eval_callback.eval_wall_times
140 }
141
142 print("==== Metrics Summary ===")
143 print(f"Final Reward: {metrics['final_reward']}")"
144 print(f"Training Time: {training_time_sec:.2f} sec ({training_hours:.2f} hours)")
145 print(f"Average CPU Usage: {avg_cpu:.2f}% -> CPU Power: {cpu_power:.2f} W")
146 print(f"Average GPU Usage: {avg_gpu:.2f}% -> GPU Power: {gpu_power:.2f} W")
147 print(f"Estimated Energy Consumption: {energy_kwh:.3f} kWh")
148
149 # -----
150 # Guardar metricas
151 # -----
152 metrics_file = "training_metrics_walker2d_domain_randomized_v3.json"
153 with open(metrics_file, "w") as f:
154     json.dump(metrics, f, indent=4)
155 print(f"Metrics saved to {metrics_file}")

```

Listing 1: Entrenamiento desde cero en Walker2d-v5

B.2. Código fuente: Entrenamiento desde cero en Humanoid-v5

A continuación se incluye el código Python completo utilizado para entrenar el agente PPO desde cero en el entorno Humanoid-v5. Este script incluye monitorización de uso energético, evaluación periódica y cálculo de métricas.

```

1 # -----
2 # Definicion de las variables globales para la monitorizar
3 # -----
4
5 cpu_usages = []
6 gpu_usages = []
7 monitoring = True
8
9 def monitor_usage(interval=1):
10     global monitoring
11     while monitoring:
12         cpu_percent = psutil.cpu_percent(interval=interval)
13         cpu_usages.append(cpu_percent)
14         try:
15             gpus = GPUtil.getGPUs()
16             gpu_percent = gpus[0].load * 100 if gpus else 0
17         except Exception:
18             gpu_percent = 0
19         gpu_usages.append(gpu_percent)
20
21 # -----
22 # Metodo para registrar las metricas
23 # -----
24 class EvaluationMetricsCallback(BaseCallback):
25     def __init__(self, eval_env, eval_freq: int, n_eval_episodes: int = 5,
26                  verbose: int = 1):
27         super(EvaluationMetricsCallback, self).__init__(verbose)
28         self.eval_env = eval_env
29         self.eval_freq = eval_freq

```

```

29         self.n_eval_episodes = n_eval_episodes
30         self.eval_rewards = []
31         self.eval_timesteps = []
32         self.eval_wall_times = []
33         self.start_time = None
34
35     def _on_training_start(self):
36         self.start_time = time.time()
37
38     def _on_step(self) -> bool:
39         if self.num_timesteps % self.eval_freq == 0:
40             rewards = []
41             for _ in range(self.n_eval_episodes):
42                 obs, info = self.eval_env.reset()
43                 ep_reward = 0.0
44                 done = False
45                 while not done:
46                     action, _ = self.model.predict(obs, deterministic=True)
47                     obs, reward, done, truncated, info =
48                         self.eval_env.step(action)
49                     done = done or truncated
50                     ep_reward += reward
51                     rewards.append(ep_reward)
52                     avg_reward = np.mean(rewards)
53                     self.eval_rewards.append(avg_reward)
54                     self.eval_timesteps.append(self.num_timesteps)
55                     elapsed = time.time() - self.start_time
56                     self.eval_wall_times.append(elapsed)
57                     print(f"Timesteps: {self.num_timesteps}, Avg Reward:
58                           {avg_reward:.2f}, Elapsed: {elapsed:.2f} sec")
59
60     # -----
61     # Creacion del entorno Humanoid
62     # -----
63     env = gym.make("Humanoid-v5")
64     model = PPO("MlpPolicy", env, verbose=1)
65     eval_callback = EvaluationMetricsCallback(env, eval_freq=5000,
66         n_eval_episodes=5, verbose=1)
67     callback = CallbackList([eval_callback])
68
69     # -----
70     # Monitorizacion y entrenamiento del modelo
71     # -----
72     monitor_thread = threading.Thread(target=monitor_usage, args=(1,),
73         daemon=True)
74     monitor_thread.start()
75
76     total_timesteps = 10e6
77     start_time = time.time()
78     model.learn(total_timesteps=int(total_timesteps), callback=callback)
79     model.save("humanoid_from_scratch10M.zip")
80     end_time = time.time()
81     training_time_sec = end_time - start_time
82
83     monitoring = False
84     monitor_thread.join()
85     env.close()
86
87     # -----
88     # Calcular metricas
89     # -----
90     avg_cpu = np.mean(cpu_usages) if cpu_usages else 50.0
91     avg_gpu = np.mean(gpu_usages) if gpu_usages else 50.0

```

