

UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos



Iterative self-refined code LLM for spacecraft landing

TRABAJO DE FIN DE MÁSTER

Presentado para la obtención del título de Máster por:

Adrián Gayo Andrés

Madrid, 2024



UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingeniería de Sistemas
Informáticos



Máster en Aprendizaje Automático y Datos Masivos

Iterative self-refined code LLM for spacecraft landing

TRABAJO DE FIN DE MÁSTER

Presentado para la obtención del título de Máster por:

Adrián Gayo Andrés

Bajo la supervisión de:
Dr. Víctor Rodríguez Fernández

Madrid, 2024

A mi familia y amigos, pilares fundamentales en mi día a día.

Agradecimientos

Agradezco al grupo de investigación ARCLab del Massachusetts Institute of Technology (MIT) por el apoyo económico para el uso de la API de OpenAI, así como por el acceso a otros recursos de computación. También valoro profundamente la puesta en común de avances en diversas líneas de investigación y la colaboración brindada.

Abstract

This project proposes to use the knowledge base of large language models (LLMs) to provide an alternative to traditional reinforcement learning models in spacecraft landing, which have a major weakness due to the high cost of training. Rather than simply making decisions about actions at each point in time, the challenge is to encode planned decision-making into a function using a code LLM, enabling near real-time responses, a crucial capability in many situations, but also a considerable challenge for current LLM models.

The proposed approach involves an iterative process of self-refinement by the model, based on the generated code logs and using OpenAI API wizards. Different approaches will be explored to address the problem, including Zero-shot prompting, Few-shot prompting and an evolutionary model composed of a population of models in charge of completing the iterative task. This project aims to improve the efficiency and effectiveness of reinforcement learning systems by integrating pre-existing knowledge in LLMs and advanced refinement techniques.

The results are promising from the point of view of the capabilities that LLM models can perform as analysts and planners of landing situations. They highlight the insights and clues extracted for their feedback on code generation from a very limited number of examples. However, they also show room for improvement when coding decision making and open the door to test better performing models for coding and problem analysis/abstraction tasks.

The line of research is challenging and has potential for growth on many fronts, from the model, to the landing agent engine, to the refinement of communication with the model itself. It also opens the door to including traditional machine learning methodologies such as evolutionary algorithms in the work with LLMs by proposing the improvement of results through collaboration in a population of assistants that must perform a given task.

Resumen

Este proyecto propone utilizar la base de conocimientos de los grandes modelos de lenguaje (LLMs) para ofrecer una alternativa a los modelos tradicionales de aprendizaje por refuerzo en el aterrizaje de naves espaciales, que presentan una gran debilidad debido al elevado coste del entrenamiento. En lugar de limitarse a tomar decisiones sobre las acciones en cada momento, el reto consiste en codificar la toma de decisiones de forma planificada en una función mediante un code LLM, lo que permite respuestas casi en tiempo real, una capacidad crucial en muchas situaciones, pero también un reto considerable para los modelos LLM actuales.

El enfoque propuesto implica un proceso iterativo de auto-refinamiento por parte del modelo, basado en los registros de código generados y utilizando los Assistants API de OpenAI. Se explorarán diferentes enfoques para abordar el problema, incluyendo Zero-shot, Few-shot y un modelo evolutivo compuesto por una población de modelos encargados de completar el proceso iterativo. Este proyecto pretende mejorar la eficiencia y eficacia de los sistemas de aprendizaje por refuerzo integrando conocimientos preexistentes en los LLM y técnicas avanzadas de refinamiento.

Los resultados arrojados son prometedores desde el punto de vista de las capacidades que los modelos LLM pueden desempeñar como analistas y planificadores de situaciones de aterrizaje. Destacan la comprensión y claves extraídas para su retroalimentación en la generación de código a partir de un número muy limitado de ejemplos. No obstante, se muestra también un margen de mejora en la codificación de la toma de decisiones, y se abre la puerta a probar modelos con mejor desempeño para las tareas de codificación y análisis o abstracción del problema.

La línea de investigación es desafiante y presenta capacidad de mejora en numerosos frentes, desde el modelo, motor del agente encargado del aterrizaje, hasta el refinamiento de la comunicación con el propio modelo. Asimismo, se abre la puerta a incluir metodologías tradicionales del aprendizaje automático, como los algoritmos evolutivos, al trabajo con LLM, proponiendo la mejora de resultados mediante la colaboración en una población de asistentes que deben desempeñar una tarea determinada.

Índice general

Agradecimientos	iii
Abstract	v
Resumen	vi
Índice de figuras	viii
Índice de tablas	ix
Índice de Códigos	xii
Abreviaturas y acrónimos	xiv
1 Introduction	1
2 Estado del arte	5
2.1 Preliminares	5
2.1.1 Large language models	8
2.2 Trabajo relacionado	9
3 Materiales y métodos	13
3.1 Assistants API	13
3.2 Ejecución de entorno Lunar Lander	14
3.3 Prompt engineering	14
3.3.1 Registros	14
3.3.2 Prompts de sistema	17
3.3.3 Prompts de usuario	17
3.4 Function calling	18
3.5 JSON mode	19
3.6 Proceso iterativo	19
3.7 Agentes ejemplo	20
3.8 Zero-shot prompting	20
3.9 Few-shot prmpting	20
3.10 Algoritmo evolutivo	21
4 Resultados	23
4.1 Comparativa entre modelos	23
4.1.1 Ejecuciones con GPT-3.5-Turbo	23
4.1.2 Ejecuciones con GPT-4o	25
4.2 JSON mode	28

4.3	Evolutivo	29
4.4	Comparativa	30
4.5	Mejor resultado	30
5	Discusión	35
5.1	Análisis de los resultados	35
5.2	Implicaciones y relevancia	36
5.3	Limitaciones	37
5.4	Recomendaciones y futuras líneas de investigación	37
6	Conclusiones	39
	Bibliografía	41
	Anexos	43
.1	Código inicial	43
.2	Ejemplos de prompt	44

Índice de figuras

1.1	Diagrama del bucle iterativo del proyecto	3
2.1	Entorno de simulación Lunar Lander	6
2.2	Esquema de funcionamiento de Language Models are Spacecraft operators sobre el entorno KSPDG. ([15])	9
2.3	Esquema de funcionamiento de Eureka [5] para la optimización de funciones de recompensa en distintos entornos.	11
4.1	Razonamiento previo del modelo GPT 3.5 turbo en la primera iteración. . .	25
4.2	Razonamiento previo del modelo GPT-4o.	26
4.3	Razonamiento en la mejora iterativa del modelo GPT-4o.	27
4.4	Evolución de la puntuación de los asistentes con mejor puntuación.	31
4.5	Cambios elaborados por el modelo en el código con mejor resultado.	33
4.6	Cambios elaborados en el código agente en la iteración 4 por el individuo que ha conseguido obtener la mejor solución de la población.	34

Índice de tablas

4.1	Tiempos de ejecución aproximados de ambas metodologías para 5 iteraciones (sin errores de compilación).	29
4.2	Scores finales obtenidos por cada agente en 5 iteraciones y su score medio. . .	29
4.3	Scores finales obtenidos por cada agente en 5 iteraciones y su score medio en la mejor solución evolutiva obtenida.	30
4.4	Comparación de valor medio, mejor solución y mejora para las metodologías utilizadas.	30
4.5	Resultados promedio y porcentaje de agentes que superan los umbrales de puntuación tras 100 ejecuciones.	31

Índice de Códigos

3.1	Esquema JSON para los datos de registro del entorno de aterrizaje de la nave espacial. Basado en el formato de la respuesta del método step.	15
3.2	Esquema de la función "store_code_in_file" proporcionada al modelo como herramienta.	18
3.3	Esquema de respuesta solicitada al modelo con formato JSON.	19
3.4	Prompt para el modelo con ejemplos exitosos y el código inicial	20
4.1	Ejemplo de código generado por el agente basado en el modelo GPT-3.5-turbo	24
4.2	Ejemplo de código generado por GPT-4o	28
4.3	Ejemplo del código inicial de la mejor solución encontrada.	31
1	Código inicial que recibe el modelo.. . . .	43
2	Prompt para el modelo con Zero shot	44
3	Prompt de fases de desarrollo anteriores	45
4	Prompt utilizado en la generación de agente con JSON mode	46
5	Retroalimentación tras cada iteración	47
6	Retroalimentación tras cada iteración con el engoque genético	48

Abreviaturas y acrónimos

UPM Universidad Politécnica de Madrid

LLM Large Language Model

API Application Programming Interface

RL Reinforcement learning

KSPDG Kerbal Space Program Differential Games

Capítulo 1

Introduction

Desde que OpenAI lanzó al público su ampliamente conocido ChatGPT a finales de 2022, el mundo ha tomado conciencia del potencial de los grandes modelos de lenguaje basados en la revolucionaria arquitectura Transformer[19]. Este avance ha impulsado el desarrollo de modelos cada vez más grandes y capaces, inaugurando lo que claramente puede considerarse la carrera espacial de nuestra era. Estas herramientas, cuyo rendimiento es notablemente elevado y cuyo alcance total aún se desconoce, han despertado un gran interés entre grupos de investigación de diversos ámbitos y disciplinas. El uso de los grandes modelos del lenguaje (LLMs) promete innovaciones y mejoras significativas en una variedad de aplicaciones. En particular, en la navegación y la exploración espacial también se ha identificado en estos modelos una oportunidad para mejorar y ofrecer alternativas a los paradigmas actuales.

