



GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

Estudio Comparativo y Desarrollo de Algoritmos de
Reward Learning y Goal Conditioned Learning:
Análisis y Aplicación

Autor: Adolfo Lozano Mendoza

Director: Lucía Güitta López

Co-Director: Álvaro Jesús López López

Madrid
Agosto de 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
Estudio Comparativo y Desarrollo de Algoritmos de Reward Learning y Goal Conditioned
Learning: Análisis y Aplicación en la ETS de Ingeniería - ICAI de la Universidad
Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido
tomada de otros documentos está debidamente referenciada.



Fdo.: Adolfo Lozano Mendoza

Fecha: 26/08/2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Lucía Güitta López

Fecha: 28/08/2025

A mis padres, Lourdes y Adolfo, por enseñarme todo, motivarme, inspirarme y apoyarme incondicionalmente.

A mi hermana Laura, por darme siempre los mejores consejos.

A mi hermana Lourdes, quien, aun siendo la más pequeña, es el mayor ejemplo de superación y lucha que tengo.

A Isabel, Juan Carlos y Álvaro, por cuidarme durante todos estos años en Madrid.

A Pablo, Antonio, Freire y Kike, por todas las risas, los bocatas de lomo en los descansos y los trabajos en grupo durante toda la carrera.

A María, por aguantarme diariamente estos últimos años de carrera y más durante el desarrollo de este trabajo.

ESTUDIO COMPARATIVO Y DESARROLLO DE ALGORITMOS DE REWARD LEARNING Y GOAL CONDITIONED LEARNING: ANÁLISIS Y APLICACIÓN

Autor: Lozano Mendoza, Adolfo.

Director: Güitta López, Lucía.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

Resumen

Este estudio compara tres algoritmos del campo de *Reward Learning* (MCE-IRL, AIRL y Text2Reward) con el objetivo de servir como guía de uso en función del interés del investigador. Los entornos seleccionados son *CliffWorld* para MCE-IRL y *Hopper* para AIRL y T2R.

MCE-IRL, que se basa en el concepto de máxima entropía, obtiene recompensas interpretables y cercanas a la reales usando trayectorias expertas, pero está limitado a entornos pequeños y tabulares. AIRL sigue la arquitectura de los problemas adversariales: un generador que trata de imitar las trayectorias de un experto y un discriminador que intenta identificar el origen de estas. AIRL logra una imitación estable del experto sin necesidad de un gran número de trayectorias. Sin embargo, la *reward net* extraída por el discriminador solo define la zona del espacio de las trayectorias que ha visto. Por lo tanto, sirve más como un identificador de buenas trayectorias que como una guía para entrenar un experto desde cero. T2R, usando *prompts* con descripciones *pythonic* del entorno y una instrucción del comportamiento deseado, logra aprovechar las capacidades de los modelos de lenguaje natural (LLMs) para generar una función de recompensa densa. El entrenamiento de un agente SAC con la función generada para *Hopper* por GPT-4o alcanza un rendimiento excepcional. Por último, la combinación de T2R + AIRL nos permitió conseguir un experto en el entorno modificado de *Hopper-Seals* con un rendimiento jamás reportado previamente.

Como marco complementario, se incluye una revisión de *Goal Conditioned Learning*, cuyo objetivo es aprender políticas condicionadas a metas, obteniendo agentes multitarea. Se sintetizan UFVA y HER y se describen en detalle Goal GAN y RIG.

Goal GAN, mediante una *Generative Adversarial Network* produce metas cuya dificultad crece de forma progresiva a medida que el agente mejora su rendimiento. Ha demostrado acelerar el aprendizaje con recompensas escasas al generar metas acordes al conocimiento del agente. RIG entrena políticas a partir de imágenes del entorno sin recompensas definidas sobre el estado. Codifica la observación y la meta con un VAE y define la recompensa como su distancia en el espacio latente. No necesita acceder al estado del entorno pues puede trabajar directamente sobre imágenes renderizadas obteniendo resultados cercanos al oráculo.

Palabras clave: Reward Learning; MCE-IRL; AIRL; Text2Reward; LLMs; Hopper; CliffWorld; Trayectorias, Goal Conditioned Learning; UFVA; HER; Goal GAN; RIG.

1. Introducción

El aprendizaje por refuerzo permite a un agente optimizar un comportamiento por medio de una función de recompensa que lo guíe. Sin embargo, diseñarla a mano es complejo y pequeñas imprecisiones inducen al agente hacia políticas no deseadas o subóptimas. Por ello, surge *Reward Learning*, que aglutina al conjunto de algoritmos que aprenden cuál es la función de recompensa que define un comportamiento deseado. La variedad de algoritmos es amplia: desde modelos probabilísticos como MaxEnt IRL [1] o B-IRL [2], enfoques basados en juegos adversariales como AIRL [3] o Guided Cost IRL [4], modelos de preferencias RLHF [5] o recientemente, modelos basados en el lenguaje natural.

Goal Conditioned Learning entrena políticas condicionadas a un objetivo utilizando un único agente multitarea. Destacan algoritmos como UFVA [6] que extienden el RL clásico con $V(s,g)$ y $Q(s,a,g)$; HER [7] como una técnica de *replay* para generar metas intermedias aprovechando trayectorias fallidas; modelos adversariales como Goal GAN [8] o basados en VAEs como RIG [9].

En este trabajo se comparan tres enfoques representativos del estado del arte y la evolución de este: Maximum Causal Entropy IRL [10], Adversarial IRL y Text2Reward [11]. Se implementan en un entorno tabular (*CliffWorld*) y otro continuo de mayor complejidad (*Hopper*). Además, se describen los algoritmos de RIG y Goal GAN en detalle, quedando como trabajo futuro una implementación y comparación en entornos similares.

2. Definición del proyecto

El objetivo general es comparar MCE-IRL, AIRL y T2R en entornos controlados, identificando fortalezas, límites y pautas de uso. Para GCL, se definen RIG y Goal GAN.

Algunos objetivos más específicos son: (i) evaluar la fidelidad de las recompensas aprendidas, analizando la jerarquía de trayectorias, (ii) la capacidad para guiar agentes desde cero, (iii) la sensibilidad de AIRL a la cantidad/calidad de demostraciones, (iv) la viabilidad computacional y (v) la interpretabilidad de las funciones de recompensa.

3. Descripción de algoritmos

MCE-IRL es la extensión causal de MaxEnt-IRL. Modela la función de recompensa con una red neuronal $R_\theta(s)$ y resuelve el MDP de horizonte finito y dinámica conocida mediante *Soft Bellman Backups*. Esto restringe su uso a entornos únicamente tabulares. A partir de Q y V obtenidos en la resolución, se induce una política y se calcula la medida de ocupación. Se compara con la ocupación experta siendo la diferencia la *loss* para actualizar los parámetros de la red.

AIRL consiste en un entrenamiento adversarial sobre las trayectorias expertas. El generador utiliza un algoritmo *on-policy* como PPO para aprender a imitar estas trayectorias. El discriminador consiste en una red neuronal potencial compuesta por g_θ que modela la función de recompensa general y un término de *shaping* h_ϕ para separarla de la dinámica y estabilizar el entrenamiento. En lugar de trayectorias completas, se parten en tripletas (s, a, s') que dotan de mayor estabilidad y permiten realizar mejoras locales en la política.

Text2Reward parte de un LLM y genera un código Python de la función de recompensa densa a partir de la descripción *pythonic* del entorno y la instrucción. Para Hopper es: “Control the Hopper to move in the forward direction.”

En MCE-IRL obtenemos los expertos entrenando con la función de recompensa original del entorno. Para AIRL tenemos 2 expertos que generan trayectorias. El primero es el mejor disponible en repositorios públicos que obtiene una recompensa de 2860 sobre la recompensa original. El segundo surge de una combinación con T2R, que nos ha permitido obtener un agente con recompensas de 3500 pero con mucha inestabilidad. Los modelos de lenguaje elegidos en Text2Reward son GPT-4o y DeepSeek-o1.

Goal GAN utiliza un GAN en el espacio de metas para proponer objetivos de dificultad intermedia según el nivel de conocimiento del agente. Se seleccionan metas del *buffer* y del generador. Estima la competencia del agente en cada meta midiendo la tasa de éxito. El discriminador se encarga de clasificar si son metas intermedias, entre 20%-80% de éxito, mientras que el generador trata de generar metas que el discriminador vaya a considerar intermedias. Con los datos recolectados se entrena un agente *off-policy*.

RIG aprende políticas condicionadas a objetivos desde imágenes mediante un VAE. El *autoencoder* manda las observaciones a un espacio latente compacto y define la recompensa como la distancia en el espacio entre la imagen actual y la objetivo. Las metas propuestas durante el entrenamiento pueden ser dadas manuales o imaginadas por el VAE muestreando del espacio latente. La política del agente *off-policy* se va actualizando con las transiciones $(z_t, a_t, z_{t+1}, z_g, r_t)$ y el VAE se va afinando para mejorar la representación latente.

4. Resultados

MCE-IRL sobre *CliffWorld* recupera ocupaciones casi idénticas a la experta. A medida que el tamaño del *grid* crece la función de recompensa aprendida se aleja de la real. Se ha evaluado en tamaños desde 4x7 a 80x80. Los tiempos de ejecución y uso de memoria son de 1 segundo y 727 MB para 4x7 y más de 4 horas y 2.7 GB para 80x80. Esto refleja un crecimiento exponencial en los recursos con el tamaño del entorno.

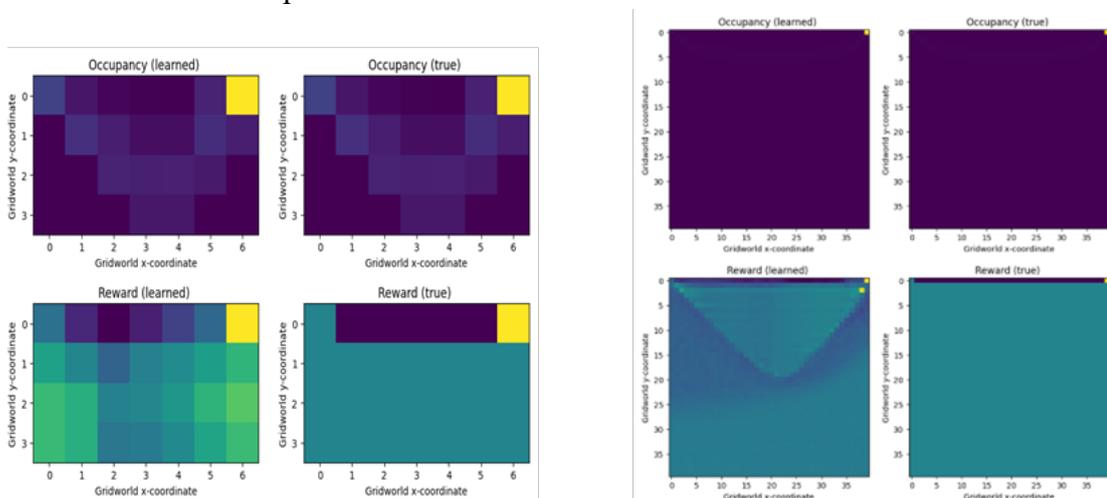


Ilustración a. Occupancy y Rewards aprendidas para grids de 4x7 y 80x80.

AIRL sobre el Experto de 2860 demostró una buena capacidad para imitación. Con 10 y 50 trayectorias se obtienen unos agentes que promedian 2600 ± 118 y 2506 ± 160 . Sin embargo, con 100 trayectorias el resultado cae a 2106 ± 23 . La explicación a este comportamiento se debe a dos motivos: un sobreajuste del discriminador en etapas tempranas y que el experto realmente no realiza una mecánica correcta debido a un *exploit* en la función de recompensa original (se arrastra).

Aislando trayectorias de 3500 del segundo experto se consigue una imitación excepcional. Tanto 10, 50 y 100 trayectorias superan los 3200 de recompensa. Destacamos el agente entrenado con 100 trayectorias que obtiene unas métricas de 3511 ± 11 . En el entorno de *Hopper* modificado por *Seals* no existe ningún agente que alcance unos valores similares.

El análisis de las funciones de recompensa aprendidas nos revela que son incapaces de guiar a un agente desde cero. No se ha conseguido que ninguno de estos agentes alcance un valor superior a 300, prácticamente de una política aleatoria. Las *reward nets* clasifican como buenas las trayectorias parecidas a las que vieron durante entrenamiento, pero no son capaces de generalizar otras zonas del espacio de observación.

Con T2R entrenamos un agente SAC y otro PPO con la función de recompensa generada por GPT-4o y por DeepSeek-o1. Para analizar su rendimiento los agentes han sido evaluados sobre la función de recompensa original del entorno. En todos los casos se obtienen valores cercanos o superiores a 3500. Las principales diferencias se encuentran en la estabilidad de los agentes, destacando SAC de GPT con 3542.6 ± 2.4 frente al resto.