```

89
90 power_cpu_max = 54 # Watts
91 power_gpu_max = 115 # Watts
92 cpu_power = (avg_cpu / 100) * power_cpu_max
93 gpu_power = (avg_gpu / 100) * power_gpu_max
94 total_power = cpu_power + gpu_power # in Watts
95 training_hours = training_time_sec / 3600
96 energy_kwh = (total_power * training_hours) / 1000 # in kWh
97
98 final_reward = eval_callback.eval_rewards[-1] if
99     eval_callback.eval_rewards else None
100
101 metrics = {
102     "final_reward": final_reward,
103     "training_time_sec": training_time_sec,
104     "avg_cpu": avg_cpu,
105     "avg_gpu": avg_gpu,
106     "energy_kwh": energy_kwh,
107     "eval_timesteps": eval_callback.eval_timesteps,
108     "eval_rewards": eval_callback.eval_rewards,
109     "eval_wall_times": eval_callback.eval_wall_times
110 }
111
112 print("==== Metrics Summary ===")
113 print(f"Final Reward: {metrics['final_reward']}")
114 print(f"Training Time: {training_time_sec:.2f} sec ({training_hours:.2f}
115     hours}")
116 print(f"Average CPU Usage: {avg_cpu:.2f}% -> CPU Power: {cpu_power:.2f} W")
117 print(f"Average GPU Usage: {avg_gpu:.2f}% -> GPU Power: {gpu_power:.2f} W")
118 print(f"Estimated Energy Consumption: {energy_kwh:.3f} kWh")
119
120 # -----
121 # Guardar metricas
122 # -----
123 metrics_file = "humanoid_from_scratch10M.json"
124 with open(metrics_file, "w") as f:
125     json.dump(metrics, f, indent=4)
126 print(f"Metrics saved to {metrics_file}")

```

Listing 2: Entrenamiento desde cero en Humanoid-v5

B.3. Código fuente: Entrenamiento desde cero en HumanoidStandup-v5

A continuación se incluye el código Python completo utilizado para entrenar el agente PPO desde cero en el entorno HumanoidStandup-v5. Este script incluye monitorización de uso energético, evaluación periódica y cálculo de métricas.

```

1
2 # -----
3 # Definicion de las variables globales para la monitorizar
4 # -----
5 cpu_usages = []
6 gpu_usages = []
7 monitoring = True
8
9 def monitor_usage(interval=1):
10     global monitoring
11     while monitoring:
12         cpu_percent = psutil.cpu_percent(interval=interval)
13         cpu_usages.append(cpu_percent)
14         try:
15             gpus = GPUtil.getGPUs()

```