Este trabajo busca continuar la línea de investigación iniciada en el artículo *Language Models are Spacecraft Operators*[15], en el cual se utilizan LLMs como agentes autónomos responsables de la toma de decisiones dentro del entorno de simulación Kerbal Space Program Differential Games suite (KSPDG). El esquema de funcionamiento consiste en proporcionar al modelo un prompt que refleje el estado actual de la nave y el entorno. En base a esta información, el modelo será el encargado de decidir la acción más conveniente (como el grado de potencia en los motores de la nave). Tras ejecutar la acción escogida, el estado del entorno cambiará, y será necesario repetir este proceso de manera iterativa.

En ambos trabajos, el objetivo es presentar una alternativa al paradigma tradicional del aprendizaje por refuerzo, aprovechando el conocimiento preexistente en los LLMs y así evitar comenzar desde cero en el aprendizaje del agente. Tradicionalmente, los problemas de aprendizaje por refuerzo requieren un gran número de ejecuciones y redes neuronales cada vez más complejas para su entrenamiento. En contraste, este enfoque busca reducir la necesidad de entrenamientos extensivos utilizando la base de conocimientos ya presente en los LLMs. Además, se pretende abordar estos problemas de manera más cercana al enfoque humano, alimentando el modelo y obteniendo salidas con lenguaje natural en lugar de la metodología tradicional utilizada en los modelos del estado del arte en aprendizaje por refuerzo para agentes autónomos en tareas de navegación.

Este proyecto introduce una variación respecto al propuesto en el artículo mencionado. En

lugar de que el agente tome decisiones en tiempo real, lo que requiere modelos capaces de ofrecer inferencia inmediata, el agente codifica su razonamiento y toma de decisiones en una función. Este cambio permite tiempos de respuesta mucho más competitivos y la utilización de modelos más potentes, dado que el tiempo de respuesta ya no es un aspecto crítico. El proceso propuesto comienza con un código Naive y los registros de su ejecución, que deben ser mejorados de forma iterativa por el agente en base a los resultados ofrecidos por el nuevo código generado. Este proceso, denominado **Self-Refinement**, busca demostrar una solución viable y aplicable a diversos problemas de aterrizaje y otros entornos. La metodología se fundamenta en el uso de registros del entorno, formateados de manera estructurada y en lenguaje natural legible, para mejorar la efectividad y adaptabilidad del modelo. El proceso descrito se ilustra en la figura 1.1

La motivación de este proyecto radica en lograr la adaptabilidad del modelo, que opera como agente espacial, a diversos entornos utilizando su conocimiento base, las indicaciones recibidas y los resultados de sus ejecuciones. Sin embargo, el desafío planteado es más complejo, ya que la codificación de tomas de decisión supone un reto considerablemente mayor que la simple toma de decisiones basada en observaciones. Esta codificación implica un análisis de prioridades, la consideración del efecto del código sobre el entorno, la previsión de posibles situaciones adversas y la generalización de la toma de decisiones para asegurar el éxito independientemente de las condiciones, codificándolo de la mejor manera posible.

En definitiva, este proyecto requiere una mayor abstracción del problema y unas capacidades de razonamiento de alta exigencia. A través de este enfoque, podremos establecer un baremo sobre el estado actual de los modelos en tareas de complejidad y exigencia.

Todo el material utilizado y desarrollado en este proyecto se encuentra disponible en el siguiente repositorio: [GitHub - TFM LLM Landing Self-Refinement](#).

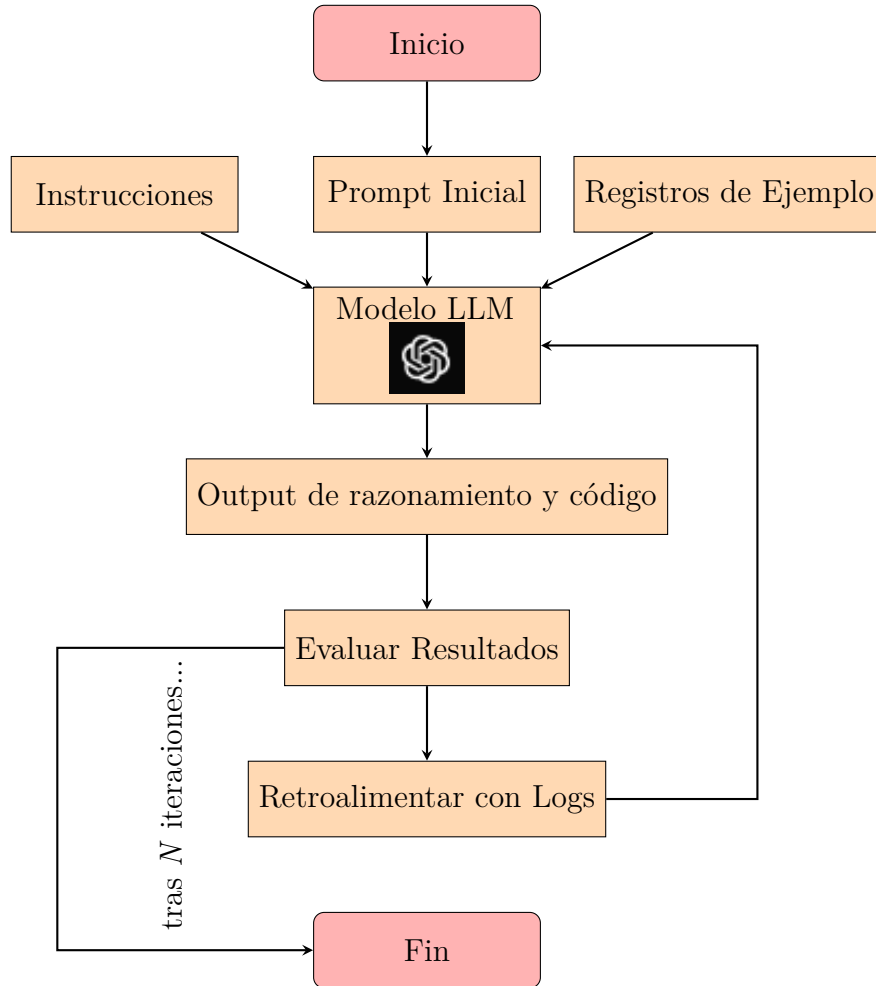


Figura 1.1: Diagrama del bucle iterativo del proyecto

Capítulo 2

Estado del arte

2.1 Preliminares

Tras considerar varias opciones, se ha decidido utilizar el entorno *LunarLander* [11] de Gymnasium (propiedad de OpenAI) como el entorno de simulación para probar el enfoque propuesto (ver en la figura 2.1). Este entorno es ampliamente utilizado, lo que permite comparar resultados con otros modelos y obtener registros de aterrizajes exitosos para alimentar al modelo. Sin embargo, el objetivo es que el modelo sea capaz de adaptarse a cualquier tipo de entorno y aborde el problema como uno de aterrizaje planetario. Por lo tanto, la popularidad del entorno de simulación podría plantear dificultades en la codificación y en la forma de abordar el desafío. Esto se debe a que el modelo podría recurrir a metodologías tradicionales, contra las que se pretende competir, debido a la gran cantidad de ejemplos aprendidos previamente, que posiblemente formen parte del conjunto de entrenamiento de un gran modelo de lenguaje como, por ejemplo, ChatGPT..

El entorno se caracteriza tener 4 posibles acciones que puede tomar el agente:

- **0:** Apagar todos los motores.
- **1:** Encender el motor izquierdo.
- **2:** Encender el motor central.
- **3:** Encender el motor derecho.

Lunar Lander utiliza la interfaz de Gymnasium, creado principalmente para el desarrollo y benchmarking de algoritmos de aprendizaje por refuerzo, la cual se basa en un entorno que ejecuta la acción solicitada mediante el método *step* y devuelve la información resultante del entorno. Además, incorpora una serie de funcionalidades útiles para el desarrollo de este proyecto, como la grabación de vídeo de los episodios.

Se puede observar que la complejidad no es elevada, ya que las acciones disponibles son binarias: un motor puede estar encendido o apagado, pero no es posible aplicar un porcentaje específico de potencia como si se tratara de un pedal. De manera similar, solo se puede realizar una acción entre las cuatro posibles, lo que implica que no más de un motor puede estar

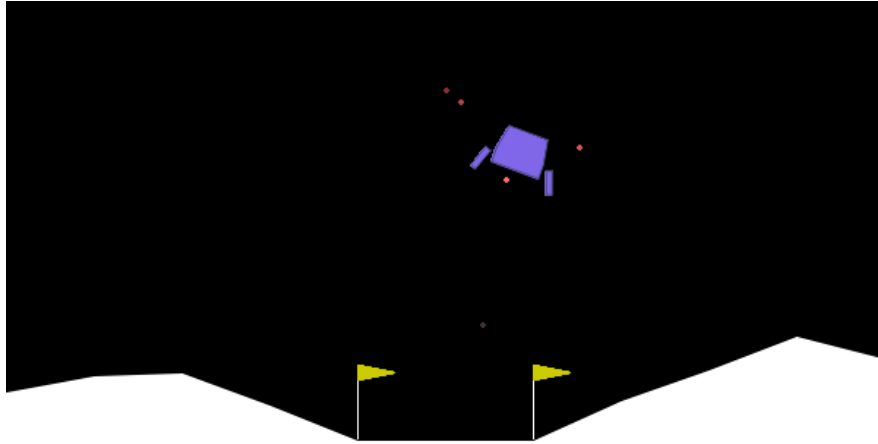


Figura 2.1: Entorno de simulación Lunar Lander

activo simultáneamente. Sin embargo, dado que se toma una decisión en cada instante de tiempo, en la práctica parece que varios motores se activan al mismo tiempo.