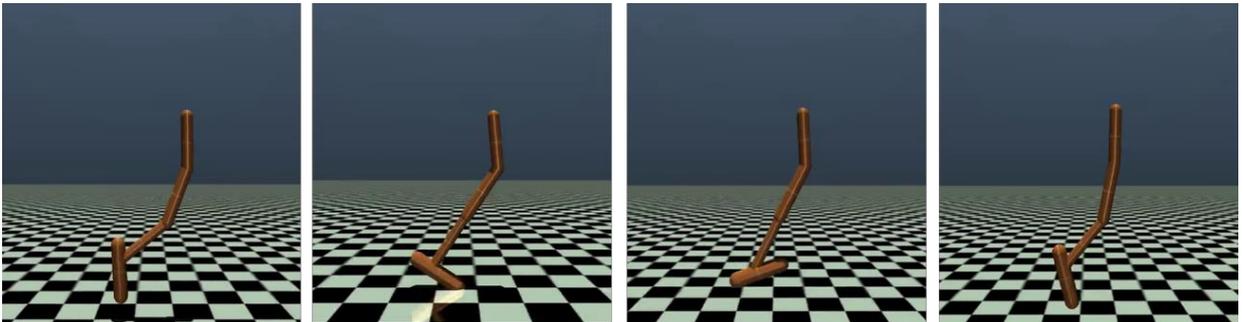


Ilustración b. Comportamiento de un agente que alcanza 3500 de reward.

5. Conclusiones

MCE-IRL es un enfoque sólido e interpretable, pero limitado a dominios simples, tabulares y presenta unos costes computacionales altos al escalar el entorno. AIRL en términos de imitación es excelente incluso con pocas demostraciones, lo cual es útil en contextos en los que es difícil obtenerlas. Sin embargo, las funciones de recompensa aprendidas no generalizan para entrenar agentes desde cero. Por último, T2R demuestra el potencial de los LLMs, produciendo funciones de recompensa densas y estables que alcanzan resultados óptimos incluso en entornos continuos como *Hopper*. La combinación de T2R y AIRL nos ha permitido obtener un agente con un rendimiento nunca reportado en *Seals Hopper*.

Cómo trabajo futuro, sería de interés ir por la línea de los modelos de lenguaje. Realizar una comparativa con modelos de lenguaje variados. En relación con GCL quedaría la implementación y comparativa.

6. Referencias

- [1] Ziebart, B. D., Maas, A. L., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, 1433–1438.
- [2] Ramachandran, D., & Amir, E. (2007). Bayesian inverse reinforcement learning. *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2586–2591.
- [3] Fu, J., Luo, K., & Levine, S. (2018). Learning robust rewards with adversarial inverse reinforcement learning. *International Conference on Learning Representations (ICLR)*.
- [4] Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 49–58. PMLR 48.
- [5] Christiano, P. F., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems (NeurIPS)*, 4299–4307.
- [6] Schaul, T., Horgan, D., Gregor, K., & Silver, D. Universal Value Function Approximators. *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, PMLR 37, 1312–1320, 2015.
- [7] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., & Zaremba, W. Hindsight Experience Replay. *Advances in Neural Information Processing Systems (NeurIPS 2017)*, 2017
- [8] Florensa, C., Held, D., Geng, X., & Abbeel, P. Automatic Goal Generation for Reinforcement Learning Agents (Goal GAN). *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, PMLR 80, 2018.
- [9] Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., & Levine, S. Visual Reinforcement Learning with Imagined Goals (RIG). *Advances in Neural Information Processing Systems (NeurIPS 2018)*, 2018
- [10] Ziebart, B. D. (2010). *Modeling purposeful adaptive behavior with the principle of maximum causal entropy* (Doctoral dissertation). Carnegie Mellon University.
- [11] Xie, T., Zhao, S., Wu, C. H., Liu, Y., Luo, Q., Zhong, V., Yang, Y., & Yu, T. (2024). Text2Reward: Reward shaping with language models for reinforcement learning. *International Conference on Learning Representations (ICLR)*.

COMPARATIVE STUDY AND DEVELOPMENT OF ALGORITHMS FOR REWARD LEARNING AND GOAL-CONDITIONED LEARNING: ANALYSIS AND APPLICATION

Abstract

This study compares three algorithms in the field of *Reward Learning* (MCE-IRL, AIRL, and Text2Reward) with the objective of serving as a usage guide according to the researcher's interest. The selected environments are *CliffWorld* for MCE-IRL and Hopper for AIRL and T2R. MCE-IRL, which is based on the concept of maximum entropy, obtains interpretable rewards that are close to the real ones using expert trajectories, but it is limited to small, tabular environments. AIRL follows the architecture of adversarial problems: a generator that tries to imitate the trajectories of an expert and a discriminator that tries to identify their origin. AIRL achieves stable imitation of the expert without the need for a large number of trajectories. However, the reward net extracted by the discriminator only defines the region of the trajectory space it has seen. This leads it to serve more as an identifier of good trajectories than as a guide to train an expert from scratch. T2R, using prompts with pythonic descriptions of the environment and an instruction for the desired behavior, manages to leverage the capabilities of large language models (LLMs) to generate a dense reward function. Training a SAC agent with the function generated by GPT-4o achieves exceptional performance. Lastly, the combination of T2R + AIRL allowed us to obtain an expert in the modified Hopper-Seals environment with performance never previously reported.

As a complementary framework, a review of *Goal Conditioned Learning* is included, where the learned policies are conditioned on goals, allowing multiple objectives to be achieved. UFVA and HER are summarized, and Goal GAN and RIG are described in detail. Goal GAN makes it possible to generate goals by means of a GAN whose difficulty grows as the agent improves performance. It has been shown to accelerate learning with sparse rewards by generating goals of difficulty appropriate to the agent's knowledge. RIG trains policies from images of the environment without rewards defined over the state. It encodes the observation and the goal with a VAE and defines the reward as the distance in the latent space. It does not need access to the environment state, since it can work directly on renderings, obtaining results close to the oracle.

Keyword: Reward Learning; MCE-IRL; AIRL; Text2Reward; LLMs; Hopper; CliffWorld; Trajectorias, Goal Conditioned Learning; UFVA; HER; Goal GAN; RIG.

1. Introduction

Reinforcement learning allows an agent to optimize behavior by means of a reward function that guides it. However, designing it by hand is complex, and small inaccuracies steer the agent toward undesired or suboptimal policies. This gives rise to Reward Learning, which brings together the set of algorithms that learn what reward function defines a desired behavior. The variety of algorithms is broad: from probabilistic models such as MaxEnt IRL [1] or B-IRL [2], approaches based on adversarial games such as AIRL [3] or Guided Cost IRL [4], preference-based models like RLHF [5], or, recently, natural-language-based models.

Goal Conditioned Learning trains policies conditioned on a goal using a single multitask agent. Notable algorithms include UFVA [6], which extends classical RL with $V(s,g)$ and $Q(s,a,g)$; HER [7] as a replay technique to generate intermediate goals by leveraging failed trajectories; adversarial models such as Goal GAN [8] or VAE-based ones such as RIG [9].

In this work, three representative approaches of the state of the art and its evolution are compared: Maximum Causal Entropy IRL [10], Adversarial IRL, and Text2Reward [11], in a tabular environment (*CliffWorld*) and another, continuous and of greater complexity (*Hopper*). In addition, the RIG and Goal GAN algorithms are described in detail, leaving as future work an implementation and comparison in similar environments.

2. Project Definition

The overall objective is to compare MCE-IRL, AIRL, and T2R in controlled environments, identifying strengths, limits, and usage guidelines.

To this end, some more specific objectives are: (i) to evaluate the fidelity of the learned rewards by analyzing the hierarchy of trajectories; (ii) the ability to guide agents from scratch; (iii) AIRL’s sensitivity to the quantity/quality of demonstrations; (iv) computational feasibility; and (v) the interpretability of the reward functions.

3. Description of algorithms

MCE-IRL is the causal extension of MaxEnt-IRL. It models the reward function with a neural network $R_\theta(s)$ and we solve the MDP using *Soft Bellman Backups* in a finite horizon with known dynamics. This restricts its use to purely tabular environments. From the Q and V obtained in the solution, a policy is induced, and the occupancy measure is computed. It is compared with the expert occupancy, with the difference being the loss used to update the network parameters.

AIRL consists of adversarial training on expert trajectories. The generator uses an on-policy algorithm such as PPO to learn to imitate these trajectories. The discriminator is a potential-based neural network composed of g_θ , which models the general reward function, and a shaping term h_ϕ to separate it from the dynamics and stabilize training. Instead of full trajectories, they are split into triplets (s, a, s') that provide greater stability and allow local improvements to the policy.

Text2Reward starts with an LLM and generates Python code for a dense reward from the pythonic description of the environment and the instruction. For Hopper it is: “Control the Hopper to move in the forward direction.”

In MCE-IRL we obtain experts by training with the environment’s original reward function. For AIRL we have 2 experts that generate trajectories. The first is the best available in public repositories, achieving a reward of 2860 on the original reward. The second arises from combining the T2R approach, which allowed us to obtain an agent that reaches up to 3500 but with a lot of instability. The language models chosen in Text2Reward are GPT-4o and DeepSeek-o1.

Goal GAN uses a GAN in goal space to propose intermediate-difficulty objectives according to the agent’s level of knowledge. Goals are selected from the buffer and from the generator. It estimates the agent’s competence on each goal by measuring the success rate. The discriminator classifies whether they are intermediate goals, between 20%-80% success, while the generator tries to produce goals that the discriminator will consider intermediate. With the collected data, an off-policy agent is trained.

RIG learns goal-conditioned policies from images via a VAE. The autoencoder sends the observations to a compact latent space and defines the reward as the distance in that space between the current and target images. The goals proposed during training can be manually provided or imagined by the VAE by drawing them from the latent space. The agent’s policy is updated with transitions $(z_t, a_t, z_{t+1}, z_g, r_t)$, and the VAE is progressively fine-tuned to improve the latent representation.

4. Results

MCE-IRL on CliffWorld recovers occupancy measures almost identical to the expert ones. As the grid size grows, the learned reward function drifts away from the true one. It has been evaluated with sizes from 4x7 to 80x80. Runtime and memory usage are 1 second and 727 MB for 4x7, and more than 4 hours and 2.7 GB for 80x80. This reflects an exponential growth in resources with the size of the environment.

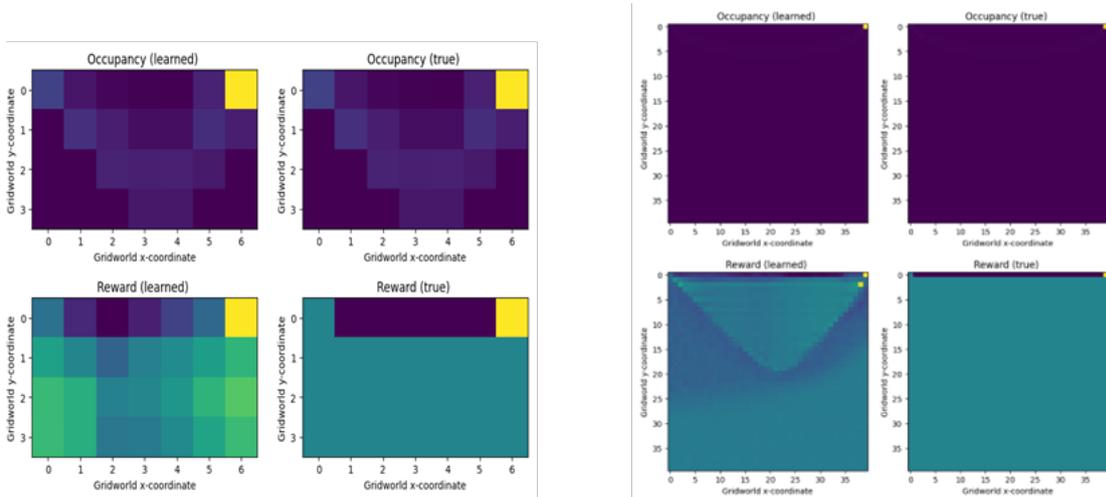


Ilustración c. Occupancy and Rewards learned for 4x7 and 80x80 grids.

AIRL on the 2860 Expert showed a good capacity for imitation. With 10 and 50 trajectories we obtain agents that average 2600 ± 118 and 2506 ± 160 . However, with 100 trajectories the result drops to 2106 ± 23 . The explanation for this behavior is due to two reasons: overfitting of the discriminator in early stages, and the fact that the Expert does not actually perform the correct mechanics because of an exploit in the original reward function (it drags).

By isolating ~ 3500 return trajectories from the second expert, exceptional imitation is achieved. With 10, 50, and 100 trajectories, rewards all exceed 3200. We highlight the agent trained with 100 trajectories, which achieves metrics of 3511 ± 11 . In the Seals-modified Hopper environment there is no agent that reaches similar values. The analysis of the learned reward functions reveals that they are incapable of guiding an agent from scratch. We were

unable to get any of these agents to exceed 300, which is practically at the level of a random policy. The reward nets classify as good the trajectories similar to those seen during training, but they cannot generalize to other regions of the observation space.

With T2R we train a SAC agent and a PPO agent with the reward function generated by GPT-4o and by DeepSeek-o1. To analyze their performance, the agents were evaluated on the environment's original reward function. In all cases, values close to or above 3500 are obtained. The main differences lie in the stability of the agents, with GPT-4o SAC standing out at 3542.6 ± 2.4 , versus the rest, which have a standard deviation between 500 and 900.