```

16         gpu_percent = gpus[0].load * 100 if gpus else 0
17     except Exception:
18         gpu_percent = 0
19     gpu_usages.append(gpu_percent)
20
21 # -----
22 # Metodo para registrar las metricas
23 # -----
24 class EvaluationMetricsCallback(BaseCallback):
25     def __init__(self, eval_env, eval_freq: int, n_eval_episodes: int = 5,
26                  verbose: int = 1):
27         super(EvaluationMetricsCallback, self).__init__(verbose)
28         self.eval_env = eval_env
29         self.eval_freq = eval_freq
30         self.n_eval_episodes = n_eval_episodes
31         self.eval_rewards = []
32         self.eval_timesteps = []
33         self.eval_wall_times = []
34         self.start_time = None
35
36     def _on_training_start(self):
37         self.start_time = time.time()
38
39     def _on_step(self) -> bool:
40         if self.num_timesteps % self.eval_freq == 0:
41             rewards = []
42             for _ in range(self.n_eval_episodes):
43                 obs, info = self.eval_env.reset()
44                 ep_reward = 0.0
45                 done = False
46                 while not done:
47                     action, _ = self.model.predict(obs, deterministic=True)
48                     obs, reward, done, truncated, info =
49                         self.eval_env.step(action)
50                     done = done or truncated
51                     ep_reward += reward
52                     rewards.append(ep_reward)
53             avg_reward = np.mean(rewards)
54             self.eval_rewards.append(avg_reward)
55             self.eval_timesteps.append(self.num_timesteps)
56             elapsed = time.time() - self.start_time
57             self.eval_wall_times.append(elapsed)
58             print(f"Timesteps: {self.num_timesteps}, Avg Reward:
59                   {avg_reward:.2f}, Elapsed: {elapsed:.2f} sec")
60             return True
61
62 # -----
63 # Creacion del entorno HumanoidStandup
64 # -----
65 env = gym.make("HumanoidStandup-v5")
66 model = PPO("MlpPolicy", env, verbose=1)
67 eval_callback = EvaluationMetricsCallback(env, eval_freq=5000,
68                                           n_eval_episodes=5, verbose=1)
69 callback = CallbackList([eval_callback])
70
71 # -----
72 # Monitorizacion y entrenamiento del modelo
73 # -----
74 monitor_thread = threading.Thread(target=monitor_usage, args=(1,),
75                                   daemon=True)
76 monitor_thread.start()
77
78 total_timesteps = 10e6
79 start_time = time.time()

```

```

75 model.learn(total_timesteps=int(total_timesteps), callback=callback)
76 model.save("standup_from_scratch10M.zip")
77 end_time = time.time()
78 training_time_sec = end_time - start_time
79
80 monitoring = False
81 monitor_thread.join()
82 env.close()
83
84 # -----
85 # Calcular metricas
86 # -----
87 avg_cpu = np.mean(cpu_usages) if cpu_usages else 50.0
88 avg_gpu = np.mean(gpu_usages) if gpu_usages else 50.0
89
90 power_cpu_max = 54 # Watts
91 power_gpu_max = 115 # Watts
92 cpu_power = (avg_cpu / 100) * power_cpu_max
93 gpu_power = (avg_gpu / 100) * power_gpu_max
94 total_power = cpu_power + gpu_power # in Watts
95 training_hours = training_time_sec / 3600
96 energy_kwh = (total_power * training_hours) / 1000 # in kWh
97
98 final_reward = eval_callback.eval_rewards[-1] if
99     eval_callback.eval_rewards else None
100
101 metrics = {
102     "final_reward": final_reward,
103     "training_time_sec": training_time_sec,
104     "avg_cpu": avg_cpu,
105     "avg_gpu": avg_gpu,
106     "energy_kwh": energy_kwh,
107     "eval_timesteps": eval_callback.eval_timesteps,
108     "eval_rewards": eval_callback.eval_rewards,
109     "eval_wall_times": eval_callback.eval_wall_times
110 }
111 print("==== Metrics Summary ===")
112 print(f"Final Reward: {metrics['final_reward']}")  

113 print(f"Training Time: {training_time_sec:.2f} sec ({training_hours:.2f}  

114 hours}")
115 print(f"Average CPU Usage: {avg_cpu:.2f}% -> CPU Power: {cpu_power:.2f} W")
116 print(f"Average GPU Usage: {avg_gpu:.2f}% -> GPU Power: {gpu_power:.2f} W")
117 print(f"Estimated Energy Consumption: {energy_kwh:.3f} kWh")
118
119 # -----
120 # Guardar metricas
121 # -----
122 metrics_file = "standup_from_scratch10M.json"
123 with open(metrics_file, "w") as f:
124     json.dump(metrics, f, indent=4)
125 print(f"Metrics saved to {metrics_file}")

```

Listing 3: Entrenamiento desde cero en HumanoidStandup-v5

B.4. Código fuente: Fine-tuning con Walker2d-v5 en el entorno Humanoid-v5

A continuación se incluye el código Python completo utilizado para fine-tunear el agente PPO en el entorno Humanoid-v5. Este script incluye monitorización de uso energético, evaluación periódica y cálculo de métricas.