En cuanto a la toma de decisiones, se recibe un vector de observaciones sobre el estado actual de la nave. Este vector incluye la posición actual de la nave en ambos ejes, su velocidad en cada uno de ellos, la velocidad angular, el ángulo de la nave en ese instante, y una variable binaria para cada sensor en los lados de la nave que indica si ha habido contacto con la superficie.

Para medir el desempeño, se utiliza una métrica que premia la suavidad del aterrizaje y el control de la nave, y penaliza el gasto excesivo de combustible. Así, maximizar la puntuación implica encontrar un equilibrio en el aterrizaje. Para considerar un aterrizaje exitoso se debe obtener al menos una puntuación de 200 puntos. No obstante, existen agentes en el estado del arte capaces de superar los 300 puntos.

- Aumenta/disminuye cuanto más cerca/más lejos esté el aterrizador de la plataforma de aterrizaje.
- Aumenta/disminuye cuanto más lento/más rápido se mueva el aterrizador.
- Disminuye a medida que el aterrizador está más inclinado (ángulo no horizontal).
- Aumenta en 10 puntos por cada pata que esté en contacto con el suelo.
- Disminuye en 0.03 puntos por cada fotograma en que se activa el motor lateral.
- Disminuye en 0.3 puntos por cada fotograma en que se activa el motor principal.

Toda esta información se detalla con mayor precisión en la documentación del entorno [11].

De cara a la interacción en el entorno del Lunar Lander, se ha seleccionado la API de OpenAI[12] como el servicio responsable de proporcionar acceso a diversos Modelos de

Lenguaje de la compañía, siendo el más reciente el GPT-4o al momento de redactar este trabajo. La justificación de esta elección sobre otros servicios ofrecidos por diferentes empresas se detallará exhaustivamente más adelante en la sección 3, enfocándose principalmente en la existencia de asistentes y su adecuación al proyecto.

Una API facilita la comunicación estandarizada entre diferentes aplicaciones o servicios mediante un conjunto de reglas y protocolos que definen cómo interactúan distintos componentes de software, especificando los métodos y datos accesibles de una aplicación o plataforma. Su capacidad de integración con otras herramientas la convierte en una solución poderosa y ampliamente utilizada.

En el ámbito del machine learning, estas APIs son fundamentales para acceder a grandes volúmenes de datos proporcionados por diversas plataformas, lo que permite entrenar modelos en diversos dominios. Sin embargo, no se trata solo de acceder a datos. En este proyecto se necesita utilizar servicios que demandan recursos computacionales y de almacenamiento elevados, limitando su accesibilidad, como es el caso de los modelos LLM. Implementar Modelos de Lenguaje de manera local requiere hardware potente y una inversión significativa fuera del alcance de muchos usuarios. Por ello, recurrir a servicios que permiten ejecutar estos modelos de forma remota bajo demanda, con un costo accesible y sin incurrir en gastos de infraestructura comparables, facilita el acceso a una amplia variedad de modelos avanzados para su aplicación práctica.

La funcionalidad principal de la API de OpenAI es ChatCompletions, diseñada para facilitar la interacción conversacional entre usuarios y modelos de lenguaje. Permite enviar una serie de mensajes de entrada al modelo y recibir respuestas en lenguaje natural.

Para utilizar las herramientas disponibles en la API, es indispensable contar con una clave vinculada a la cuenta del usuario. Esta clave se utilizará al crear el objeto correspondiente, que quedará asociado a dicha cuenta. De esta forma, se regularán tanto el tipo de herramientas disponibles según la suscripción del usuario como la cantidad de tokens consumibles por los modelos a los que tiene acceso.

El algoritmo Q-Learning, introducido por Watkins en 1989[22], es un pilar fundamental del aprendizaje por refuerzo (RL) debido a su simplicidad y eficacia. Q-Learning busca aprender una función de valor de acción $Q(s, a)$ que estima la recompensa esperada de tomar una acción a en un estado s y seguir la política óptima desde allí. Utilizando una fórmula de actualización iterativa, este método permite que los agentes mejoren sus políticas independientemente de las acciones seguidas durante el aprendizaje (off-policy). No obstante, el algoritmo clásico de Q-learning presenta desafíos en espacios de alta dimensionalidad y en términos de estabilidad.

Una evolución significativa son las Deep Q-Networks (DQN), presentadas por Mnih et al. en 2015 [8]. DQN combina Q-Learning con redes neuronales profundas para manejar espacios de estado, como los observados en los juegos de Atari, permitiendo que el agente aprenda directamente de las imágenes crudas del entorno. Para estabilizar las actualizaciones de los valores Q , DQN introduce técnicas clave como la memoria de repetición de experiencia y una red de objetivos separada. Además, Double Q-Learning, propuesto por Hasselt en 2015 [18], aborda el sesgo positivo en la estimación del valor máximo al dividir la tarea de selección y evaluación de acciones en dos redes diferentes, reduciendo así la sobreestimación de los valores

Q y mejorando la estabilidad y precisión del aprendizaje. Por otro lado, Dueling Q-Network, introducido por Wang et al. en 2016 [20], descompone la función de valor de acción $Q(s, a)$ en dos partes: una función de valor del estado $V(s)$ y una función de ventaja $A(s, a)$. Esta descomposición permite al agente distinguir entre la importancia de ser un estado valioso y la de elegir una acción específica en ese estado, mejorando así la eficiencia y la robustez del aprendizaje en entornos complejos.

Otra contribución importante en el campo del RL son los algoritmos basados en políticas, como el *Trust Region Policy Optimization* (TRPO) y el *Proximal Policy Optimization* (PPO). TRPO, introducido por Schulman et al. en 2015 [16], asegura actualizaciones de política grandes pero confiables, maximizando una función de objetivo mientras mantiene la desviación de la nueva política con respecto a la antigua dentro de una región confiable. PPO, también propuesto por Schulman et al. en 2017 [17], simplifica TRPO al usar una técnica de optimización por lotes que aplica restricciones de confianza en las actualizaciones de la política mediante penalizaciones en la función de objetivo, resultando en una implementación más simple y eficiente.

Estos desarrollos, han sido cruciales para aplicaciones avanzadas en áreas como juegos y robótica, permitiendo que los agentes aprendan y actúen de manera efectiva en entornos complejos y de alta dimensionalidad.

2.1.1 Large language models

Los Grandes Modelos de Lenguaje (LLM) han revolucionado el campo del procesamiento del lenguaje natural (NLP) gracias a su capacidad para comprender y generar texto coherente y contextualizado. Todo ello parte de la arquitectura fundamental que ha habilitado estos avances, el **Transformer** [19], que introdujo un mecanismo de atención eficiente y paralelizable para modelar relaciones a largo plazo en secuencias de entrada. Los primeros modelos como ELMo (Peters et al., 2018) y BERT (Devlin et al., 2019) introdujeron el uso de embeddings contextuales y técnicas de preentrenamiento bidireccional, lo que permitió grandes avances en tareas de NLP [3], [14]. Sin embargo, la verdadera explosión de los LLM se produjo con la introducción de GPT-3 por OpenAI en marzo de 2021 [9], que demostró una capacidad sin precedentes para realizar una amplia variedad de tareas de NLP sin necesidad de entrenamiento adicional específico para cada tarea. Este avance se consolidó con la salida de ChatGPT en 2022, que popularizó enormemente el uso de LLMs en aplicaciones prácticas y cotidianas.

Desde entonces, ha habido un continuo desarrollo y perfeccionamiento de LLMs tanto por parte de grandes compañías como de iniciativas open source. Recientemente, OpenAI ha presentado GPT-4o [10], que continúa ofreciendo las mejoras de capacidad de GPT-4 para seguir instrucciones y proporcionar respuestas más precisas, pero de forma más compacta y rápida. Destaca también Claude 3.5 Sonnet[2] de Anthropic, que ofrece competencia a OpenAI, o Llama 3 [6] de Meta y Mistral [7], que representan los últimos avances en la tecnología de LLMs en el ámbito open source.

2.2 Trabajo relacionado

Como se mencionó en la introducción, este proyecto se enmarca en una línea de investigación inspirada por el artículo "Language Models are Spacecraft Operators"[15], que propone un enfoque innovador para la toma de decisiones de modelos en entornos de misiones no cooperativas en el ámbito espacial. Dentro del ámbito de los agentes autónomos espaciales, este estudio presenta una metodología que se distingue por aprovechar el conocimiento previo de los LLMs, con el fin de reducir los costes de entrenamiento de los agentes y mejorar la explicabilidad mediante el uso de lenguaje natural.