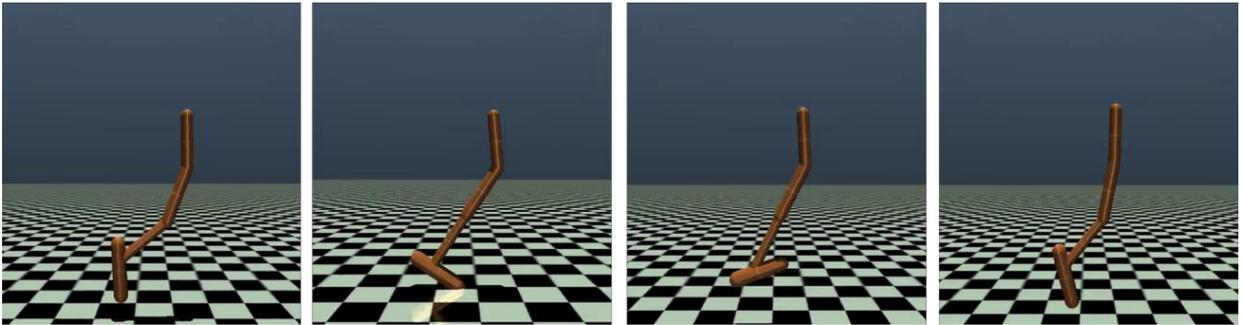


Ilustración d. 3500 reward trayectory..

5. Conclusions

MCE-IRL is a solid and interpretable approach, but limited to simple, tabular domains, and it presents high computational costs as the environment scales. AIRL, in terms of imitation, is excellent even with few demonstrations, which is useful in contexts where they are difficult to obtain. However, the learned reward functions do not generalize to train agents from scratch. Lastly, T2R demonstrates the potential of LLMs, producing dense and stable reward functions that achieve optimal results even in continuous environments such as Hopper. The combination of T2R and AIRL has allowed us to obtain an agent with performance never reported on Seals Hopper.

As future work, it would be of interest to continue along the line of language models, carrying out a comparison with a variety of language models. Regarding GCL, implementation and comparison remain to be done.

6. References

- [1] Ziebart, B. D., Maas, A. L., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, 1433–1438.
- [2] Ramachandran, D., & Amir, E. (2007). Bayesian inverse reinforcement learning. *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2586–2591.
- [3] Fu, J., Luo, K., & Levine, S. (2018). Learning robust rewards with adversarial inverse reinforcement learning. *International Conference on Learning Representations (ICLR)*.

- [4] Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 49–58. PMLR 48.
- [5] Christiano, P. F., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems (NeurIPS)*, 4299–4307.
- [6] Schaul, T., Horgan, D., Gregor, K., & Silver, D. Universal Value Function Approximators. *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, PMLR 37, 1312–1320, 2015.
- [7] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., & Zaremba, W. Hindsight Experience Replay. *Advances in Neural Information Processing Systems (NeurIPS 2017)*, 2017
- [8] Florensa, C., Held, D., Geng, X., & Abbeel, P. Automatic Goal Generation for Reinforcement Learning Agents (Goal GAN). *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, PMLR 80, 2018.
- [9] Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., & Levine, S. Visual Reinforcement Learning with Imagined Goals (RIG). *Advances in Neural Information Processing Systems (NeurIPS 2018)*, 2018
- [10] Ziebart, B. D. (2010). *Modeling purposeful adaptive behavior with the principle of maximum causal entropy* (Doctoral dissertation). Carnegie Mellon University.
- [11] Xie, T., Zhao, S., Wu, C. H., Liu, Y., Luo, Q., Zhong, V., Yang, Y., & Yu, T. (2024). Text2Reward: Reward shaping with language models for reinforcement learning. *International Conference on Learning Representations (ICLR)*.

Índice de la memoria

1. Introducción	5
1.1. Contexto y Motivación.....	5
1.2. Objetivos	6
1.3. Estructura del Trabajo	6
2. Estado del Arte.....	6
2.1. Reward Learning	6
2.2. Goal Conditioned Learning	7
3. Metodología	8
3.1. Descripción Técnica	8
3.1.1. <i>Maximum Causal Entropy</i>	8
3.1.2. <i>Adversarial IRL</i>	9
3.1.3. <i>Text2reward</i>	10
3.1.4. <i>Goal GAN</i>	11
3.1.5. <i>RIG</i>	12
3.2. Diseño del Modelo	12
3.2.1. <i>Modificación de entornos Gymnasium</i>	13
3.2.2. <i>Problema del horizonte variable</i>	13
3.2.3. <i>Adaptaciones específicas Text2Reward</i>	13
3.2.4. <i>Selección de entornos</i>	14
3.3. Entrenamiento y Validación	14
3.3.1. <i>Trayectorias</i>	14
3.3.2. <i>Modelos de lenguaje</i>	15
3.3.3. <i>Agentes</i>	15
4. Experimentos	15
4.1. Objetivos	15
4.2. Conjunto de Datos	15
4.2.1. <i>MCE-IRL</i>	15
4.2.2. <i>AIRL</i>	16
4.2.3. <i>Text2Reward</i>	17
4.3. Configuración.....	17
4.3.1. <i>Entrenamiento MCE-IRL</i>	17

4.3.2.	<i>Entrenamiento AIRL</i>	17
4.3.3.	<i>Entrenamiento T2R</i>	18
4.4.	<i>Análisis de Rendimiento</i>	18
4.4.1.	<i>Rendimiento MCE-IRL</i>	18
4.4.2.	<i>Rendimiento AIRL</i>	18
4.4.3.	<i>Rendimiento T2R</i>	20
4.5.	<i>Visualización</i>	21
4.5.1.	<i>Gráficas MCE-IRL</i>	21
4.5.2.	<i>Gráficas AIRL</i>	22
5.	<i>Resultados</i>	23
6.	<i>Conclusiones y Trabajos Futuros</i>	23
7.	<i>Bibliografía</i>	24
ANEXO I		25

Índice de figuras

Ilustración a. Occupancy y Rewards aprendidas para grids de 4x7 y 80x80.	6
Ilustración b. Comportamiento de un agente que alcanza 3500 de reward.	7
Ilustración c. Mejor experto disponible para Hopper Seals.....	16
Ilustración d. Comportamiento del experto SAC en trayectorias de 3500 de recompensa. 16	
Ilustración e. Entrenamiento del SAC personalizado.....	16
Ilustración f. Evaluación del experto SAC sobre la reward original.	17
Ilustración g. Evolución de la recompensa durante el entrenamiento de los agentes.....	18
Ilustración h. Occupancy y Reward aprendidas vs Real (4 x 7).....	21
Ilustración i. Occupancy y Reward aprendidas vs Real (10 x 10).....	21
Ilustración j. Occupancy y Reward aprendidas vs Real (20 x 20).....	22
Ilustración k. Occupancy y Reward aprendidas vs Real (40 x 40).....	22
Ilustración l. Occupancy y Reward aprendidas vs Real (80 x 80).....	22
Ilustración m. Gráfica de la evolución de los aprendices durante el entrenamiento adversarial.....	22

Índice de tablas

Tabla 1. Rendimiento aprendices AIRL con Experto A.....	18
Tabla 2. Métricas de las reward nets aprendidas mediante AIRL	20
Tabla 3. Evaluación de los agentes entrenados con funciones de recompensas de distintos LLMs.	20
Tabla 4. Configuraciones agentes PPO y SAC.	25
Tabla 5. Configuración reward nets.	25
Tabla 6. Configuración Adversarial Trainer.....	25

1. INTRODUCCIÓN

El aprendizaje por refuerzo permite a un agente optimizar un comportamiento por medio de una función de recompensa que lo guíe. La función de recompensa debe contener la información necesaria para poder alcanzar ese objetivo. Sin embargo, diseñarla a mano es un problema complejo debido a que pequeñas imprecisiones inducen al agente a conductas no deseadas o subóptimas. En entornos y objetivos complejos es un reto extra la capacidad de poder codificar un comportamiento, muchas veces requiere de un conocimiento experto que no se posee.

El *Reward Learning* aborda este problema aprendiendo la función de recompensa mediante trayectorias expertas, preferencias o incluso descripciones en lenguaje natural. En este trabajo se estudian tres enfoques representativos: *Maximum Causal Entropy IRL*, *Adversarial IRL* y *Text2Reward*. Esta selección cubre desde formulaciones probabilísticas hasta métodos más recientes basados en modelos de lenguaje.

La contribución principal es presentar un análisis comparativo en entornos controlados. Se aborda un caso tabular (*CliffWorld*) para MCE-IRL y un caso continuo y de mayor complejidad (*Hopper*) para AIRL y T2R.

El trabajo muestra una guía práctica en la que se obtienen resultados entre los que destacamos: MCE-IRL recupera recompensas interpretables, pero con un coste computacional y limitación en la complejidad del entorno. AIRL destaca en términos de imitación, produciendo agentes capaces de reproducir trayectorias expertas con estabilidad. Sin embargo, AIRL recupera unas funciones de recompensa incapaces de guiar a un agente desde cero. Por otro lado, T2R nos demuestra el potencial y la capacidad de los LLMs. Consigue generar funciones de recompensa densas que consiguen guiar de forma excelente al rendimiento deseado. Destacamos que la combinación de T2R y AIRL nos ha permitido obtener un agente en la variedad *Seals*

de *Hopper* con un rendimiento que hasta la fecha nadie había alcanzado.

Por otro lado, Goal Conditioned Learning es el campo del RL cuyo objetivo es que un único agente pueda alcanzar varios objetivos. Es decir, que aprenda una política condicionada a una meta.

En este trabajo se revisa el estado del arte y se proponen 2 algoritmos para su evaluación. Los algoritmos seleccionados son Goal GAN y RIG, los cuales son explicados en detalle. Como trabajo futuro, se propone implementarlos y hacer una comparativa rigurosa en entornos controlados, tal y como se ha hecho con *Reward Learning*.

1.1. CONTEXTO Y MOTIVACIÓN

El diseño de funciones de recompensas es una causa frecuente de fallos en *Reinforcement Learning* (RL). Señales mal definidas inducen a los agentes a reproducir conductas indeseadas.

Los agentes disponibles en repositorios como *Stable Baselines*, entrenados en entornos como *Hopper* sobre la *ground-truth reward* alcanzan retornos altos, pero sin reproducir la dinámica correcta, aprovechando el avance “arrastrado” en lugar de saltando.

Este tipo de problemas empeora con la complejidad del entorno o del comportamiento que queremos aprender. Entornos con terminaciones por estados indeseados y con episodios de horizonte variables hace que aparezcan *exploits* en la recompensa.

Por ello, surge el campo del *Reward Learning* que propone soluciones a estos problemas. Se presentan métodos basados en *Inverse Reinforcement Learning* (IRL), preferencias, aprendizaje adversarial y lenguaje natural. Sin embargo, no existen comparativas rigurosas de estos métodos que ayuden a investigadores a seleccionar la mejor opción en función de sus objetivos y limitaciones.

El objetivo de GCL es entrenar agentes multitarea sobre un entorno. Para ello, se deben aprender

políticas condicionadas a la meta que se desea alcanzar.

Una de las bases del campo es UFVA, que extiende el modelo clásico de RL añadiendo el término de meta g a $V(s,g)$ y $Q(s,a,g)$.

Otro de los principales problemas es la definición de metas en entornos complejos. Aquí surgen técnicas como HER que etiquetan trayectorias fallidas de una meta como exitosas considerando el estado alcanzado como un objetivo más.

Goal GAN y RIG son enfoques más modernos. El primero genera metas de forma progresiva utilizando una arquitectura adversarial. El segundo trabaja con *autoencoders* de imágenes que compactan el estado actual y objetivo, definiendo la función de recompensa como su distancia en el espacio latente. Permite también aprovechar el *encoder* para imaginar metas.

1.2. OBJETIVOS

Objetivo general:

El objetivo principal es comparar MCE-IRL, AIRL y T2R en entornos controlados, identificando fortalezas, límites y pautas de uso. Para GCL, explicar en detalle los algoritmos de Goal GAN y RIG.

Objetivos específicos:

- Evaluar la fidelidad de las funciones de recompensa aprendidas, analizando puntos como la jerarquía de trayectorias.
- Evaluar la capacidad de las funciones de recompensa para guiar a un agente desde cero.
- Analizar la sensibilidad de AIRL en función de la cantidad y calidad de las trayectorias expertas.
- Comparar la capacidad de varios LLMs para generar funciones de recompensa coherentes.
- Estudiar el escalado computacional y viabilidad de MCE-IRL.

- Valorar la interpretabilidad de las recompensas.

1.3. ESTRUCTURA DEL TRABAJO

El trabajo se divide en varias secciones.

- Estado del arte: análisis del estado actual en el campo.
- Metodología: definiciones teóricas de los algoritmos seleccionados y modificaciones.
- Experimentos: implementación, entrenamiento y evaluación de los diferentes algoritmos.
- Resultados: Análisis de los experimentos.
- Conclusiones: Resumen de los resultados y propuestas futuras.

2. ESTADO DEL ARTE

2.1. REWARD LEARNING

Se entiende por *Reward Learning* al conjunto de técnicas cuyo objetivo es inferir o aprender la función de recompensa, por lo general $R(s)$ o $R(s,a)$, que explique el aprendizaje de un comportamiento de interés. Esta función puede ser utilizada posteriormente para entrenar un agente mediante algoritmos clásicos de *Reinforcement Learning* (RL) que maximizan la recompensa con el fin de reproducir el comportamiento deseado [1].