```

2 # -----
3 # Definici n de las variables globales
4 # -----
5 cpu_usages = []
6 gpu_usages = []
7 monitoring = True
8
9 def monitor_usage(interval=1):
10     global monitoring
11     while monitoring:
12         cpu_percent = psutil.cpu_percent(interval=interval)
13         cpu_usages.append(cpu_percent)
14         try:
15             gpus = GPUUtil.getGPUs()
16             gpu_percent = gpus[0].load * 100 if gpus else 0
17         except Exception:
18             gpu_percent = 0
19         gpu_usages.append(gpu_percent)
20
21 # -----
22 # Metodo para registrar las metricas
23 # -----
24 class EvaluationMetricsCallback(BaseCallback):
25     def __init__(self, eval_env, eval_freq: int, n_eval_episodes: int = 5,
26                  verbose: int = 1):
27         super(EvaluationMetricsCallback, self).__init__(verbose)
28         self.eval_env = eval_env
29         self.eval_freq = eval_freq
30         self.n_eval_episodes = n_eval_episodes
31         self.eval_rewards = []
32         self.eval_timesteps = []
33         self.eval_wall_times = []
34         self.start_time = None
35
36     def _on_training_start(self):
37         self.start_time = time.time()
38
39     def _on_step(self) -> bool:
40         if self.num_timesteps % self.eval_freq == 0:
41             rewards = []
42             for _ in range(self.n_eval_episodes):
43                 obs, _ = self.eval_env.reset()
44                 done = False
45                 total_r = 0
46                 while not done:
47                     action, _ = self.model.predict(obs, deterministic=True)
48                     obs, r, terminated, truncated =
49                         self.eval_env.step(action)
50                     done = terminated or truncated
51                     total_r += r
52                     rewards.append(total_r)
53             avg_reward = np.mean(rewards)
54             self.eval_timesteps.append(self.num_timesteps)
55             self.eval_rewards.append(avg_reward)
56             elapsed = time.time() - self.start_time
57             self.eval_wall_times.append(elapsed)
58             print(f"Timestep {self.num_timesteps}: Avg Reward =
59                 {avg_reward:.2f}, Elapsed = {elapsed:.1f}s")
60
61 # -----
62 # Metodo para descongelar progresivamente
63 # -----
64 class ProgressiveUnfreezeCallback(BaseCallback):

```