El artículo se originó a partir de la participación el desafío de *Kerbal Space Program Differential Games* (KSPDG), donde logró el segundo puesto con una propuesta diseñada para desafiar a los agentes tradicionales de aprendizaje por refuerzo. El proceso consiste en convertir las observaciones en tiempo real en instrucciones de texto para el modelo LLM. Este procesa las instrucciones y genera una acción específica, implementada en un entorno de simulación para controlar la nave espacial (véase la Figura 2.2). Para este propósito se emplea GPT-3.5 como LLM, utilizando la API de OpenAI para la interacción. Además, se utilizan datos de ejemplo recopilados de jugadores humanos para llevar a cabo un ajuste del modelo (fine-tuning), con el objetivo de aprender y replicar su comportamiento en las tareas del desafío.



Figura 2.2: Esquema de funcionamiento de Language Models are Spacecraft operators sobre el entorno KSPDG. ([15])

La afinación del prompt es crucial para obtener resultados óptimos en los LLM. Según Wang et al. (2024), el artículo *Chain of Table: Evolving tables in the reasoning chain for table understanding* [21] representa uno de los avances más recientes en este campo al guiar a los LLM a través de operaciones tabulares dinámicamente planificadas, generando tablas intermedias que reflejan el proceso de razonamiento. Este enfoque detallado en cada fase facilita la formulación de consultas finales y ha demostrado un rendimiento superior en la comprensión de tablas en comparación con métodos de razonamiento genéricos, superando a metodologías previas como *Chain of Thought Prompting Elicits Reasoning in Large Language Models* (Wei et al., 2022) [23] y *Thread of Thought Unraveling Chaotic Contexts* (Zhou et al., 2023) [25].

Otros estudios se centran en mejorar el rendimiento mediante el prompt para tareas específicas. Según Li et al. (2023), en su artículo *Chain of Code: Reasoning with a Language Model-*

Augmented Code Emulator [4], se propone un enfoque innovador que combina la generación de código con la simulación mediante un *LMulator*, un emulador de código que integra las capacidades de un Modelo de Lenguaje con la ejecución de código. Este enfoque permite a los modelos de lenguaje abordar desafíos que implican razonamiento lógico, aritmético y semántico, mejorando la toma de decisiones en entornos dinámicos y adaptativos. La metodología de Chain of Code logra una integración efectiva entre la estructura formal del código y las capacidades de razonamiento de los modelos de lenguaje, prometiendo resolver una amplia gama de problemas de manera más eficiente y precisa en diversos campos.

Además, *Take a step back: Evoking reasoning via abstraction in large language model* (Zheng et al., 2023) [24] destaca la técnica de retroceso en la formulación de preguntas para evocar razonamientos abstractos en modelos de lenguaje a gran escala. A pesar de los avances, el razonamiento complejo de múltiples pasos sigue siendo un desafío para los LLM más avanzados, y el uso de retroceso en la formulación de preguntas podría potenciar su capacidad para abordar cuestiones que requieren un pensamiento abstracto y profundo.

Un artículo particularmente innovador es *EUREKA: Human-level Reward Design via Coding Large Language Models* [5], que presenta un método alineado con el enfoque de este problema para generar funciones de recompensa mejoradas en problemas de aprendizaje por refuerzo. Este método logra la generación de recompensas a nivel humano mediante la mutación contextual y la optimización iterativa de una población de soluciones generadas y evolucionadas a partir de modelos LLM. Este enfoque novedoso y escalable resalta la eficacia de combinar LLMs con algoritmos evolutivos para resolver problemas complejos de búsqueda. Esto representa una línea de investigación abierta con un enorme potencial y aplicaciones en diversos tipos de problemas.

La figura 2.3 ilustra el ciclo iterativo descrito en el artículo. En ella se muestra cómo el proceso comienza generando múltiples funciones de recompensa mediante un LLM alimentado con el código del entorno y la descripción de la tarea. A continuación, se selecciona una función de recompensa inicial que se modifica, generando nuevas soluciones basadas en la retroalimentación textual. Este proceso se repite iterativamente, proporcionando feedback al modelo sobre las soluciones creadas para mejorar progresivamente las funciones de recompensa.

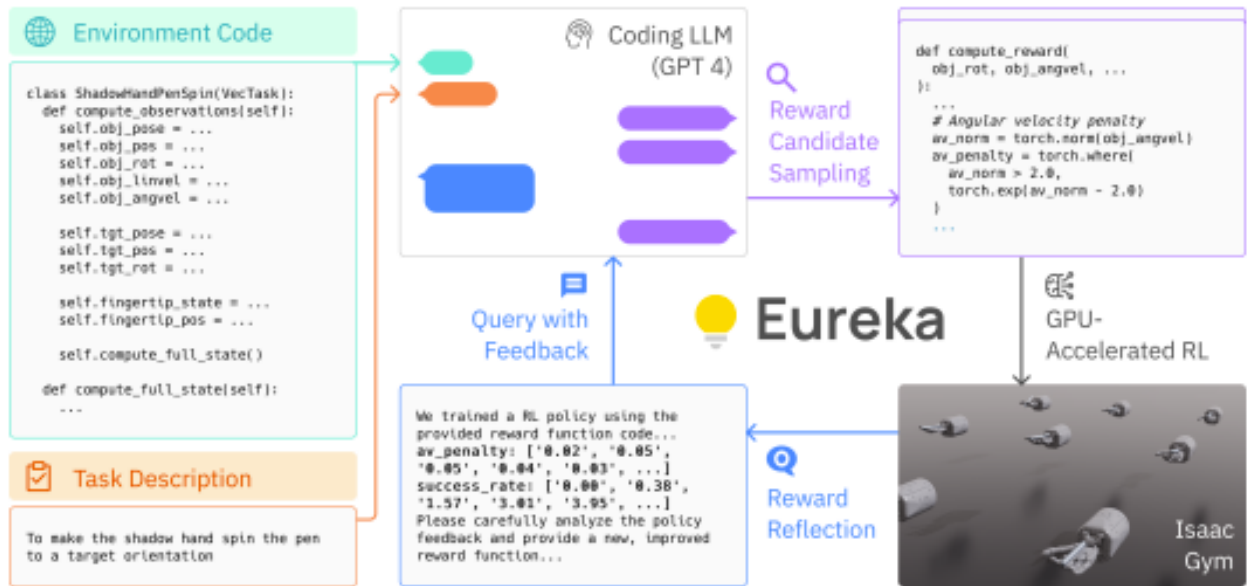


Figura 2.3: Esquema de funcionamiento de Eureka [5] para la optimización de funciones de recompensa en distintos entornos.

Capítulo 3

Materiales y métodos

El motor de este proyecto son los LLMs accesibles mediante las API habilitadas por diversas empresas desarrolladoras. Aunque existe una amplia oferta de modelos que cumplen estas características, como por ejemplo *Antrophic* [1] con su modelo *Claude*, en este proyecto hemos optado por desarrollar el framework utilizando la API de *OpenAI*. La razón principal para considerar esta herramienta más adecuada que las demás disponibles es la incorporación de asistentes [13]. Este módulo permite integrar el modelo deseado dentro del propio asistente, junto con un hilo que mantiene el intercambio de mensajes entre el usuario y el modelo. A diferencia de las implementaciones de ChatCompletions de OpenAI y Anthropic, este enfoque mantiene el estado de la conversación, facilitando un diálogo más coherente y contextualizado. Esta metodología de trabajo se alinea perfectamente con el enfoque planteado, donde de forma iterativa el modelo debe mejorar el código generado en base a los registros de ejecución de sus implementaciones pasadas.

3.1 Assistants API

En este proyecto, se ha desarrollado una clase encargada de encapsular todas las llamadas necesarias a la API para realizar las tareas requeridas. Estas llamadas consisten principalmente en las acciones tradicionales de un entorno API REST (POST, GET y DELETE) sobre los objetos propios del asistente. Estos objetos son los siguientes:

- **assistant**: El propio objeto del asistente. Recibe como parámetros el modelo, nombre, instrucciones, descripción y las herramientas disponibles. Al igual que en ChatCompletion, tiene disponibles funcionalidades como búsqueda de archivos, interpretación de código y llamadas a funciones.
- **thread**: Hilo asociado con el asistente donde se concatenan los mensajes de la conversación entre usuario y modelo.
- **run**: Es el objeto con el que se solicita la inferencia al modelo, también permite responder al modelo cuando el asistente incluye alguna de las herramientas disponibles.

3.2 Ejecución de entorno Lunar Lander

La base del proceso iterativo radica en evaluar el rendimiento del código generado en el entorno Lunar Lander. Mediante la ejecución de episodios en este entorno se generan registros iniciales y satisfactorios que sirven de ejemplos para el modelo. Para la implementación del código encargado de la ejecución de estos episodios, se parte de los conceptos explicados en la sección 2.1.

Todo el proceso se gestiona a través de una función que recibe como parámetros el número de episodios, las semillas del entorno y el tipo de agente responsable del aterrizaje. La función recibe una lista de semillas asociadas a cada episodio que se desea ejecutar. Su uso es crucial para asegurar la reproducibilidad y la comparación entre diferentes implementaciones, ya que la forma adecuada de comparar soluciones es ejecutarlas bajo las mismas condiciones iniciales. Además, facilita la generalización del modelo al permitir la ejecución bajo diversas condiciones iniciales, evitando así que el modelo memorice un único escenario de aterrizaje.