El problema ha sido abordado de diversas formas a lo largo de la historia. Comenzando con aproximaciones teóricas en entornos tabulares, hasta técnicas más recientes basadas en *Deep Neural Networks* (DNNs) o *Large Language Models* (LLMs).

En la literatura destaca el término de *Inverse Reinforcement Learning* (IRL), propuesto por Ng y Russell [1], que define el problema como la búsqueda de una función de recompensa que justifique como óptimas, otorgando recompensas altas; a aquellas trayectorias generadas por un experto en un entorno de *Markov Decision Process* (MDP).

Uno de los principales desafíos de este enfoque es su inherente ambigüedad, ya que múltiples funciones de recompensa pueden generar o explicar la misma política.

Para solucionarlo, surgen variantes como *Maximum Entropy IRL* [2] o *Bayesian IRL* (B-IRL) [3]. La primera introduce el principio de entropía, seleccionando de entre todas las políticas compatibles con la del experto, aquella más estocástica. Por otro lado, B-IRL adopta un enfoque probabilístico, generando una distribución posterior sobre las funciones de recompensa condicionada a las trayectorias observadas.

El auge del *Deep Learning* dio lugar a técnicas como *Adversarial IRL* (AIRL) [4] o *Guided Cost Learning* [5], que reformulan el problema como uno adversarial con una arquitectura basada en *Generative Adversarial Networks* (GANs). Este avance permitió aplicar *Reward Learning* a entornos de mayor complejidad y generar funciones de recompensa más transferibles.

Paralelamente, surgieron métodos *preference-based*, como *Reinforcement Learning from Human Feedback* (RLHF) [6], clave en el desarrollo de sistemas como ChatGPT. Estos métodos permiten aprender a partir de comparaciones entre trayectorias con retroalimentación humana. En sintonía, aparece la familia REX (*T-REX*, *D-REX*) [7], que permiten aprender de trayectorias subóptimas ordenadas por calidad, sin necesidad de un experto perfecto.

Recientemente emergieron algoritmos como *Text2Reward* [8] o *EUREKA* [9] que obtienen funciones de recompensa densas directamente mediante instrucciones en lenguaje natural, aprovechando la capacidad de grandes modelos generativos basados en texto. Estos enfoques eliminan la necesidad de datos expertos o la compleja codificación manual de las recompensas.

Este trabajo se centra en analizar y comparar tres enfoques representativos de *Reward Learning*: *Maximum Causal Entropy IRL* (MCE-IRL) [10], AIRL y Text2Reward.

La selección abarca desde algoritmos basados en métodos probabilísticos (MCE-IRL), pasando por técnicas modernas basadas en aprendizaje adversarial y redes profundas (AIRL), hasta los enfoques más recientes que utilizan LLMs (Text2Reward).

Pese a la existencia de numerosos avances, se observa una carencia de estudios comparativos que evalúen sus rendimientos en entornos controlados y con distintos niveles de complejidad.

La presente contribución busca no solo sintetizar el estado del arte, sino también establecer una base comparativa práctica que permita identificar fortalezas y debilidades de cada enfoque. Resulta especialmente útil para investigadores y desarrolladores que deban elegir entre diferentes técnicas en función del tipo de entorno, recursos disponibles y nivel de intervención humana requerido.

2.2. GOAL CONDITIONED LEARNING

Se entiende por Goal Conditioned Learning (GCL) al conjunto de técnicas de Reinforcement Learning cuyo objetivo es aprender un comportamiento condicionado a metas específicas. En estos casos, el agente es capaz de resolver múltiples tareas al recibir como entrada un objetivo.

La variedad de modelos propuestos es amplia. Destaca el uso de funciones de valor universales (UVFA) [11] que extienden $V(s)$ o $Q(s,a)$ a $V(s,g)$ o $Q(s,a)$, incorporando de forma explícita la meta como una entrada más.

También, métodos como *HER* [12] revolucionaron el campo. Durante el entrenamiento a un objetivo, se reutilizan trayectorias fallidas interpretándolas como un éxito a una meta distinta.

El uso de redes adversariales también se aplica en CGL, destacando *Goal GAN* [13]. Durante el entrenamiento se generan metas intermedias de dificultad creciente a medida que el agente aprende.

Por último, destacan las metas visuales e imaginación de estados. El algoritmo más representativo es Reinforcement Learning with Imagined Goals (RIG) [14], que aprende políticas condicionadas en alta dimensionalidad como imágenes usando un *Variational Autoencoder* (VAE) que obtiene un espacio latente donde se define una recompensa densa y calcula la distancia al objetivo, mientras muestrea metas imaginadas.

3. METODOLOGÍA

3.1. DESCRIPCIÓN TÉCNICA

En esta sección se presentan los algoritmos seleccionados para el estudio comparativo de Reward Learning: *Maximum Causal Entropy* (MCE-IRL), *Adversarial IRL (AIRL)* Y *Text2Reward (T2R)*.

3.1.1. MAXIMUM CAUSAL ENTROPY

El algoritmo de *Maximum Entropy IRL* (MaxEnt IRL), propuesto por Zierbart et al. (2008) [2], es una de las primeras soluciones al problema de ambigüedad que presenta de forma innata el *Inverse Reinforcement Learning*. Este problema surge al existir múltiples funciones de recompensa que pueden justificar el comportamiento del experto.

MaxEnt IRL lo aborda seleccionando aquella función de recompensa cuya política óptima maximice la entropía, es decir, tiene preferencia por las políticas más estocásticas. La función de recompensa $R(s)$ o $R(s,a)$ se expresa como una combinación lineal de características $\phi(s)$ o $\phi(s,a)$.

$$R(s, a) = \theta^\top \phi(s, a)$$

El proceso para definir buenas características del entorno puede ser complejo. *Deep MaxEnt IRL* es una extensión del algoritmo que modela la función de recompensa como una red neuronal.

$$R(s, a) = f_\theta(s, a)$$

Esto permite captar relaciones más complejas y no linealidades.

Maximum Causal Entropy IRL es una extensión natural de MaxEnt IRL que introduce el concepto de entropía causal.

A diferencia del enfoque anterior, este algoritmo añade una restricción temporal más realista. Las decisiones del agente dependen exclusivamente de la información disponible hasta el momento actual, no del futuro.

El funcionamiento del algoritmo es el siguiente:

1. Se define la función de recompensa $R(s)$ como una red neuronal que toma el espacio de observación (estados). Esta red es configurable en tanto a capas ocultas, funciones de activación y *dropout*.

2. Se realiza el cálculo de $V(s)$ y $Q(s,a)$ mediante *Soft Bellman Backup*. Esto limita el algoritmo a entornos de horizonte finito y tabulares donde la matriz de transiciones $P(s'|s, a)$ es conocida.

$$Q(t, s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) V(t + 1, s')$$

$$V(t, s) = \log \sum_a \exp(Q(t, s, a))$$

3. Cálculo de la política:

$$\pi(t, s, a) = \frac{\exp(Q(t, s, a) - V(t, s))}{\sum_{a'} \exp(Q(t, s, a') - V(t, s))}$$

4. Medida de Ocupación $D(s)$

$$D(t + 1, s') = \sum_a D(t, s) \pi(t, s, a) P(s'|s, a)$$

5. Medida de ocupación acumulado.

$$D_{cum}(s) = \sum_t \gamma^t D(t, s)$$

6. Comparación con experto / Función de pérdida

$$R_{\theta}(s) = \sum_a \pi_{\theta}(a|s) \cdot R_{\theta}(s, a)$$

$$L(\theta) = \sum_s (D_{cum}(s) - D_{experto}(s)) R_{\theta}(s)$$

7. Optimización: mediante descenso de gradiente se optimizan los parámetros de la red neuronal $R(s)$ de forma iterativa hasta cumplir una condición de parada.

```
Initialize:
-  $\theta \leftarrow$  random parameters for reward network
- Set horizon T (max trajectory length)
- Learning rate  $\alpha$ 
- Convergence threshold  $\epsilon$ 

Loop until convergence:
1. Compute reward:
  For all (s):
     $R(s, a) \leftarrow R_{\theta}(s)$ 
2. Soft value iteration (backward pass)
    $Q(t, s, a)$  and  $V(t, s)$ 
3. Compute policy  $\pi$ 
4. Forward pass: compute state occupancy  $D_s$ 
5. Compute cumulative occupancy  $D_{sum}$ 
6. Estimate expert occupancy  $D_{expert}$ 
7. Compute loss:
8. Update  $\theta$ :
    $\theta \leftarrow \theta - \alpha * \nabla_{\theta} L(\theta)$ 
9. Check convergence:
   If  $|\Delta L| < \epsilon$ :
     break

Output:
Learned reward function  $R_{\theta}(s)$ 
Stochastic policy  $\pi(t, s, a)$ 
```

3.1.2. ADVERSARIAL IRL

Adversarial IRL, propuesto por Fu et al. (2018) [4], reformula el problema de IRL como una tarea de entrenamiento adversarial entre dos redes neuronales. Está profundamente inspirado en la estructura de las *Generative Adversarial Networks* (GANs).

AIRL plantea una competencia entre dos componentes:

- **Generador:** un agente cuya política genera trayectorias con el objetivo de ser lo más parecidas a un experto.
- **Discriminador:** red neuronal que trata de distinguir si una trayectoria proviene del generador o del conjunto de trayectorias expertas.

Este algoritmo presenta una modificación en la estructura del discriminador. El enfoque clásico descrito en *GAN-GCL* [11] discrimina sobre las trayectorias completas:

$$D_{\theta}(\tau) = \frac{\exp\{f_{\theta}(\tau)\}}{\exp\{f_{\theta}(\tau)\} + \pi(\tau)}$$

Esto acarrea un alto coste computacional y aprendizaje lento. Al juzgarse el episodio entero no se localiza en qué parte de la trayectoria se necesitan correcciones en la política.

AIRL reduce el problema a tripletas (s, a, s') que aportan más información local y estabilidad en el aprendizaje. El discriminador se define como:

$$D_{\theta}(s, a, s') = \frac{\exp\{f_{\theta, \phi}(s, a, s')\}}{\exp\{f_{\theta, \phi}(s, a, s')\} + \pi(a|s)}$$

La función aprende a diferenciar las trayectorias, o en este caso tripletas, que provienen del experto o del generador.

$$f_{\theta, \phi}(s, a, s') = g_{\theta}(s, a) + \gamma h_{\phi}(s') - h_{\phi}(s)$$

Aquí g_θ representa la recompensa aprendida, mientras que h_ϕ actúa como potencial de *shaping* ligado a la dinámica.

Esta configuración surge para solucionar el problema de ambigüedad del *reward* y dependencia de la dinámica. Bajo una dinámica T fija, existen infinitas funciones de recompensa que inducen a la política experta. Sin embargo, ligeros cambios en la dinámica T' o el entorno afectan tanto que la recompensa deja de generar una política óptima. Mediante esta formulación se soluciona el problema pues h_ϕ absorbe la información de la dinámica y g_θ aprende los componentes importantes de la toma de decisiones. Así, g_θ es transferible y general.

La recompensa con la que se actualiza la política del generador se obtiene como:

$$r_{\theta,\phi}(s, a, s') = \log D_\theta(s, a, s') - \log(1 - D_\theta(s, a, s'))$$

El funcionamiento del algoritmo es el siguiente:

1. Configuración inicial:
 - a. Recolectar conjuntos de trayectorias expertas $\tau = (s_1, a_1, \dots, s_T, a_T)$.
 - b. Inicializar agente generador.
 - c. Definir red neuronal compuesta por una red básica y una potencial, que modelan g_θ y h_ϕ respectivamente.
2. Entrenamiento del discriminador: Se entrena un discriminador $D_\theta(s, a, s')$ que aprende a distinguir si una transición proviene del experto o del agente.
3. Recolectar trayectorias: se ejecuta la política del agente $\pi_\theta(a|s)$ en el entorno para recolectar nuevas trayectorias.
4. Actualizar política: se mejora la política del generador para producir trayectorias más parecidas al experto y confundir al discriminador.
5. Se repite iterativamente los puntos 2, 3, 4 alternando el entrenamiento del generador y

discriminador hasta que la política se establezca y el discriminador no alcance a discernir de quien viene la trayectoria.

```
# Adversarial Inverse Reinforcement Learning (AIRL)
1: Obtain expert trajectories.
2: Initialize policy and discriminator.
3: for step  $t$  in  $1, \dots, N$  do
    4: Collect trajectories by executing  $\pi$ .
    5: Train  $D_\theta$  to classify
    6: Update  $r_{\theta,\phi}(s, a, s') = \log D_\theta(s, a, s') - \log(1 - D_\theta(s, a, s'))$ 
    7: Update  $\pi$  with respect  $r_{\theta,\phi}$ 
8: end for
```

Pseudo-código AIRL

3.1.3. TEXT2REWARD

Text2Reward, propuesto por Wang et al. (2023) [8], es un enfoque reciente de *Reward Learning* basado en generar funciones de recompensa densas a partir de una descripción en lenguaje natural mediante el uso de LLMs.