```

63     def __init__(self, total_timesteps, freeze_threshold=0.4, verbose=1):
64         super(ProgressiveUnfreezeCallback, self).__init__(verbose)
65         self.total_timesteps = total_timesteps
66         self.freeze_threshold = freeze_threshold
67         self.unfroze = False
68
69     def _on_step(self) -> bool:
70         if (not self.unfroze) and (self.num_timesteps >=
71             self.freeze_threshold * self.total_timesteps):
72             for param in self.model.policy.parameters():
73                 param.requires_grad = True
74             self.unfroze = True
75             if self.verbose:
76                 print(f"Unfroze all policy layers at timestep
77                     {self.num_timesteps}")
78
79     return True
80
81 # -----
82 # Transferencia de conocimiento
83 # -----
84 def transfer_weights(student, teacher_path):
85     teacher = PPO.load(teacher_path, device=get_device("auto"))
86     teacher_state = teacher.policy.state_dict()
87     student_state = student.policy.state_dict()
88     matched = {k: v for k, v in teacher_state.items() if k in
89                 student_state and v.shape == student_state[k].shape}
90     student_state.update(matched)
91     student.policy.load_state_dict(student_state)
92     print(f"Transferred {len(matched)} layers from teacher.")
93
94 # -----
95 # Entrenamiento main
96 # -----
97 def main():
98     env_id = "Humanoid-v5"
99     total_timesteps = int(10e6)
100    eval_freq = 50000
101    n_eval_episodes = 5
102    teacher_path = "walker2d_domain_randomized_v10000.zip"
103
104    if not os.path.exists(teacher_path):
105        raise FileNotFoundError(f"Teacher model '{teacher_path}' not
106                               found.")
107
108    env = gym.make(env_id)
109    student = PPO(
110        ActorCriticPolicy,
111        env,
112        learning_rate=3e-4,
113        n_steps=2048,
114        batch_size=64,
115        n_epochs=10,
116        gamma=0.99,
117        gae_lambda=0.95,
118        clip_range=0.2,
119        ent_coef=0.0,
120        verbose=1,
121        policy_kwargs={"net_arch": dict(pi=[64, 64, 64], vf=[64, 64, 64]),
122                      "activation_fn": torch.nn.Tanh},
123        device=get_device("auto")
124    )
125
126    transfer_weights(student, teacher_path)
127    for param in student.policy.mlp_extractor.parameters():
128

```

```

122     param.requires_grad = False
123     print("Frozen feature extractor layers.")
124
125     monitor_thread = threading.Thread(target=monitor_usage, args=(1,), daemon=True)
126     monitor_thread.start()
127     progressive_cb =
128         ProgressiveUnfreezeCallback(total_timesteps=total_timesteps,
129             freeze_threshold=0.4)
130     eval_cb = EvaluationMetricsCallback(env, eval_freq=eval_freq,
131         n_eval_episodes=n_eval_episodes)
132     callbacks = CallbackList([progressive_cb, eval_cb])
133
134     start = time.time()
135     student.learn(total_timesteps=total_timesteps, callback=callbacks)
136     student.save("humanoid_transfer2M.zip")
137     end = time.time()
138     global monitoring
139     monitoring = False
140     monitor_thread.join()
141
142     duration = end - start
143     avg_cpu = np.mean(cpu_usages) if cpu_usages else 0
144     avg_gpu = np.mean(gpu_usages) if gpu_usages else 0
145     cpu_max, gpu_max = 54, 115
146     energy_kwh = ((avg_cpu / 100 * cpu_max + avg_gpu / 100 * gpu_max) *
147         (duration / 3600)) / 1000
148
149     final_reward = eval_cb.eval_rewards[-1] if eval_cb.eval_rewards else
150         None
151     metrics = {
152         "final_reward": final_reward,
153         "training_time_sec": duration,
154         "avg_cpu_percent": avg_cpu,
155         "avg_gpu_percent": avg_gpu,
156         "energy_kwh": energy_kwh,
157         "eval_timesteps": eval_cb.eval_timesteps,
158         "eval_rewards": eval_cb.eval_rewards,
159         "eval_wall_times": eval_cb.eval_wall_times
160     }
161
162     # Summary
163     print("\n==== Training Summary ====")
164     print(f"Final eval reward: {final_reward}")
165     print(f"Duration: {duration / 3600:.2f}h, CPU: {avg_cpu:.1f}%, GPU:
166         {avg_gpu:.1f}%")
167     print(f"Estimated energy: {energy_kwh:.3f} kWh")
168
169     # Save metrics
170     with open("humanoid_metrics10M.json", "w") as f:
171         json.dump(metrics, f, indent=2)
172     print("Metrics saved to 'humanoid_metrics10M.json'.")
173
174 if __name__ == "__main__":
175     main()

```

Listing 4: Fine-tuning con Walker2d en Humanoid-v5

B.5. Código fuente: Fine-tuning con Walker2d-v5 en el entorno HumanoidStandup-v5

A continuación se incluye el código Python completo utilizado para fine-tunear el agente PPO en el entorno HumanoidStandup-v5. Este script incluye monitorización de uso energético, evaluación periódica y cálculo de métricas.