Asimismo, la función permite seleccionar cómo mostrar los episodios de aterrizaje: mostrando el aterrizaje en pantalla, sin mostrarlo o generando y guardando un vídeo en una ubicación específica.

Esta función también se encarga de almacenar los registros de cada instante de tiempo hasta completar el episodio completo. En esta etapa se elige un enfoque de compresión de registros, ajustando la cantidad de instantes de tiempo registrados. La clave reside en encontrar un equilibrio que optimice la cantidad de tokens empleada para cada registro, asegurando que el modelo no pierda información esencial sin incurrir en un consumo excesivo. Para lograrlo, es crucial identificar momentos críticos durante el aterrizaje, como la fase inicial donde la estabilización de la nave requiere mayor precisión informativa que durante un descenso controlado.

3.3 Prompt engineering

Una parte esencial que influye directamente en los resultados es el prompt que recibe el modelo. Afinarlo y aplicar las técnicas descubiertas por los investigadores en este campo constituye una parte fundamental de este trabajo. El objetivo principal es mejorar la comprensión del problema planteado y de las variables involucradas. Dada la limitada cantidad de datos que el modelo recibe para generar una salida de calidad, es crucial que esta información se presente con la máxima precisión y claridad posible.

3.3.1 Registros

La parte esencial de la comunicación con el agente reside en los registros de los episodios de aterrizaje, los cuales sirven como retroalimentación para que el modelo analice el aterrizaje efectuado con el código generado, así como en los episodios de ejemplo que pueda recibir en el prompt inicial. Un aspecto clave en la construcción de este prompt es seguir una estructura que facilite la comprensión de una tarea intrínsecamente difícil, como es recibir las observaciones de cada instante de tiempo e interpretar lo que está ocurriendo. Para ello, se almacena cada

registro de cada instante de tiempo de un episodio en un diccionario con formato JSON. Previamente, se ha alimentado al modelo con el esquema del diccionario de registros (ver código 3.1) , lo que ayuda a interpretar cada elemento y permite omitir esta información en el propio registro, haciendo más compacto un elemento que inevitablemente consumirá un número elevado de tokens. Con esta misma filosofía, se busca identificar los momentos clave del aterrizaje para incluir todos sus instantes de tiempo, y agregar datos cada n instantes en otros momentos.

Listing 3.1: Esquema JSON para los datos de registro del entorno de aterrizaje de la nave espacial. Basado en el formato de la respuesta del método step.

```
{
  "description": "Log data for each step of the spacecraft
    landing environment.",
  "landing attempt": {
    "type": "integer",
    "description": "The episode number."
  },
  "logs": [
    {
      "time": {
        "type": "integer",
        "description": "The instant within the landing
          attempt where the current log is taken."
      },
      "action": {
        "type": "integer",
        "description": "The possible actions taken.",
        "options": {
          "0": "Switch off engines",
          "1": "Push left engine",
          "2": "Push both engines (upwards)",
          "3": "Push right engine"
        }
      },
      "current status": {
        "type": "array",
        "description": "The state of the environment
          after the action is taken.",
        "positions": {
          "0": {
            "name": "X position",
            "description": "The horizontal position
              of the spacecraft relative to the
              landing zone."
          },

```

```
    "1": {
        "name": "Y position",
        "description": "The vertical position
                        of the spacecraft relative to the
                        landing zone."
    },
    "2": {
        "name": "X velocity",
        "description": "The horizontal velocity
                        of the spacecraft."
    },
    "3": {
        "name": "Y velocity",
        "description": "The vertical velocity
                        of the spacecraft."
    },
    "4": {
        "name": "Angle",
        "description": "The angle of the
                        spacecraft relative to the vertical (
                        left negative, right positive)."
    },
    "5": {
        "name": "Angular velocity",
        "description": "The rate of change of
                        the angle of the spacecraft."
    },
    "6": {
        "name": "Left contact sensor with
                landing zone",
        "description": "Indicates whether the
                        left side of the spacecraft is in
                        contact with the landing zone."
    },
    "7": {
        "name": "Right contact sensor with
                landing zone",
        "description": "Indicates whether the
                        right side of the spacecraft is in
                        contact with the landing zone."
    }
},
"min_values": [-1.5, -1.5, -5.0, -5.0, -3.14, -
               5.0, 0, 0],
"max_values": [1.5, 1.5, 5.0, 5.0, 3.14, 5.0, 1
```



```

        , 1]
    },
    "score": {
        "type": "number",
        "description": "The score received for the
            action."
    }
},
],
"total score": {
    "type": "number",
    "description": "The total score received for the
        landing attempt."
}
}

```

3.3.2 Prompts de sistema

El método encargado de crear el asistente recibe como parámetros el nombre, la descripción y las indicaciones que recibirá internamente el asistente. La descripción proporciona al modelo una guía general sobre su funcionalidad, especificando su tarea y objetivo. Por otro lado, las instrucciones aportan información más precisa al agente sobre cómo llevar a cabo su tarea. En esta sección se describen detalladamente las subtarefas que debe seguir para completarlas, como analizar los logs, razonar sobre la evolución de los estados a partir de las acciones tomadas, codificar la toma de decisiones en la función 'act' y almacenarla en un archivo de tipo Python. Además, se incluyen consejos y puntos clave para obtener un mejor rendimiento, así como un esquema del sistema de puntuación que utiliza el entorno para valorar el aterrizaje.

3.3.3 Prompts de usuario

Además de los prompts de sistema que recibe el modelo, existen los prompts de usuario que se concatenan junto con las respuestas del modelo en el objeto hilo. Hay dos tipos de mensajes de usuario destacados. El mensaje inicial incluye ejemplos de aterrizajes exitosos, así como la ejecución y el código Naive desde el cual el modelo comienza a mejorar incrementalmente su rendimiento. Después de cada ejecución, se agrega otro mensaje a nivel de usuario que contiene el registro de las ejecuciones con el último código generado. En este mensaje se anima al modelo a identificar errores en el aterrizaje y relacionarlos con la parte del código donde ocurren estos errores en su toma de decisiones, además de comparar con ejecuciones anteriores y observar cómo está evolucionando. También se alienta al modelo a no temer realizar cambios significativos que mejoren el rendimiento, ya que tiende a resistirse a modificar la estructura y se limita típicamente a ajustar valores de hiperparámetros.

3.4 Function calling

La herramienta que recibe el asistente en este proyecto es la llamada a funciones. En particular, la llamada que recibe como parámetro (el esquema de la función, 3.2) tiene como objetivo guardar el código generado en un archivo Python, recibiendo como parámetros tanto el código como el nombre del archivo. El objetivo es guiar al modelo dentro del proceso iterativo y obtener código ejecutable de forma directa y sin complicaciones. Una de las ventajas de esta metodología es la inclusión de un método de commit dentro del bucle iterativo, lo que permite observar directamente en el repositorio del proyecto los cambios efectuados en el archivo (que será siempre el mismo) tras cada iteración de mejora del agente.

Listing 3.2: Esquema de la función "store_code_in_file" proporcionada al modelo como herramienta.

```
{
  "name": "store_code_in_file",
  "description": "Store code in a file",
  "parameters": {
    "type": "object",
    "properties": {
      "reasoning": {
        "type": "string",
        "description": "Pre-coding deep reasoning on
          log analysis and the influence of actions on
          observations and steps to follow to achieve a
          successful landing. It is an expansion but
          not the code itself."
      },
      "code": {
        "type": "string",
        "description": "The Python code to store. Use
          the reasoning parameter to generate this code
          ."
      },
      "filename": {
        "type": "string",
        "description": "The filename to store the code
          in."
      }
    },
    "required": ["code", "filename"]
  }
}
```

3.5 JSON mode

Otra alternativa al uso directo de herramientas (function calling, code interpreter y file search) es solicitar la salida del modelo en el formato deseado, que frecuentemente, también en este proyecto, se realiza en JSON. Una práctica recomendada, cuando el problema lo requiere, consiste en proporcionar al modelo un esquema de salida definido con el formato deseado. En este caso, se busca mantener una coherencia con la filosofía del function calling, donde los argumentos de la función se convierten en atributos en formato JSON. El esquema se muestra en el código 3.3

Listing 3.3: Esquema de respuesta solicitada al modelo con formato JSON.

```
{
  "reasoning": {
    "type": "string",
    "description": "Pre-coding deep reasoning on log
                    analysis and the influence of actions on observations
                    and steps to follow to achieve a successful landing.
                    It is an expansion but not the code itself."
  },
  "code": {
    "type": "string",
    "description": "The Python code to store. Use the
                    reasoning parameter to generate this code."
  },
  "filename": {
    "type": "string",
    "description": "The filename to store the code in (
                    Action.py)."
  }
}
```

3.6 Proceso iterativo

El núcleo del framework comienza generando el asistente y los objetos necesarios para establecer la comunicación con el modelo. Como se ha mencionado anteriormente, el objetivo es que el modelo mejore iterativamente su codificación de toma de decisiones a partir de los resultados de ejecuciones previas. Además, se tienen en cuenta los errores de programación que puedan causar una excepción; en estos casos, se retroalimenta al modelo con el error y no se avanza a la siguiente iteración hasta obtener un código que compile correctamente. En esta sección, es crucial prestar atención a los tiempos de espera, tanto en las llamadas a la API (donde es necesario esperar a que el estado de los objetos sea el adecuado para continuar) como en el propio sistema, asegurando que el almacenamiento y la carga del nuevo código generado se realicen correctamente. Este proceso se ilustra en la figura 1.1 del capítulo 1.