Esta técnica elimina la necesidad de trayectorias expertas o de un conocimiento avanzado para modelar la función de recompensa que guíe a un agente para aprender un comportamiento deseado. Aprovecha la potencia de los grandes modelos de lenguaje para generar unas funciones óptimas e interpretables.

La calidad de las funciones de recompensa generadas recae principalmente en dos puntos: la claridad y expresividad de la instrucción y el modelo de lenguaje elegido.

Modelos como GPT-4 de OpenAI o Claude de Anthropic han demostrado ser capaces de conseguir resultados excelentes.

También es necesario proporcionar al LLM información sobre el entorno en el que se pretende

trabajar. Para ello, el modelo ingiere un *prompt* específico del entorno sobre el que el agente actuará.

Este *prompt* incluye:

- Instrucción con la tarea que queremos que el agente aprenda. Por ejemplo: “Haz un backflip”.
- Representación *pythonic* del entorno que debe indicar como está construido y qué clases tiene.
- *Few-shots*: opcionalmente se pueden agregar otras instrucciones junto con la función de recompensa verdadera de la misma.

Una de las principales debilidades es la ambigüedad textual al proporcionar una tarea. Esto sucede cuando el LLM entiende un objetivo que no es el realmente buscado. Una posible solución está en añadir la posibilidad de dar *feedback* que permite al modelo volver a generar una función más acorde.

El pipeline general se divide en las siguientes fases:

1. Proporcionar instrucción que queremos aprender.
2. La instrucción se integra en el *prompt template* del entorno seleccionado.
3. LLM procesa *prompt* y genera el código fuente en Python con la función de recompensa.
4. Algoritmo RL clásico como PPO o SAC envuelve el entorno con un *wrapper* que sustituye la función de recompensa original por la nueva.
5. *Finetune* opcional: Si no se alcanza el resultado deseado se realimenta el LLM añadiendo una explicación de cuál es el comportamiento que se está obteniendo y cómo mejorarlo. Por ejemplo: “Está caminando con los brazos y debería caminar con las piernas”.

3.1.4. GOAL GAN

La idea principal es usar un Generative Adversarial Network sobre el espacio de metas para proponer objetivos de dificultad intermedia en función del nivel actual del agente.

El algoritmo se compone principalmente de:

- Estimador de competencia: se calcula la tasa de éxito del agente para cada meta.
- Discriminador: aprende a distinguir metas buenas. dado un rango de competencia definido.
- Generador: produce metas nuevas consideradas buenas por el discriminador.

El pipeline del algoritmo consiste en:

1. Muestreo de metas y trayectorias: Ejecuta la política $\pi_{\theta}(a | s, g)$ condicionada con metas extraídas de:
$$g \sim \alpha \cdot g_G + (1 - \alpha) g_{replay}$$
2. Tras n intentos a una meta se estima la competencia.
 - Criterio de éxito: $\|s_t - g\|_2 \leq \epsilon$
 - Competencia: $C_{\theta}(g) = \frac{1}{n} \sum \text{éxito}_i$
3. Discriminador se entrena para diferenciar metas intermedias del resto. No se considera intermedias ni a las imposibles ni a las fáciles.
4. Generador aprende a generar nuevas metas con el objetivo de que el discriminador las identifique como intermedias
5. Se actualiza la política del agente con la información recibida (s, g, a, r, s'). Por lo general se usa un algoritmo *off-policy* extendido a UVFA.

3.1.5. RIG

RIG se centra en aprender una política condicionada a metas desde imágenes mediante un VAE para:

- Obtener un espacio latente compacto para comparar estado y meta.
- Definir una recompensa densa como la proximidad en el espacio latente.
- Imaginar metas muestreando del *prior* del VAE.

Además, usa HER para re-etiquetar transiciones con metas alternativas.

El componente principal es el VAE. Su estructura es un encoder-decoder que aprende una transformación de imágenes un espacio latente compacto y a reconstruirlas en base a su representación en ese espacio.

- Encoder: $z = e_\phi(i)$
- Decoder: $i = d_\psi(z)$
- Prior: $N(0, I)$

El entrenamiento se compone de dos fases:

Fase 0:

- Recolección de imágenes del entorno con una política aleatoria.
- Preentrenar VAE con esas imágenes. A lo largo del entrenamiento se irá ajustando suavemente.

Bucle principal:

1. Elección de meta, puede venir de una imagen objetivo dada o una imaginada generada con el prior del VAE.
2. Observación: toma una imagen y obtén su representación en el espacio latente.

$$z_t = e_\phi(i_t)$$

3. Ejecuta la política actual para obtener la siguiente imagen y calcula representación.

$$\text{Política: } \pi_\theta(a | z_t, z_g)$$

$$z_{t+1} = e_\phi(i_{t+1})$$

4. Calcula recompensa mediante la distancia de la imagen con el objetivo en el espacio latente:

$$r_t = \|z_{t+1} - z_g\|_2$$

5. Guardamos en *replay* $(z_t, a_t, z_{t+1}, z_g, r_t)$.
6. Podemos aplicar HER para re-etiquetar metas y ampliar así la cobertura.
7. Se actualiza la política del agente con la información recibida del replay. Es importante notar que el encoder del VAE seguirá siendo necesario pues el estado y el objetivo se tratan desde el espacio latente. Se usa un algoritmo *off-policy* extendido a UVFA.

8. Se hace iterativamente *finetune* del VAE para mejorarlo.

3.2. DISEÑO DEL MODELO

A la hora de implementar los algoritmos se tomaron las siguientes decisiones técnicas, centradas en maximizar la reutilización de código existente y asegurar unos resultados consistentes.

Para los algoritmos de MCE-IRL y AIRL se utilizó la librería de *Imitation*. La implementación disponible en la documentación fue modificada acorde a los objetivos de este trabajo y se creó una versión en Jupyter Notebook y un script Python.

Para Text2Reward se tomó el repositorio oficial, pero pese a mostrar resultados y experimentos en entornos MuJoCo, no había código existente por lo que se ha hecho de cero para este trabajo.

3.2.1. MODIFICACIÓN DE ENTORNOS GYMNASIUM

La librería *Imitation* realiza una serie de modificaciones sobre los entornos de Gymnasium mediante decoradores.

Estos ajustes tienen como objetivo mitigar sesgos comunes que pueden comprometer la evaluación de algoritmos de aprendizaje de recompensas e imitación. En particular, se aplican dos transformaciones destacadas:

- **Posición global:** los entornos clásicos de *Gymnasium* por defecto excluyen esta coordenada del vector de observación con el fin de obtener políticas invariantes a traslaciones.

Esta decisión es útil en el contexto clásico del aprendizaje por refuerzo. Sin embargo, en *Inverse Reinforcement Learning* resulta contraproducente pues comportamientos intrínsecamente relacionados con un cambio en la posición global como “*avanzar hacia delante en Hopper*” resultan prácticamente imposibles de inferir con información local.

Por ello, la librería de *Imitation* usa *Seals* que modifica los entornos añadiendo este estado.

- **Terminación de episodio:** un episodio puede terminar porque se llega a una condición de finalización.

Evitar la terminación anticipada debido a condiciones de fallo, como que un robot se caiga, asegura que el agente tenga la oportunidad de experimentar diferentes escenarios y aprender de ellos sin que el entorno interrumpa el proceso de aprendizaje antes de tiempo.

La función de recompensa que se desea aprender necesita recibir información sobre qué estados deben ser penalizados y si

cortamos las trayectorias al alcanzar esos estados nunca aprenderá a penalizarlos. Esto también soluciona el problema del horizonte variable explicado a continuación.

3.2.2. PROBLEMA DEL HORIZONTE VARIABLE

Los entornos con horizonte variable son comunes porque permiten que los episodios finalicen cuando se alcanzan condiciones de éxito o fracaso.

Aunque esto suele acelerar el entrenamiento, en *Reward Learning* causa un problema. Los algoritmos aprovechan la señal de terminación para predecir el éxito o fracaso, en lugar de aprender el comportamiento real necesario para completar la tarea.

Este fenómeno genera un sesgo hacia trayectorias específicas como las que terminan más rápido o evitan fallos más penalizados, sin que el algoritmo aprenda realmente cómo resolver la tarea.

Para evitar este sesgo, la librería *Imitation* recomienda entrenar los algoritmos exclusivamente en entornos de horizonte fijo, donde los episodios tienen una duración constante y no están influenciados por la señal de terminación.

3.2.3. ADAPTACIONES ESPECÍFICAS TEXT2REWARD

Como se ha mencionado anteriormente, el repositorio oficial de T2R no tenía implementaciones para los entornos MuJoCo por lo que se ha hecho de cero. Muchas secciones de código se han podido reutilizar sin necesidad de hacer muchas modificaciones.

Se han diseñado *prompt templates* para los entornos de CartPole, Ant y Hopper siguiendo el estilo sugerido en el artículo original. Las descripciones del entorno siguen un formato *pythonic*, estructuradas como *docstrings* y con comentarios

que describen explícitamente los objetos, atributos y métodos disponibles en el entorno.

Para una mayor expresividad los entornos están descritos en término de objetos y clases propias de entornos MuJoCo como *self.qpos* o *self.z_vel*. Sin embargo, para poder trabajar con los entornos de *Gymnasium* se añade un proceso de transformación que mapea esas referencias a los índices específicos de los vectores que expone *Gymnasium*.

Estas modificaciones permitieron adaptar Text2Reward a un entorno controlado y estandarizado, facilitando su evaluación en condiciones comparables con el resto de los algoritmos analizados.

Como funcionalidad extra se ha adaptado el código para poder usar modelos de lenguaje locales, a través de herramientas como LM Studio. Esto permite operar de forma completamente *offline*, manteniendo control sobre la generación del código sin depender de una API externa.

Por limitaciones de hardware no se han probado modelos locales.

3.2.4. SELECCIÓN DE ENTORNOS

La elección de los entornos se realizó teniendo en cuenta tanto las capacidades técnicas de cada algoritmo como los objetivos comparativos del estudio.

MCE-IRL se aplicó exclusivamente sobre una versión del entorno *CliffWorld*, una cuadrícula discreta de tamaño reducido. Esta decisión se debe a que MCE-IRL, en su formulación e implementación estándar, requiere espacios de estados y acciones finitos y tabulares, y no escala eficientemente a dominios de observación continua o de alta dimensión.

AIRL y Text2Reward se evalúan sobre el mismo entorno para permitir una comparación directa del

rendimiento entre ambos enfoques. El entorno elegido para ellos es *Hopper*.

3.3. ENTRENAMIENTO Y VALIDACIÓN

El proceso de entrenamiento y validación se diseñó para mantener consistencia entre los distintos algoritmos evaluados, permitiendo una comparación significativa de su rendimiento en condiciones controladas. Las estrategias difieren ligeramente según el enfoque de aprendizaje utilizado.

3.3.1. TRAYECTORIAS

Tanto MCE-IRL como AIRL necesitan un conjunto de trayectorias generadas por agentes experto. Estas trayectorias deben reproducir el comportamiento del cual queremos estimar una función de recompensa que los explique.

MCE-IRL utiliza las trayectorias expertas para estimar la medida de ocupación experta. Con la función de recompensa parametrizada, MCE-IRL resuelve el entorno mediante *Soft Bellman Backups* para obtener la política óptima. Luego los parámetros de la red se actualizan hasta que la ocupación que genera la política aprendida coincida con la del experto.

AIRL utiliza las trayectorias expertas como entradas positivas para el discriminador, mientras que las que produce el generador aprendiz son ejemplos negativos. Realmente las trayectorias se parten en tripletas (s, a, s'). El discriminador aprende a diferenciar cuales vienen del experto y cuáles del aprendiz.

Para los experimentos de AIRL en *Hopper* se dispone de dos expertos.

El primero es el mejor que proporciona *Stable-Baselines* y obtiene una recompensa promedio de 2860 sobre la función de recompensa original.

El segundo es un experto personalizado que se ha obtenido mediante un proceso explicado en la sección de experimentos. Este experto consigue generar trayectorias de más de 3500 de recompensa.

Con ambos expertos se generan sets de 10, 50 y 100 trayectorias para entrenar y un set de 15 para evaluación. Como se ha comentado en la sección de modificaciones de los entornos, las trayectorias tienen un horizonte fijo. En el caso de *Hopper* los episodios tienen una duración de 1000 pasos.

3.3.2. MODELOS DE LENGUAJE

Para Text2Reward se proporciona un *prompt template* por entorno que lo describe. Se puede visualizar en el Anexo I.

3.3.3. AGENTES

Para poder hacer un entrenamiento y evaluación correcta se hace uso de agentes que participan en distintas etapas.

En AIRL hace falta configurar un agente generador de trayectorias. T2R usa directamente un enfoque de *Reinforcement Learning*, por lo que entrenamos un agente sobre la función de recompensa que nos proporciona el LLM.

Los algoritmos utilizados por los agentes elegidos son PPO y SAC. En Adversarial IRL solo usamos PPO pues la implementación de *Imitation* está limitada a métodos *on-policy*. Por otro lado, al ser T2R más flexible, hemos añadido SAC para su evaluación.

4. EXPERIMENTOS

4.1. OBJETIVOS

El objetivo principal es evaluar comparativamente los tres enfoques de *Reward Learning*. Por un lado, MCE-IRL en un entorno tabular *CliffWorld* a modo

introdutorio del campo de estudio. Por otro lado, se hará una comparativa más directa entre AIRL y T2R en el entorno continuo *Hopper*.