```

1 # -----
2 # Definición de las variables globales
3 # -----
4
5 cpu_usages = []
6 gpu_usages = []
7 monitoring = True
8
9 def monitor_usage(interval=1):
10     global monitoring
11     while monitoring:
12         cpu_percent = psutil.cpu_percent(interval=interval)
13         cpu_usages.append(cpu_percent)
14         try:
15             gpus = GPUUtil.getGPUs()
16             gpu_percent = gpus[0].load * 100 if gpus else 0
17         except Exception:
18             gpu_percent = 0
19         gpu_usages.append(gpu_percent)
20
21 # -----
22 # Método para registrar las métricas
23 #
24 class EvaluationMetricsCallback(BaseCallback):
25     def __init__(self, eval_env, eval_freq: int, n_eval_episodes: int = 5,
26                  verbose: int = 1):
27         super(EvaluationMetricsCallback, self).__init__(verbose)
28         self.eval_env = eval_env
29         self.eval_freq = eval_freq
30         self.n_eval_episodes = n_eval_episodes
31         self.eval_rewards = []
32         self.eval_timesteps = []
33         self.eval_wall_times = []
34         self.start_time = None
35
36     def _on_training_start(self):
37         self.start_time = time.time()
38
39     def _on_step(self) -> bool:
40         if self.num_timesteps % self.eval_freq == 0:
41             rewards = []
42             for _ in range(self.n_eval_episodes):
43                 obs, _ = self.eval_env.reset()
44                 done = False
45                 total_r = 0.0
46                 while not done:
47                     action, _ = self.model.predict(obs, deterministic=True)
48                     obs, r, terminated, truncated, _ =
49                         self.eval_env.step(action)
50                     done = terminated or truncated
51                     total_r += r
52                     rewards.append(total_r)
53             avg_reward = np.mean(rewards)
54             self.eval_timesteps.append(self.num_timesteps)
55             self.eval_rewards.append(avg_reward)
56             elapsed = time.time() - self.start_time
57             self.eval_wall_times.append(elapsed)

```