3.7 Agentes ejemplo

Una de las principales ventajas del entorno previamente mencionada es su amplio uso en entornos académicos de aprendizaje por refuerzo. Para proporcionar al agente ejemplos de aterrizajes exitosos, hemos empleado un modelo con una arquitectura de *Dueling Q Network* utilizando los pesos obtenidos de su entrenamiento previo. Este modelo ofrece resultados satisfactorios de manera confiable, superando consistentemente una puntuación de 200, lo que permite generar registros de episodios de aterrizaje exitosos con distintos valores de semilla del entorno.

3.8 Zero-shot prompting

Una prueba más exigente para el modelo consiste en comenzar sin ejemplos de aterrizajes exitosos, ni código inicial ni registros previos. De esta manera, se otorga al modelo una mayor libertad en la generación de código al no contar con un esquema base predefinido (si cuenta con el esquema de formato de logs). Sin embargo, el desafío inicial es considerablemente más complejo al carecer de cualquier referencia inicial, ya sea exitosa o de partida. Este enfoque busca poner a prueba la capacidad del modelo para mejorar y analizar registros en condiciones desafiantes para su conocimiento base.

Partir desde un punto tan complejo evalúa su capacidad de convergencia en esta situación, utilizando las herramientas proporcionadas para adaptarse y evolucionar.

3.9 Few-shot prmpting

Esta metodología constituye el enfoque principal de este proyecto. Se presentan al modelo unos pocos ejemplos de registros de aterrizajes exitosos, junto con un código inicial básico (inicialmente elige acciones al azar, ver sección .1 del Anexo) diseñado para que el modelo ajuste su código al formato de función deseado. Además, se incluye el registro de ejecución de este código inicial. Esta estrategia permite al modelo comparar episodios de aterrizaje exitosos y no exitosos, analizando los pasos necesarios para tomar decisiones que conduzcan al agente de aterrizaje deseado.

Esta metodología es de gran ayuda, ya que, aunque las condiciones iniciales pueden variar, proporciona al modelo una comprensión detallada de los objetivos del código generado. Sin embargo, el punto de partida en la generación de código es esencialmente similar al enfoque "zero-shot", dado que no se proporcionan ejemplos para guiar modificaciones iterativas en el código, lo cual podría influir en la estructura y desarrollo del modelo. En el código 3.4 se observa el prompt inicial junto con las instrucciones en la creación de un agente.

Listing 3.4: Prompt para el modelo con ejemplos exitosos y el código inicial

```
DESCRIPTION = "You are an expert spacecraft landing agent. Your
specialty is to take the appropriate action at each instant
of time based on the environment and state of the spacecraft.
To achieve this, you put your knowledge base and the
```

```

knowledge you acquire by analyzing each execution into a
python function in charge of directing the landing."
INSTRUCTIONS = f"""To complete the task you must follow the
following steps and indications:
1. Analyze and reason about the records received in the last
landing attempts to learn how the spacecraft behaves in the
environment. Keep in mind that conditions may vary.
2. To measure how good a landing is, a scoring system that
appears in the logs is used, your role is to maximize it. To
do so, keep the following in mind:
    - Increases/decreases the closer/further the spacecraft is
      from the landing area (both axes).
    - Increased/decreased the slower/faster the spacecraft is
      moving.
    - Decreased the more the spacecraft is tilted.
    - Decreased by 0.03 points each frame a side engine is
      firing and 0.3 points for the center engine.
    - Receive 100 points for a successful landing and lose them
      for crashing.
    - A succesful landing must exceed 200 points.
3. This is the log schema:\n {log_description}
4. Add your decision code to the 'act' method and locate errors
   in it if the landing is unsuccessful.
5. Save the code of the act function in the file 'Action.py'
   using store_code_in_file function.
"""
NAME = "Spacecraft Landing Master"

initial_msg = f"This is the record of an example of a
successful landing in this environment, but under other
conditions: {success_logs}. This is the code of the initial
function: {agent_initial_code} and these are the execution
logs of one landing attempt: {initial_logs}. What does each
value and each change in the observations mean? What effects
does each action have on the current state observations? What
are the steps to follow to achieve a successful landing?.
Take a deep breath and reason step-by-step. After reasoning
analyze the results, learn and make better code."

```

3.10 Algoritmo evolutivo

El artículo *Eureka* [5] presenta una metodología evolutiva para mejorar funciones de recompensa. Aunque las naturalezas de ambos trabajos difieren, este artículo ha servido de inspiración para implementar en este estudio una metodología evolutiva en el algoritmo iterativo de

mejora del código de toma de decisiones. El enfoque consiste en comenzar con una población inicial de modelos diseñados para resolver la misma tarea bajo condiciones idénticas, lo que promueve una diversidad inicial en el espacio de búsqueda de la solución óptima.

En cada iteración, todos los individuos de la población reciben el código y el registro del individuo que obtuvo la mejor puntuación en el episodio de aterrizaje, junto con los registros del propio aterrizaje según su código. De esta manera, cada modelo tiene la responsabilidad de ajustar su código a partir de su experiencia individual y el conocimiento del mejor rendimiento global de la población. Este enfoque busca incrementar la variabilidad en las soluciones iniciales generadas y fomentar una mejor convergencia con la ayuda colectiva de otros individuos.

Capítulo 4

Resultados

Un aterrizaje se considera exitoso en el entorno de Lunar Lander cuando la puntuación supera los 200 puntos. Una solución es considerada buena si puede alcanzar este objetivo de manera consistente en múltiples ejecuciones con diferentes condiciones iniciales. Sin embargo, lograr este objetivo a través de la codificación de la toma de decisiones es ambicioso y resalta el potencial de los modelos LLM en el desarrollo de tareas que permitan ahorrar tiempo de entrenamiento y ofrecer una respuesta rápida en entornos donde este aspecto sea crítico.

En esta sección, se presentarán los resultados obtenidos en las pruebas realizadas con las distintas variantes propuestas. Posteriormente, en la sección 5, se discutirá la competitividad de estas variantes, sus puntos fuertes y débiles, y su potencial.

Uno de los problemas del entorno es que la puntuación final, por sí sola, no es un indicador preciso de cuán cerca está el modelo de alcanzar un código que ofrezca buenos resultados. El problema es extremadamente sensible a la prioridad de los objetivos considerados durante el aterrizaje, así como a los valores establecidos como umbrales en las observaciones recibidas para tomar una u otra acción. Esto significa que, en el análisis, no basta con valorar únicamente las métricas obtenidas, sino también la calidad del código y el razonamiento llevado a cabo por el modelo.

4.1 Comparativa entre modelos

Durante las pruebas, se han evaluado dos modelos de OpenAI: GPT-3.5-turbo y GPT-4o. En esta sección se analizarán los resultados obtenidos por cada uno de ellos y se extraerán las conclusiones pertinentes para el proyecto, considerando la dependencia del modelo utilizado.

4.1.1 Ejecuciones con GPT-3.5-Turbo

Una particularidad del bucle iterativo es que, en caso de error, se retroalimenta al modelo con la traza para lograr que compile, sin contar esta corrección como una de las N iteraciones destinadas a mejorar el rendimiento del código.

Las pruebas han mostrado consistentemente que GPT-3.5-Turbo no es adecuado para el

problema planteado. Su capacidad de razonamiento y análisis es considerablemente inferior a la de GPT-4o. Las comparaciones de métricas entre ambos modelos no son viables, ya que GPT-3.5 Turbo tiene dificultades para generar código compilable con las indicaciones dadas. Además, su código suele presentar inconsistencias en los tipos de variables o es demasiado simple, indicando una falta de análisis y comprensión del problema, así como de una estrategia de mejora continua.

En el código 4.1 se observa que, a pesar de recibir indicaciones claras sobre el formato de las observaciones y una explicación detallada de cada uno de sus elementos, el modelo intenta acceder a una posición inexistente en la lista para buscar el atributo score. Este atributo no está incluido en el espacio de observaciones proporcionado, lo que evidencia una falta de comprensión del modelo respecto a la estructura de los datos de entrada. Se observa, por tanto, cierta dificultad del modelo para entender que la función debe devolver una acción y no un score, el cual debería ser utilizado como retroalimentación para analizar las ejecuciones. Además, el razonamiento y la complejidad del código en la toma de decisiones son demasiado simples, lo que indica una falta de profundidad en el análisis del problema.