Los objetivos se presentan de forma más específica en la siguiente lista:

- Fidelidad de la recompensa aprendida. Analizar la capacidad de preservar la jerarquía de trayectorias respecto a la original.
- Rendimiento y estabilidad de políticas al entrenar con las funciones de recompensa nuevas.
- Sensibilidad a cantidad y calidad de las demostraciones de expertos en AIRL.
- Escalado y viabilidad computacional. Principalmente analizando las limitaciones de MCE-IRL.
- Interpretabilidad. Evaluar si la recompensa aprendida identifica estados objetivo, penaliza estados no deseados y consigue guiar hacia comportamientos óptimos.
-

4.2. CONJUNTO DE DATOS

4.2.1. MCE-IRL

CliffWorld consiste en un *grid* donde el agente debe llegar hasta la meta situada en la esquina superior derecha, evitando caer por un precipicio presente en la primera fila del *grid*.

Para evaluar MCE-IRL se consideraron *grids* de distintos tamaños: 4x7, 10x10, 20x20, 40x40 y 80x80.

Para cada caso se define un horizonte finito H proporcional al tamaño del entorno:

$$H = \text{altura} + \text{anchura}$$

La capacidad de la red neuronal de la red de recompensa se ajusta según el número de estados. La arquitectura de las capas ocultas es de [64, 64], [128, 128] y [256, 256].

4.2.2. AIRL

El entorno *Hopper* consiste en una figura bidimensional con una sola pierna, formada por un torso, muslo, pierna y pie. El objetivo principal es que el agente avance mediante saltos sin caerse. La *ground-truth reward* del entorno codifica el progreso hacia delante, pero como se verá más adelante, no necesariamente refleja el comportamiento deseado.

El mejor experto disponible en *SB3* es el entrenado por los creadores del paquete *Seals*. Un PPO que alcanza un retorno medio de $2862,69 \pm 127,20$ y será nuestra referencia inicial de desempeño óptimo.

Sin embargo, al renderizar trayectorias de este experto podemos observar que, aunque avanza, el agente se arrastra sin seguir el patrón de salto deseado.

Hemos tratado de entrenar varios agentes usando otros algoritmos como SAC o A2C, pero ninguno consigue superar el umbral de los 2900 ni reproducir un comportamiento de salto. Esto evidencia que la función de recompensa original de *Seals* con la que se ha entrenado no codifica correctamente el objetivo deseado.

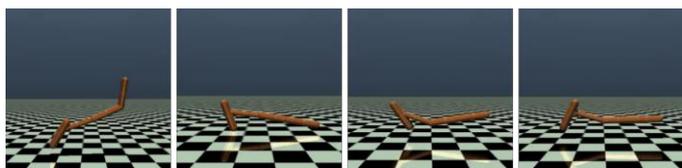


Ilustración e. Mejor experto disponible para Hopper *Seals*.

Para superar estas limitaciones y poder hacer un análisis más completo se hicieron dos experimentos.

1. Tomar agentes expertos del entorno original de *Gymnasium*, que no tienen las modificaciones de *Seals*, y hacer *fine-tune* sobre el entorno de *Seals*.

Los resultados no fueron los deseados. El cambio en la dimensionalidad que se introduce es suficiente para perder a los expertos sin conseguir que se adapten.

2. Codificar nuestra propia función de recompensa densa usando *Text2Reward*. Usando la descripción del entorno que usamos para T2R, añadiendo comentarios acerca de los cambios que hace *Seals* e indicando el objetivo “Control the Hopper to move in the forward direction.”

El primer resultado fue suficiente para poder entrenar un experto SAC capaz de generar trayectorias que cumplieran la dinámica correcta y alcanzaban valores de más de 3500 sobre la función de recompensa original.

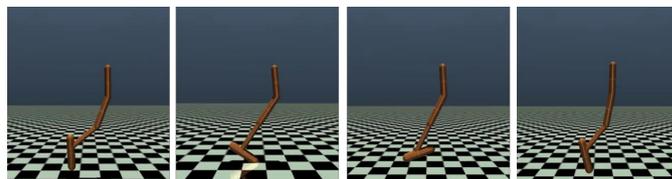


Ilustración f. Comportamiento del experto SAC en trayectorias de 3500 de recompensa.

Se seleccionó el *checkpoint* del modelo que mayor *reward* promedio alcanzaba. Al evaluarlo sobre la recompensa original genera episodios muy oscilantes. Muchos de 1100 de recompensa, lejano a lo que buscamos, pero otros de más de 3500 lo cual jamás se había alcanzado antes.

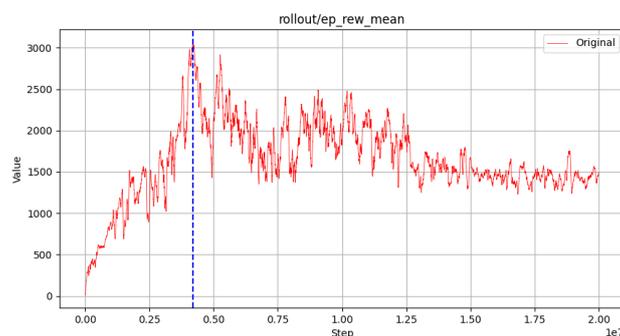


Ilustración g. Entrenamiento del SAC personalizado.

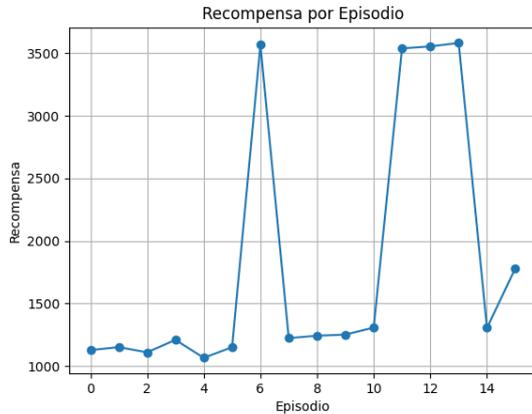


Ilustración h. Evaluación del experto SAC sobre la reward original.

Estos resultados nos revelan varios datos. El primero es que la *reward* original del entorno *Seals* sí que codifica el comportamiento deseado. Sin embargo, el ligero cambio en el espacio de observaciones al tener en cuenta la posición absoluta del *Hopper* hace que los expertos tiendan a quedarse en un máximo local de la función de recompensa, aprendiendo en un comportamiento subóptimo. Por otro lado, vemos el potencial que tienen los LLMs.

4.2.3. TEXT2REWARD

El entorno utilizado es el Hopper original. Se ha hecho uso de un *prompt template* que describe de forma *pythonic* el entorno y permite añadir la tarea específica a realizar. En este caso:

“Control the Hopper to move in the forward direction.”

Los modelos de lenguaje elegidos para comparar son GPT-4o y DeepSeek-o1. Cada uno de los LLMs generar una función de recompensa densa (código disponible en el Anexo I).

4.3. CONFIGURACIÓN

4.3.1. ENTRENAMIENTO MCE-IRL

El experimento se ejecuta en dos fases:

1. Obtención del experto: Se resuelve el problema con la recompensa original y calculamos la *occupancy* experta ρ_e .
2. Entrenamiento MCE: Se entrena una red neuronal para que produzca una política con una *occupancy* ρ_θ que se parezca a la experta. La red representa la función de recompensa $R(s)$.

Para cada uno de los experimentos se registra:

- Tiempo de cómputo del experto.
- Tiempo de cómputo para el paso MCE.
- Memoria usada.

Grid Size	Horizon	Expert Train	MCE Pass	Memory Used
4 x 7	11	0.001 s	1.025 s	727 MB
10 x 10	20	0.002 s	2.356 s	738 MB
20 x 20	40	0.023 s	26.429 s	767 MB
40 x 40	80	0.159 s	3 min 42 s	916 MB
80 x 80	160	2.929 s	4 h 22 min	2.7 GB

Tabla 1. Benchmark de MCE-IRL en *CliffWorld* con distintos tamaños de *grid*

4.3.2. ENTRENAMIENTO AIRL

Para el entrenamiento de AIRL tomamos un aprendizaje PPO, una *BasicShapedRewardNet* y un Adversarial Trainer, con la misma configuración inicial. La configuración está disponible en el Anexo I.

La distinción de cada experimento es el número de trayectorias expertas: 10, 50 y 100. Se usa el experto *SB3* y el experto personalizado que denotaremos como Experto A y Experto B respectivamente.

4.3.3. ENTRENAMIENTO T2R

Para evaluar la calidad de las funciones de recompensa, entrenamos un agente SAC y otro PPO, usando un *wrapper* que envuelve el entorno original sustituyendo la función de recompensa original.

Independientemente de la función de recompensa, ambos PPO y SAC utilizaron la misma configuración. En el caso de PPO también coincide con la de los aprendices de AIRL.

En todos los casos se entrenaron durante 4 millones de *steps*.

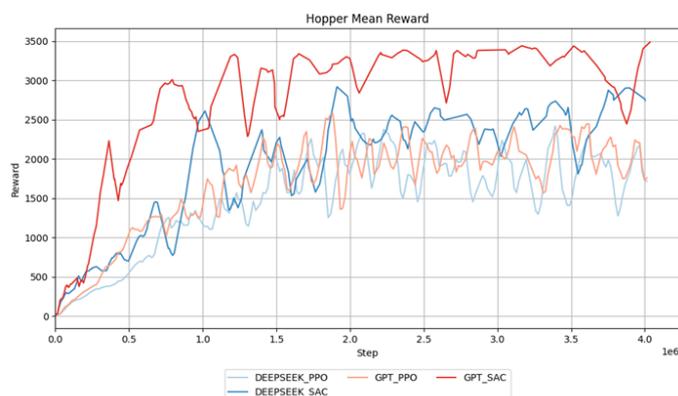


Ilustración i. Evolución de la recompensa durante el entrenamiento de los agentes.

4.4. ANÁLISIS DE RENDIMIENTO

4.4.1. RENDIMIENTO MCE-IRL

La tabla de benchmark muestra que los tiempos de cómputo y el uso de memoria crecen de manera pronunciada con el tamaño del *grid*.

- El caso 4x7 requiere aproximadamente 1 segundo y 727 MB de memoria.
- El caso de 80x80 requiere más de 4 horas y 2.7GB de memoria.

En la sección de visualización 4.5.1 se comparan mapas de calor de la *occupancy* y recompensas aprendidas con las reales.

A pesar de que la recompensa aprendida se aleja de la recompensa original a medida que el entorno se hace más grande y complejo, la *occupancy* aprendida es prácticamente idéntica a la experta. Esto indica que la función de recompensa aprendida es capaz de explicar los resultados de manera fidedigna.

El modelo identifica correctamente los estados objetivos con recompensa positiva y penaliza los estados cercanos al precipicio, mostrando un grado razonable de interpretabilidad.

4.4.2. RENDIMIENTO AIRL

Análisis de los aprendices con el Experto A - 2860

La tabla 1 muestra el rendimiento del aprendiz PPO para cada entrenamiento y las métricas de las trayectorias expertas utilizadas para entrenar. En el caso de las trayectorias expertas todas obtienen un valor similar de recompensa media y desviación como es de esperar.

<i>Nº</i> Trayectorias	Recompensa	PPO Learner
10	2737.00 ± 170.42	2600.56 ± 118.18
50	2726.88 ± 144.27	2506.63 ± 160
100	2712.69 ± 139.77	2106.51 ± 23.26

Tabla 1. Rendimiento aprendices AIRL con Experto A

En cuanto a los aprendices, podemos observar como con 10 y 50 trayectorias se logra reproducir un comportamiento similar al del experto.

Con 100 trayectorias, el rendimiento se degrada significativamente. Esto se debe a que el discriminador alcanza una precisión alta en etapas muy tempranas del entrenamiento. Siendo perfectamente capaz de distinguir las trayectorias expertas de las del aprendiz. La consecuencia es que la señal del gradiente se reduce tanto que el aprendizaje se ralentiza.

Este fenómeno de *overfitting* en el discriminador es típico entornos adversariales. El hecho de que estas trayectorias sean fruto de *exploit* en la función de recompensa también podría explicar esta anomalía.

Se podría mitigar modificando parámetros del entrenamiento, pero se aleja de los objetivos de este trabajo.

Análisis de los aprendices con el Experto B - 3500

Al ser este experto inestable se han recolectado aquellas trayectorias que superan los 3500 de *reward*.

Nº Trayectorias	Recompensa	PPO Learner
10	3518.09 ± 161.21	3436.84 ± 25.06
50	3566.64 ± 48.27	3250.01 ± 243.69
100	3551.29 ± 75.59	3511.95 ± 11.40

Tabla 2. Rendimiento aprendices AIRL con Experto B

En este caso, todos los aprendices PPO logran generar trayectorias con una recompensa muy alta.

Podemos notar que el experto que usa 50 trayectorias tiene una desviación en sus trayectorias bastante más alta ± 243. Tras analizar los registros durante entrenamiento no se ha podido apreciar nada irregular que explique esta inestabilidad.

El gran logro que podemos sacar es que el agente de 100 trayectorias obtiene unas métricas extremadamente buenas $3511,95 \pm 11,40$.