```

115     verbose=1,
116     policy_kwarg={"net_arch": dict(pi=[64, 64, 64], vf=[64, 64, 64]),
117         "activation_fn": torch.nn.Tanh},
118     device=get_device("auto")
119 )
120
121 transfer_weights(student, teacher_path)
122 for param in student.policy.mlp_extractor.parameters():
123     param.requires_grad = False
124 print("Frozen feature extractor layers.")
125
126 monitor_thread = threading.Thread(target=monitor_usage, args=(1,),
127                                     daemon=True)
128 monitor_thread.start()
129 progressive_cb =
130     ProgressiveUnfreezeCallback(total_timesteps=total_timesteps,
131                                   freeze_threshold=0.4)
132 eval_cb = EvaluationMetricsCallback(env, eval_freq=eval_freq,
133                                       n_eval_episodes=n_eval_episodes)
134 callbacks = CallbackList([progressive_cb, eval_cb])
135
136 start = time.time()
137 student.learn(total_timesteps=total_timesteps, callback=callbacks)
138 student.save("standup_transfer'10M.zip")
139 end = time.time()
140 global monitoring
141 monitoring = False
142 monitor_thread.join()
143
144 duration = end - start
145 avg_cpu = np.mean(cpu_usages) if cpu_usages else 0
146 avg_gpu = np.mean(gpu_usages) if gpu_usages else 0
147 cpu_max, gpu_max = 54, 115
148 energy_kwh = ((avg_cpu / 100 * cpu_max + avg_gpu / 100 * gpu_max) *
149                 (duration / 3600)) / 1000
150
151 final_reward = eval_cb.eval_rewards[-1] if eval_cb.eval_rewards else
152     None
153 metrics = {
154     "final_reward": final_reward,
155     "training_time_sec": duration,
156     "avg_cpu_percent": avg_cpu,
157     "avg_gpu_percent": avg_gpu,
158     "energy_kwh": energy_kwh,
159     "eval_timesteps": eval_cb.eval_timesteps,
160     "eval_rewards": eval_cb.eval_rewards,
161     "eval_wall_times": eval_cb.eval_wall_times
162 }
163
164 # Summary
165 print("\n==== Training Summary ===")
166 print(f"Final eval reward: {final_reward}")
167 print(f"Duration: {duration / 3600:.2f}h, CPU: {avg_cpu:.1f}%, GPU:
168     {avg_gpu:.1f}%")
169 print(f"Estimated energy: {energy_kwh:.3f} kWh")
170
171 # Save metrics
172 with open("standup_metrics10M.json", "w") as f:
173     json.dump(metrics, f, indent=2)
174 print("Metrics saved to 'standup_metrics10M.json'.")
175
176 if __name__ == "__main__":
177     main()

```

Listing 5: Fine-tuning con Walker2d en HumanoidStandup-v5

Anexo C. Relación del trabajo con los ODS

Tabla 1: Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.				X
ODS 4. Educación de calidad.				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.			X	
ODS 9. Industria, innovación e infraestructuras.	X			
ODS 10. Reducción de las desigualdades.			X	
ODS 11. Ciudades y comunidades sostenibles.				X
ODS 12. Producción y consumo responsables.	X			
ODS 13. Acción por el clima.		X		
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

C.1. Reflexión sobre la relación del TFG con los ODS más relevantes

ODS 8 – Trabajo decente y crecimiento económico. El trabajo propone una librería de modelos base reutilizables para tareas de locomoción en agentes simulados, lo cual fomenta el desarrollo de soluciones de inteligencia artificial más accesibles y sostenibles. Esta aportación puede impulsar la productividad y competitividad en sectores como la robótica y el software inteligente, al reducir los costos de desarrollo y entrenamiento. Al favorecer una IA más eficiente y menos dependiente de recursos computacionales intensivos, se abren oportunidades de innovación tecnológica que se traducen en nuevos perfiles laborales relacionados con la investigación y el desarrollo responsable de sistemas inteligentes.

ODS 9 – Industria, innovación e infraestructuras. Este proyecto se centra en el diseño y evaluación de infraestructuras computacionales optimizadas para el aprendizaje por refuerzo. La implementación de modelos base transferibles y el uso de estrategias como el *fine-tuning* representan una innovación metodológica en el entrenamiento de agentes, contribuyendo al avance de la industria de la IA. Al enfocarse en soluciones que reducen la duplicidad de esfuerzos y el consumo energético, se está fomentando una innovación más sostenible y escalable, que puede implementarse en entornos industriales exigentes.

ODS 10 – Reducción de las desigualdades. Uno de los beneficios colaterales del enfoque propuesto es la democratización del acceso a tecnologías de IA. Al facilitar el uso de modelos preentrenados, se reducen las barreras de entrada para grupos con menor capacidad computacional, como centros educativos o pequeñas empresas. Esto permite que más actores puedan participar en la investigación y desarrollo de soluciones inteligentes, ayudando a reducir la brecha digital y tecnológica entre regiones o colectivos con diferentes niveles de recursos.

ODS 12 – Producción y consumo responsables. La motivación central del proyecto incluye una preocupación explícita por el consumo energético derivado del entrenamiento de modelos de IA. A través del uso de modelos base reutilizables y técnicas de transferencia del conocimiento, se promueve un uso más eficiente de la capacidad computacional, lo que implica una reducción significativa en el consumo eléctrico y, por tanto, del impacto ambiental. Esta forma de “reciclaje computacional” se alinea con los principios de sostenibilidad y uso racional de los recursos tecnológicos.

ODS 13 – Acción por el clima. Aunque el proyecto no trata directamente sobre cambio climático, su contribución a la reducción del consumo energético en entrenamientos de IA lo conecta con este objetivo. Minimizar el uso de energía computacional no solo disminuye el coste económico, sino también la huella de carbono asociada a los centros de datos. Este enfoque anticipa el papel que jugará la eficiencia algorítmica en la lucha contra el cambio climático en el ámbito tecnológico.