Listing 4.1: Ejemplo de código generado por el agente basado en el modelo GPT-3.5-turbo

```
# Ejemplo 1.
def act(obs):
    # Analyzing observations
    speed_x = obs[0]
    speed_y = obs[1]
    pos_x = obs[2]
    pos_y = obs[3]
    angle = obs[4]
    left_engine = obs[5]
    main_engine = obs[6]
    right_engine = obs[7]
    score = obs[8]
    # Decision-making
    if pos_x < 0:
        return 3 # Fire left engine to correct position
    elif pos_x > 0:
        return 1 # Fire right engine to correct position
    else:
        return 2 # Fire main engine to stabilize

# Ejemplo 2.
def act(observation):
    if observation[2] < 0.1:
        return 1 # Fire the main engine to slow down
    elif observation[2] > 0.1 and observation[3] > -0.3:
        return 2 # Keep using the main engine to maintain
                  altitude
    else:
```



```
return 3 # Use side engines to adjust the position and  
prepare for landing
```

Quizá la mayor decepción es el análisis de la situación por parte de GPT-3.5-turbo. El modelo apenas realiza un razonamiento retrospectivo, como se muestra en la figura 4.1, y no aporta nada adicional a lo que ya recibió como input para la resolución del problema.

The reasoning behind the code is to implement logic that optimizes the spacecraft landing by considering factors such as the position, velocity, angle, angular velocity, and side engines firing. The goal is to maximize the score by adjusting actions based on the current status of the spacecraft.

Figura 4.1: Razonamiento previo del modelo GPT 3.5 turbo en la primera iteración.

4.1.2 Ejecuciones con GPT-4o

La mejora con GPT-4o es verdaderamente significativa. El nivel de comprensión del problema y análisis de la situación pese a la poca cantidad de registros y a la alimentación verbal es verdaderamente asombroso. En cuanto a la codificación, se ve una estructura de cara a la resolución del problema, y la capacidad de generar código compilable es prácticamente un seguro, en los pocos casos que no es así, es capaz de corregirlo de forma inmediata y centrarse en la mejora iterativa del código.

En el código 4.2 se observa un notable incremento en la calidad del razonamiento previo del modelo. Este detalla las bases del problema e identifica los puntos clave a seguir durante el desarrollo del código, demostrando una sorprendente capacidad de comprensión a partir del formato recibido y con una cantidad limitada de ejemplos proporcionados. Además, en la figura 4.3, se aprecia una retroalimentación útil tras analizar cada episodio, localizando los aspectos que requieren mejoras o ajustes.

El código presentado en la figura 4.2 muestra una mejora significativa en la calidad con el cambio de modelo. Se observa una estructura claramente definida basada en prioridades, acompañada de comentarios que reflejan el razonamiento del modelo. Esto evidencia una relación clara entre el razonamiento retrospectivo (step-back reasoning) y la localización de los elementos detallados en el código.

```
### Analysis of Successful Landing Log and Initial Logic
1. Initial Conditions:
- The spacecraft is at a high altitude.
- It has a slight horizontal and vertical velocity and a minor tilt.
2. Actions:
- The sequence primarily uses actions 1, 2, and 3 (engine pushes in
different directions).
- In many instances, action 0 (engines off) is chosen strategically.
3. Scoring:
- The score decreases with higher velocities and tilt angles; there's also a
cost for firing engines.
- Maintaining minimal horizontal and vertical velocities is crucial.
- A balanced angle near to vertical (0 degrees relative angle) is necessary.
4. Key Observations:
- X and Y velocities should be minimized before landing.
- Angular velocity and angle need to be properly managed using side engines.
- Keep the use of engines balanced to minimize score penalties.
### Strategy for Successful Landing
1. Start Descent: Use balanced engine thrust to control descent, ensuring
that vertical velocity doesn't become too high.
2. Minimize Horizontal Movement: Use side engines to correct any horizontal
drift. Aim to keep horizontal velocity close to zero.
3. Correct Angle: Use side engines to ensure the spacecraft is as close to
vertical as possible before landing.
4. Final Approach: Ensure that vertical velocity is minimal, horizontal
velocity is near zero, and the spacecraft is vertical.
5. Touchdown: Ensure contact sensors indicate a stable landing with no
bouncing or sliding.
```

Figura 4.2: Razonamiento previo del modelo GPT-4o.

```

### Analysis of the Logs
### Performance Differences:
1. Initial Descent:
- The initial strategy of not using engines (action = 0) is ineffective. The spacecraft's velocity and position drift significantly before any corrective action is taken.
2. Horizontal Velocity Management:
- The spacecraft has a persistent horizontal drift, leading to continual corrections without effectively slowing down horizontal speed. The code intermittently uses both engines to slow descent but does not consistently manage horizontal drift early enough.
- Instead of reducing horizontal speed, the spacecraft drifts increasingly, causing frequent use of side engines, leading to increased costs and tilting problems.
3. Vertical Velocity Management:
- The vertical speed is not adequately reduced, leading to hard impacts and subsequent crashes. This is exacerbated by the spacecraft drifting horizontally the entire time.
4. Tilt and Angle Control:
- The angle control is insufficient, allowing significant tilt and high angular velocities to develop and persist. This causes further use of side engines, inducing penalties and instability.
### Logical Adjustments
- Early Intervention: Start controlling descent early, with gradual vertical speed reduction and drift correction before high speeds set in.
- Prioritize Major Corrections: Rotate to vertical quickly; manage angular velocity before reducing horizontal drift. Gradually correct horizontal speed while handling vertical descent continuously.
- Angle Prioritization: First, always correct angles to near zero before reducing speed so control can be more predictable.
- Controlled Descent and Drift: Alternate actions between reducing both vertical and horizontal velocities and balancing tilting.
### Implementation Strategy
1. Start descent with immediate engine use to slow vertical speed and correct drift.
2. Alternate between angle corrections and horizontal velocity corrections if vertical velocity is controlled.
3. Final descent control for reduced horizontal and vertical velocity, ensuring minimal tilt.
4. Prioritize consistent vertical speed reduction and drift balancing throughout descent.

```

Figura 4.3: Razonamiento en la mejora iterativa del modelo GPT-4o.

Listing 4.2: Ejemplo de código generado por GPT-4o

```
def act(state):
    x_pos, y_pos, x_vel, y_vel, angle, ang_vel, left_contact,
        right_contact = state

    # Stabilization
    if abs(angle) > 0.1:
        if ang_vel > 0:
            return 1 # Push left engine
        else:
            return 3 # Push right engine

    # Horizontal Positioning
    if x_pos < -0.1:
        return 3 # Push right engine
    elif x_pos > 0.1:
        return 1 # Push left engine

    # Descent Control
    if y_vel < -0.1:
        return 2 # Push both engines

    # Landing Phase
    if y_pos < 0.1 and abs(y_vel) < 0.1 and abs(x_vel) < 0.1:
        return 0 # Switch off engines

    # Default action
    return 2 # Push both engines
```

4.2 JSON mode

La opción de recibir salidas en formato JSON presenta el inconveniente inicial de una reducción en el rendimiento de la respuesta generada por el modelo, pero ofrece la ventaja de una mejora en la velocidad de respuesta. Esta variación en los tiempos no tiene relación directa con el rendimiento del modelo, que permanece constante, sino a la eficiencia en la comunicación con la API.

En este estudio, se ha observado que, al menos para el problema planteado, la disminución en el rendimiento tanto en el razonamiento como en la generación de código entre ambos métodos es pequeña y suele mostrarse más en el ratio de compilaciones exitosas. Sin embargo, la reducción en el tiempo de ejecución es significativa, como se muestra en la tabla 4.1.

Para este problema, el objetivo principal es maximizar el rendimiento, sin centrarse en mejorar el tiempo de respuesta, ya que el propósito es generar el código encargado de la planificación

Metodología	Tiempo de ejecución (min)
Function calling	7 - 8
JSON mode	2 - 3

Tabla 4.1: Tiempos de ejecución aproximados de ambas metodologías para 5 iteraciones (sin errores de compilación).

del aterrizaje, no del aterrizaje en sí. No obstante, esta mejora en los tiempos de respuesta será crucial en la implementación de un enfoque evolutivo para la resolución del problema.

4.3 Evolutivo

El objetivo de esta metodología es ampliar el número de soluciones iniciales para reducir la dependencia de un inicio prometedor, dada la naturaleza del problema, utilizando un número mínimo de iteraciones.

En la tabla 4.2 se presenta un ejemplo de ejecución con 5 iteraciones y una población de 4 modelos. Se observa una convergencia inicial muy favorable gracias a la cooperación entre los individuos. Sin embargo, la mejora continua del código eventualmente se estanca de manera generalizada. Aunque el rendimiento global de la población sigue mejorando, la mejor solución no muestra un avance consistente. En muchos casos, la diferencia entre la solución propia y la mejor solución es significativa, lo que lleva al modelo a enfocarse exclusivamente en la mejor solución disponible. Los valores negativos en la puntuación indican que las acciones del agente tienden a llevarlo a estados que incurren en penalizaciones en lugar de bonificaciones, lo que lo aleja del objetivo de alcanzar los 200 puntos.

Agente	Agente 1	Agente 2	Agente 3	Agente 4	Score Medio
Iteración 1	-705,548	-344,388	-451,782	-28,834	-382,638
Iteración 2	-28,834	-102,242	-28,834	-131,890	-72,950
Iteración 3	-28,834	-28,834	-28,834	-131,890	-54,598
Iteración 4	-28,834	-28,834	-28,834	-131,890	-54,598
Iteración 5	-28,834	-40,115	-28,834	-28,834	-31,654

Tabla 4.2: Scores finales obtenidos por cada agente en 5 iteraciones y su score medio.