En el momento de realizar este trabajo, nadie ha publicado ningún agente que se acerque a estos valores. Incluso el agente que usa solo 10 trayectorias obtiene grandes resultados.

Análisis de las *Reward Nets* aprendidas

En todos los experimentos de AIRL se parte de una misma configuración de la red neuronal que se aprende.

La red se basa en una *BasicShaped* de la librería *Imitation* compuesta por una red básica g_θ que trata de representar la función de recompensa original y un término de *shaping* h_ϕ para dar estabilidad durante el entrenamiento.

$$f_{\theta,\phi}(s, a, s') = g_\theta(s, a) + \gamma h_\phi(s') - h_\phi(s)$$

Para poder hacer una evaluación justa de las funciones aprendidas se recolectan 15 trayectorias del Experto A (2800), 15 del Experto B (3500) y 15 de una política aleatoria.

Se evalúan las trayectorias en cada una de las 6 funciones de recompensas aprendidas y se compara con la recompensa original.

Reward Net	Spearman	Kendall	Cohen A-B	Cohen A-Rnd	Cohen B-Rnd
A_10	0.490	0.314	-5.73	6.98	5.01
A_50	0.514	0.356	-10.27	16.84	12.85
A_100	0.519	0.367	-4.80	7.35	4.95
B_10	0.846	0.711	25.37	1.18	11.59
B_50	0.436	0.251	75.28	-3.40	7.12

B_100	0.889	0.754	22.47	2.84	6.96
-------	-------	-------	-------	------	------

Tabla 2. Métricas de las *reward nets* aprendidas mediante AIRL

Las *reward nets* obtenidas usando trayectorias del experto A (2800) obtienen métricas de *Spearman* y *Kendall* bajas, 0.5 y 0.36 respectivamente. Esto indica que no conservan bien el orden relativo de las trayectorias entre la recompensa original y la predicha por la red.

Los valores positivos en d de Cohen muestran que son capaces de distinguir como mejores las recompensas que provienen de un experto, ya sea A o B, que las que vienen de la política aleatoria.

Sin embargo, el valor es negativo entre A y B. La interpretación de este resultado es que las *reward nets* consideran que el experto A (2800) es mejor que el experto B (3500), lo cual es erróneo.

Este resultado sugiere la existencia de *overfitting*. Las *reward nets* solo explican la zona de la función de recompensa original que justifica el comportamiento del Experto A, sin generalizar más allá.

Las redes entrenadas con trayectorias del Experto B en general muestran métricas mejores. Las obtenidas con 10 y 100 trayectorias alcanzan valores superiores a 0.80 en *Spearman* y a 0.7 en *Kendall*.

Por tanto, distingue correctamente la jerarquía $B > A > \text{Random}$, mostrando una mayor capacidad de generalización.

En cambio, el experimento con 50 trayectorias parece haber sufrido un problema de sobreajuste. Muestra malos resultados en *Spearman* y *Kendall* (0.43 y 0.25), valores extremos en Cohen e incluso considera que las trayectorias aleatorias son mejores que las del Experto A.

Reentrenamiento

Incluso en los mejores casos del Experto B con 10 o 100 trayectorias, las *reward nets* no pueden considerarse una réplica fiel de la *reward* original.

La única forma de comprobarlo es reentrenar políticas nuevas desde cero utilizando estas *reward nets* como función de recompensa. Se implementaron y en ningún caso se logró alcanzar un experto que superara un retorno promedio de 300 sobre la *reward* original.

Esto demuestra que las redes obtenidas mediante AIRL son capaces de representar bien las trayectorias que han visto. Pero en espacios de observaciones grandes no son capaces de guiar a un agente hacia un comportamiento óptimo.

4.4.3. RENDIMIENTO T2R

Para evaluar las funciones de recompensa cada agente genera 50 trayectorias. Se ha optado por calcular la recompensa promedio sobre la función de recompensa en la que su agente fue entrenado, sobre la función de recompensa original y la que usa el otro experto.

De forma complementaria, se calcula la métrica de *Spearman* con las trayectorias del experto seleccionado entre la recompensa original y la recompensa sobre la que fue entrenado.

<i>Expert</i>	<i>Ground Truth</i>	<i>GPT</i>	<i>DeepSeek</i>	<i>Spearman</i>
GPT_PPO	3228.5 ± 582.5	3390.8 ± 414.6	2865.1 ± 440.1	± 0.998
GPT_SAC	3542.6 ± 2.4	3540.7 ± 2.3	3039.6 ± 2.3	± 0.999
DS_PPO	2964.5 ± 525.6	3211.5 ± 387.6	2670.2 ± 411.0	± 0.997
DS_SAC	2869.5 ± 903.3	2787.1 ± 1033.5	2491.2 ± 783.1	± 0.993

Tabla 3. Evaluación de los agentes entrenados con funciones de recompensas de distintos LLMs.

El valor de la columna *ground-truth* representa la *reward* obtenida por el experto en el entorno con la función de recompensa original. Aquí destaca con diferencia el agente SAC entrenado con la *reward* generada por GPT-o4 con una media de 3542.6 sobre la *reward* original, pero principalmente impresiona su desviación típica que es de tan solo 2.4 mostrando una estabilidad enorme.

Al evaluar estas trayectorias sobre la función de recompensa de GPT-o4 y de DeepSeek también se obtienen valores muy altos superando los 3000 en todas. Esto denota que todas las funciones de recompensa reconocen de forma similar un comportamiento excelente del agente.

El agente GPT-PPO también consigue buen rendimiento 3228 pero con una gran inestabilidad con desviaciones de 582.

Por otro lado, tanto el agente SAC como PPO de *DeepSeek* se quedan a las puertas de los 3000 sobre la *ground truth reward* pero también con variaciones exageradas de 903 y 525 respectivamente.

Las métricas de *Spearman* se calculan con las trayectorias expertas de cada agente y comparando su *reward* en la función original y en la usada para entrenar el agente.

En todos los casos, obtenemos unos valores altísimos, cercanos al 1.0. Por lo tanto, en términos de cómo de similar clasifican trayectorias las funciones de recompensa, tanto GPT como *DeepSeek* son muy similares a la original.

Las diferencias sustanciales se dan en la estabilidad a la hora de aprender sobre ellas, siendo *DeepSeek* claramente más inestable y GPT siendo inestable en PPO, pero sorprendentemente estable con SAC.

La escala de las recompensas difiere ligeramente, es curioso que parece que el máximo en GPT coincide con la función original, en torno a los 3500 mientras que *DeepSeek* considera el comportamiento excelente en el 3000.

4.5. VISUALIZACIÓN

4.5.1. GRÁFICAS MCE-IRL

Se comparan los mapas de calor de *occupancy* y recompensas aprendidas frente a las originales para cada experimento.

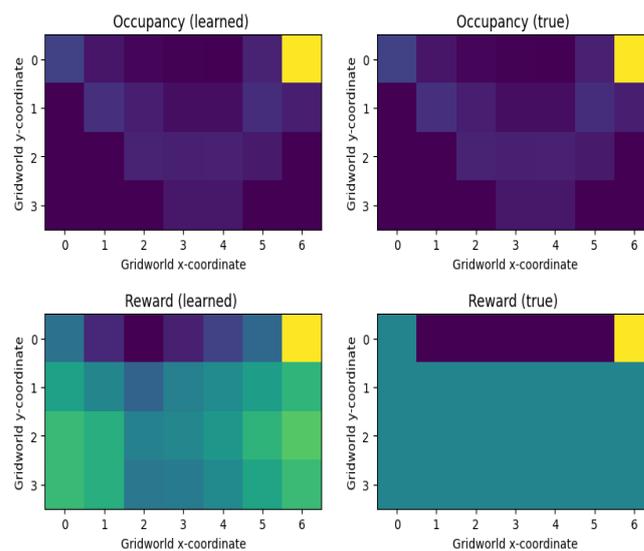


Ilustración j. *Occupancy* y *Reward* aprendidas vs Real (4 x 7)

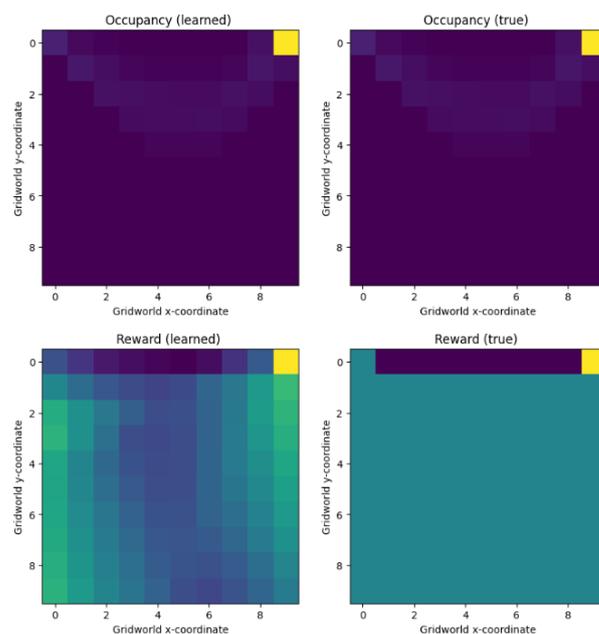


Ilustración k. *Occupancy* y *Reward* aprendidas vs Real (10 x 10)

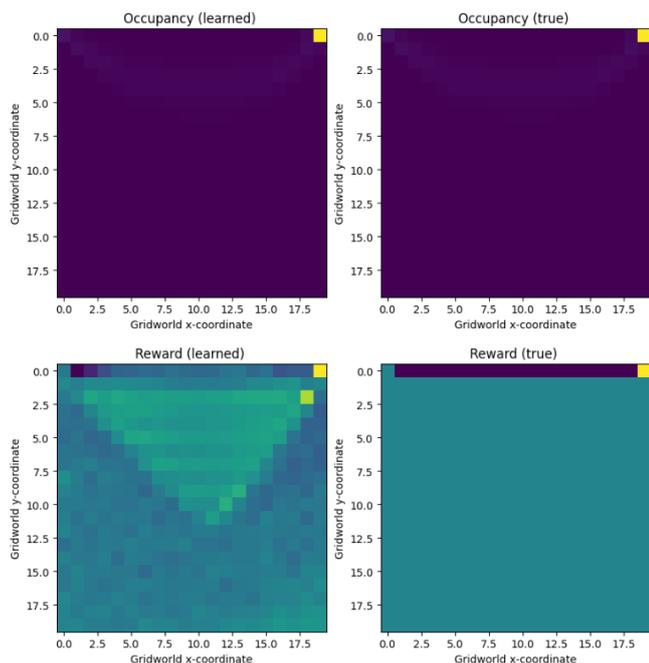


Ilustración l. Occupancy y Reward aprendidas vs Real (20 x 20)

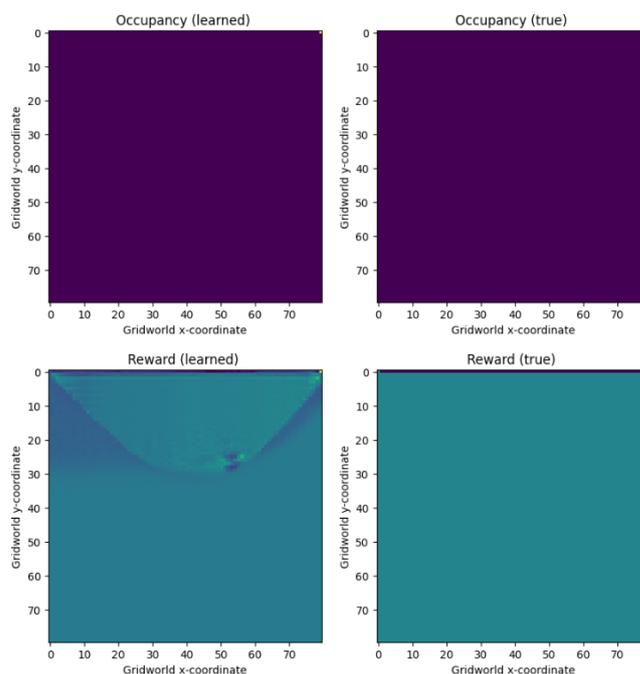


Ilustración n. Occupancy y Reward aprendidas vs Real (80 x 80)

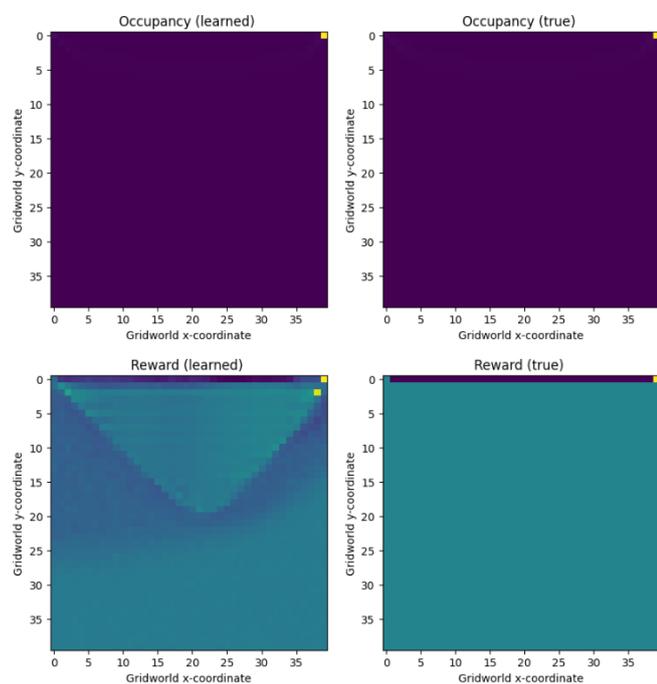


Ilustración m. Occupancy y Reward aprendidas vs Real (40 x 40)

4.5.2. GRÁFICAS AIRL

La siguiente gráfica muestra el proceso de entrenamiento de los aprendices de todos los experimentos AIRL. La recompensa que aparece es la que generan las trayectorias sobre la función de recompensa original.



Ilustración o. Gráfica de la evolución de los aprendices durante el entrenamiento adversarial.

5. RESULTADOS

El algoritmo de *Maximum Causal Entropy* es capaz de recuperar funciones de recompensas interpretables, mantener medidas de ocupación coherentes con las expertes incluso con *grids* grandes. Sin embargo, al escalar en complejidad se sufre un aumento significativo en tiempos de cómputo y memoria necesaria.

Adversarial IRL muestra unos resultados ambiguos. En términos de imitación es capaz de lograr unos muy buenos resultados independientemente del número de trayectorias. Esto es de especial interés si queremos aprender un comportamiento cuyos ejemplos expertos sean costosos de adquirir.

Sin embargo, en tareas de *Reward Learning* puro sirve más como un detector de comportamientos de calidad que como una función de recompensa generalizable. Esto se concluye del hecho de que no son capaces de guiar a un agente desde cero al objetivo óptimo.

Las funciones de recompensa generadas mediante LLMs permiten entrenar agentes que reconocen y reproducen comportamientos óptimos, destacando la capacidad de GPT-4o de producir recompensas densas y estables. Esto refuerza la utilidad de los LLMs para corregir o reemplazar funciones de recompensa mal definidas.

En el entorno original se han obtenidos grandes resultados. Consideramos que la gran contribución es haber conseguido expertos con un rendimiento nunca visto en el entorno modificado de *Hopper Seals*.

6. CONCLUSIONES Y TRABAJOS FUTUROS

Este trabajo aborda un estudio comparativo de algoritmos de *Reward Learning*. Se ha presentado el estado del arte en el campo y seleccionado tres enfoques representativos.

Se presenta las definiciones formales de *Maximum Causal Entropy IRL*, *Adversarial IRL* y *Text2Reward*. Se han configurado entornos de evaluación y métricas para poder analizar los puntos fuertes y debilidades de cada uno de ellos.

MCE-IRL demuestra una solidez teórica palpable con los experimentos. Sin embargo, está muy limitado a entornos simples, tabulares y muy definidos. Además, presenta unos costes computacionales que no permiten extrapolarlo a entornos de mayor dimensionalidad.

Por otro lado, AIRL se presenta como una gran opción en el campo de la imitación. Se consiguen expertos muy capaces de reproducir comportamientos expertos. La debilidad está en su capacidad de generar una red de recompensa que realmente modele el comportamiento experto. Al ser tan dependiente de las trayectorias dadas y el poder utilizarlo en entornos más complejos hace que la recompensa aprendida está bien definida solo en los estados próximos al comportamiento experto. Esto hace que realmente no pueda guiar a agentes desde cero hasta la política deseada.

El gran resultado es T2R. Este enfoque tan reciente ha permitido demostrar una capacidad impresionante para modelar funciones de recompensa densas, interpretables y óptimas. Requiere la preparación de un *prompt* que defina al detalle el entorno, siendo un proceso tedioso. Pero la calidad de las funciones de recompensa que se obtienen con modelos de lenguaje como GPT-4o son excepcionales.

Como trabajo futuro se podrían añadir otros de los algoritmos nombrados en el estado del arte. Aunque el potencial real lo vemos en un análisis más profundo de técnicas que usen modelos de lenguaje natural.

En relación con Goal Conditioned Learning quedaría implementar los algoritmos y realizar un análisis comparativo.

7. BIBLIOGRAFÍA

- [1] Ng, A. Y., & Russell, S. J. (2000). Algorithms for inverse reinforcement learning. *Proceedings of the 17th International Conference on Machine Learning (ICML)*, 663–670.
- [2] Ziebart, B. D., Maas, A. L., Bagnell, J. A., & Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, 1433–1438.
- [3] Ramachandran, D., & Amir, E. (2007). Bayesian inverse reinforcement learning. *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2586–2591.
- [4] Fu, J., Luo, K., & Levine, S. (2018). Learning robust rewards with adversarial inverse reinforcement learning. *International Conference on Learning Representations (ICLR)*.
- [5] Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 49–58. PMLR 48.
- [6] Christiano, P. F., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems (NeurIPS)*, 4299–4307.
- [7a] Brown, D. S., Goo, W., Nagarajan, P., & Niekum, S. (2019). Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations (T-REX). *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR 97.
- [7b] Brown, D. S., Goo, W., & Niekum, S. (2020). Better-than-demonstrator imitation learning via automatically-ranked demonstrations (D-REX). *Conference on Robot Learning (CoRL)*. PMLR 100.
- [8] Xie, T., Zhao, S., Wu, C. H., Liu, Y., Luo, Q., Zhong, V., Yang, Y., & Yu, T. (2024). Text2Reward: Reward shaping with language models for reinforcement learning. *International Conference on Learning Representations (ICLR)*.
- [9] Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., & Anandkumar, A. (2023). EUREKA: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*.
- [10] Ziebart, B. D. (2010). *Modeling purposeful adaptive behavior with the principle of maximum causal entropy* (Doctoral dissertation). Carnegie Mellon University.
- [11] Schaul, T., Horgan, D., Gregor, K., & Silver, D. Universal Value Function Approximators. *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, PMLR 37, 1312–1320, 2015.
- [12] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., & Zaremba, W. Hindsight Experience Replay. *Advances in Neural Information Processing Systems (NeurIPS 2017)*, 2017
- [13] Florensa, C., Held, D., Geng, X., & Abbeel, P. Automatic Goal Generation for Reinforcement Learning Agents (Goal GAN). *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, PMLR 80, 2018.
- [14] Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., & Levine, S. Visual Reinforcement Learning with Imagined Goals (RIG). *Advances in Neural Information Processing Systems (NeurIPS 2018)*, 2018

ANEXO I

<i>Algoritmo</i>	<i>PPO</i>	<i>SAC</i>
Learning Rate	3e-4	3e-4
Batch Size	256	256
Gamma	0.99	0.99
Ent. Coef	0.01	Auto
N. Steps	2048	None
Net Size	[256, 256]	[256, 256]
Tau	None	0.005
Buffer Size	None	1_000_000

Tabla 4. Configuraciones agentes PPO y SAC.

<i>Basic Shaped Reward Net</i>	<i>Values</i>
Input Layer Norm	RunningNorm
Activation	Tanh
Reward Hidden Size	[64, 64]
Potential Hidden Size	[32, 32]
Use State	True
Use Action	True

Use Next State	False
Use Done	False
Discount Factor	0.99

Tabla 5. Configuración reward nets.

<i>Adversarial Trainer</i>	<i>Values</i>
Demo Batch Size	2048
Gen Replay Buffer	512
Disc Updates per Round	12

Tabla 6. Configuración Adversarial Trainer.

Prompt Template para el entorno de Hopper

HOPPER PROMPT TEMPLATE

You are an expert in robotics locomotion, reinforcement learning and code generation. We are going to control a Hopper to complete given tasks.

The hopper is a two-dimensional one-legged figure that consist of four main body parts: the torso at the top, the thigh in the middle, the leg in the bottom, and a single foot on which the entire body rests. All adjacent body parts are connected by hinges. The action space of the Hopper is a Box(-1, 1, (3,), float32), with action[0] indiactes Torque applied on the thigh rotor, action[1] indiactes Torque applied on the leg rotor, and action[2] indiactes Torque applied on the foot rotor

Now I want you to help me write a reward function of reinforcement learning. I'll give you the attributes of the environment and Hopper itself. You can use these class attributes to write the reward function.

```
class BaseEnv (gym.Env) :
    self.hopper : HopperRobot
    self.dt : float # the time between two
actions, in seconds, default is 0.008 s
```

```
def __init__(self): # initialize state of
the Hopper
    self.hopper.top.position_x = 0.0
    self.hopper.top.position_z = 1.25
    self.hopper.top_joint.angle = 0.0
    self.hopper.thigh_joint.angle = 0.0
    self.hopper.leg_joint.angle = 0.0
    self.hopper.foot_joint.angle = 0.0
```

```
class HopperRobot:
    self.top : SlideJoint # indicate the top
endpoint of the Hopper
    self.top_joint: HingeJoint # indicate the
top hinge joint of the Hopper
    self.thigh_joint : HingeJoint # indicate
the hinge joint between Hopper torso and
Hopper thigh
    self.leg_joint : HingeJoint # indicate
the hinge joint between Hopper thigh and
Hopper leg
    self.foot_joint : HingeJoint # indicate
the hinge joint between Hopper leg and Hopper
foot
```

```
class SlideJoint:
    self.position_x : float # x-coordinate
position in the world frame, in meters
    self.position_z : float # z-coordinate
position in the world frame, in meters
    self.velocity_x : float # x-coordinate
velocity in the world frame, in meters per
second
    self.velocity_z : float # z-coordinate
velocity in the world frame, in meters per
second
```

```
class HingeJoint:
    self.angle : float # joint angle value of
the HingeJoint, in radians
    self.angular_velocity : float # joint
angular velocity of the HingeJoint, in
radians per second
```

Here we define that the two-dimensional hopper can only move in the x-coordinate, with positive x-coordinate to be the forward direction and negative x-coordinate to be the backward direction.

I want it to fulfill the following task:
{Control the Hopper to back flip}
1. please think step by step and tell me what does this task mean;

2. then write a function that format as `def compute_dense_reward(self, action) -> float` and returns the `reward : float` only.
3. Take care of variable's type, never use the function of another python class and do not use ungiven variables.
4. When you writing code, you can also add some comments as your thought.
5. Do not give too much weight on regularization or penalty. And give different weight to different reward term, according to its measurement and importance.

Función de recompensa generada por DeepSeek-01

```
import math # For cosine calculation

def compute_dense_reward(self, obs, action) -
> float:
    # === Desempaquetar observación ===
    (_, # Ignorado (puede
ser torso z-pos)
    top_ang, # Ángulo del torso
    thigh_ang, # Ángulo del muslo
    leg_ang, # Ángulo de la pierna
    foot_ang, # Ángulo del pie
    vel_x, # Velocidad
horizontal (forward)
    _, # Ignorado
(posiblemente vel_y)
    ang_vel_top, # Velocidad angular
del torso
    ang_vel_thigh,
    ang_vel_leg,
    ang_vel_foot) = obs

    # === Recompensa por velocidad hacia
adelante ===
    forward_vel = vel_x

    # === Recompensa por mantener el torso
erguido ===
    upright_reward = math.cos(top_ang) # +1
erguido, -1 invertido

    # === Penalización por altura si está
debajo de 1.25m ===
    # Suponiendo que obs[0] es la altura del
torso
    current_height = obs[0]
```

```

height_penalty = max(0, 1.25 -
current_height) ** 2

# === Penalización por torque (esfuerzo
de control) ===
torque_penalty = 0.001 * (action[0]**2 +
action[1]**2 + action[2]**2)

# === Recompensa total ponderada ===
reward = (
    1.0 * forward_vel +      # Prioridad
alta al avance
    0.5 * upright_reward -  #
Bonificación por estabilidad
    0.1 * height_penalty -  #
Penalización si cae
    torque_penalty         #
Penalización por esfuerzo
)
return reward

```

```

# 1. Forward velocity reward (primary
goal)
reward_forward = 1.0 * vel_x

# 2. Alive bonus (small constant for not
falling)
reward_alive = 1.0

# 3. Control cost (penalize large torques
lightly)
ctrl_cost = 0.001 * (action[0]**2 +
action[1]**2 + action[2]**2)

# 4. Torso uprightiness cost (keep top
hinge near 0)
angle_cost = 0.1 * (top_ang**2)

# Total reward
reward = reward_forward + reward_alive -
ctrl_cost - angle_cost
return float(reward)

```

Función de recompensa generada por GPT-4o

```

import numpy as np

def compute_dense_reward(self, obs:
np.ndarray, action: np.ndarray) -> float:
    """
    Compute dense reward from Gymnasium
observation and action.

    Args:
        obs (ndarray, shape=(11,)): [z,
top_ang, thigh_ang, leg_ang, foot_ang,
vel_x,
vel_z, ang_vel_top, ang_vel_thigh,
ang_vel_leg, ang_vel_foot]
        action (ndarray, shape=(3,)):
[torque_thigh, torque_leg, torque_foot]

    Returns:
        reward (float)
    """
    # Unpack observation
    (_, top_ang, thigh_ang, leg_ang,
foot_ang,
vel_x, _, ang_vel_top, ang_vel_thigh,
ang_vel_leg, ang_vel_foot) = obs

```

Los siguientes repositorios contienen el trabajo desarrollado. Contienen además vídeos con trayectorias expertas.

https://github.com/CptAdolito/Analisis_Reward_Learning