Sin embargo, en la tabla 4.3 de la mejor solución del enfoque evolutivo se muestra como la mayoría de agentes tratan de mejorar la solución en lugar de copiar la mejor hasta el momento. Esto ayuda a evitar el estancamiento y aumenta las posibilidades de explorar áreas prometedoras del espacio de búsqueda, como se evidencia claramente en la tercera iteración

Agente	Agente 1	Agente 2	Agente 3	Agente 4	Score Medio
Iteración 1	-612,95	-612,95	-141,601	-556,65	-496,037
Iteración 2	-120,303	-141,601	-118,51	-131,723	-128,034
Iteración 3	-118,51	-110,018	192,155	-126,327	-40,675
Iteración 4	192,155	206,036	201,954	192,155	198,075
Iteración 5	206,036	-147,042	198,005	203,367	115,008

Tabla 4.3: Scores finales obtenidos por cada agente en 5 iteraciones y su score medio en la mejor solución evolutiva obtenida.

4.4 Comparativa

En esta sección se presentará en la tabla 4.4 el promedio de los valores obtenidos utilizando diversas metodologías a lo largo de un ciclo de 5 iteraciones. Se destacarán especialmente los mejores resultados y la capacidad de mejora de cada metodología. Es importante recordar que el score por sí solo no es suficiente para evaluar la calidad de un agente y su razonamiento, ni para determinar qué tan cerca está de una solución óptima.

Metodología	Valor Medio	Peor solución	Mejor Solución	Mejora total
Function calling	-265,69	-1491,79	9,09	732,493
Zero shot	-343,917	-1157,908	-39,725	457,497
JSON mode	-519,928	-1147,908	-134,717	382,146
Evolutivo	-208,915	-1157,908	10,455	927,347

Tabla 4.4: Comparación de valor medio, mejor solución y mejora para las metodologías utilizadas.

4.5 Mejor resultado

Los mejores resultados obtenidos logran superar el umbral de 200 puntos para considerarse un aterrizaje exitoso. En la figura 4.4 se ilustra la evolución a lo largo de las iteraciones para ambos enfoques de agentes. Mientras que el enfoque individual muestra una sensibilidad notable a un inicio prometedor, el enfoque evolutivo exhibe una mayor probabilidad de obtener un buen resultado inicial debido a los múltiples puntos de partida simultáneos, o al menos minimiza la posibilidad de comenzar con una solución extremadamente deficiente. Se observa también que, mientras la metodología individual puede oscilar en la puntuación en su intento de mejorar el código, el enfoque evolutivo mantiene la mejor solución obtenida por algún individuo, mientras que el resto de la población trabaja para mejorar sin afectar negativamente la solución global.

Para evaluar su capacidad de generalización, se presentará el resultado promedio y el porcentaje de aterrizajes exitosos en episodios con diversas condiciones iniciales (consultar tabla 4.5). Aunque la planificación basada en decisiones mediante código tiende a ser metodológicamente generalizable, este estudio no ha revelado modelos que intenten memorizar las acciones en función de condiciones iniciales específicas en cada instante de tiempo.

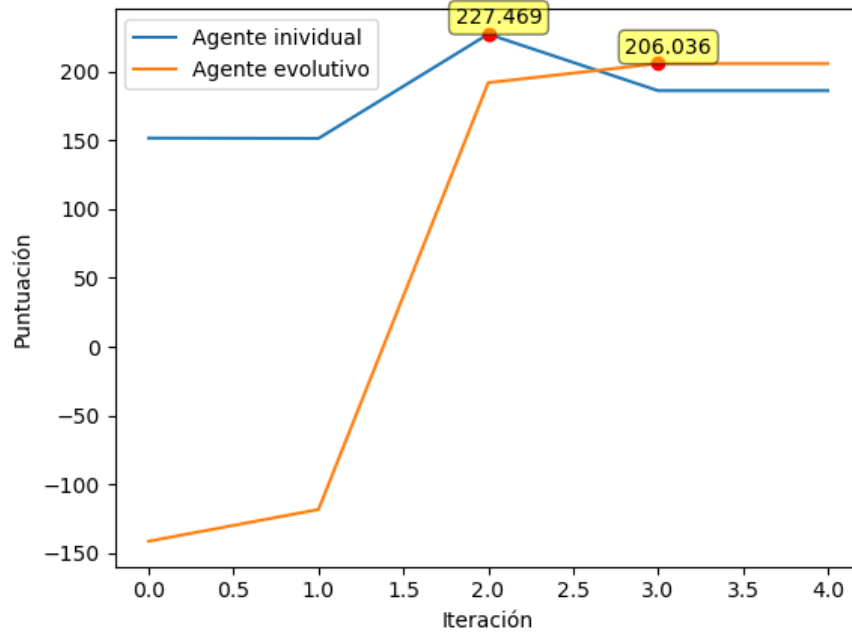


Figura 4.4: Evolución de la puntuación de los asistentes con mejor puntuación.

Metodología	Media	Umbral 100	Umbral 150	Umbral 200
Individual	140,931	75 %	73 %	51 %
Evolutivo	76,851	46 %	46 %	38 %

Tabla 4.5: Resultados promedio y porcentaje de agentes que superan los umbrales de puntuación tras 100 ejecuciones.

En la tabla 4.5 se observa que aproximadamente en la mitad de los episodios se supera el umbral considerado para el éxito con el enfoque individual. Sin embargo, al analizar los vídeos, se puede concluir que en muchos episodios donde no se alcanza este umbral se completa la tarea de manera satisfactoria, aunque una velocidad baja de descenso afecte negativamente a su puntuación. Además, se nota que la media general es baja debido a los resultados accidentales que tienen un impacto significativo en la puntuación global.

En el código con la mejor puntuación en el entrenamiento (ver 4.3), se observa un claro orden de prioridades y una serie de parámetros que deben ajustarse para intentar mejorar la puntuación. No obstante, esta tarea es muy delicada, ya que en la última ejecución del código se ajustaron más umbrales, lo cual resultó en un desempeño ligeramente inferior al logrado en el ajuste de la tercera iteración.

En la imagen 4.5 se presentan los cambios realizados por el agente en el código con la mejor valoración. Se puede observar cómo el agente realiza un refinamiento de los parámetros y plasma su razonamiento previo mediante comentarios en el código.

Listing 4.3: Ejemplo del código inicial de la mejor solución encontrada.

```
def act(observation):
    '''
```

```
Description ...
'''
x, y, x_vel, y_vel, angle, ang_vel, left_contact,
    right_contact = observation

# Define thresholds
x_vel_threshold = 0.1
y_vel_threshold = 0.1
angle_threshold = 0.1
ang_vel_threshold = 0.1

# Prioritize reducing y velocity if it's too high (falling too
    fast)
if y_vel < -y_vel_threshold:
    return 2 # Main engine on

# Stabilize angular velocity if needed
elif abs(ang_vel) > ang_vel_threshold:
    if ang_vel > 0:
        return 3 # Turn right engine on
    else:
        return 1 # Turn left engine on

# Stabilize angle if needed
elif abs(angle) > angle_threshold:
    if angle > 0:
        return 3 # Turn right engine on
    else:
        return 1 # Turn left engine on

# Reduce x velocity if it's too high
elif abs(x_vel) > x_vel_threshold:
    if x_vel > 0:
        return 1 # Turn left engine on
    else:
        return 3 # Turn right engine on

# If all is stable, keep main engine off for gentle descent
return 0
```


...	...	@@ -27,15 +27,16 @@ def act(observation):
27	27	
28	28	# Define thresholds for velocities and angles
29	29	x_vel_threshold = 0.1
30	-	y_vel_threshold = 0.1
30	30	+ y_vel_threshold = 0.2
31	31	angle_threshold = 0.1
32	32	ang_vel_threshold = 0.1
33	33	
34	-	# Check if either contact sensor is triggered (landing or crash)
34	34	+ # Ensure that everything is within the thresholds before cutting the engines
35	35	if left_contact == 1 or right_contact == 1:
36	-	return 0 # Cut all engines to settle
36	36	+ if abs(x_vel) < x_vel_threshold and abs(y_vel) < y_vel_threshold and abs(angle) < angle_threshold and abs(ang_vel) < ang_vel_threshold:
37	37	+ return 0 # Cut all engines to settle
37	38	
38	-	# Paragraph order of priority
38	39	+ # Order of priority
39	40	# 1. Reduce y velocity if it's too high (falling too fast)
40	41	if y_vel < -y_vel_threshold:
41	42	return 2 # Main engine on
...	...	
↓		

Figura 4.5: Cambios elaborados por el modelo en el código con mejor resultado.

```

4      - # Constants for safe landing
5      + safe_angle = 0.1 # Safe angle range for landing
6      - safe_angle_vel = 0.1 # Safe angular velocity
7      - safe_vertical_speed = -0.1 # Safe vertical descent speed
8      - safe_horizontal_speed = 0.2 # Safe horizontal speed

9      4 + # Constants
10     5 + safe_angle = 0.1 # Define safe angle range for landing
11     6 + max_safe_speed = -0.2 # Define safe vertical speed range for landing
12     7 + max_horizontal_drift = 0.1 # Define safe horizontal speed at landing
13     8 + max_ang_vel = 0.1 # Define safe angular velocity at landing

14     9
15     10 - # If the spacecraft has already landed
16     10 + # If already contacted the ground
17     11 if left_contact == 1 and right_contact == 1:
18     12 - return 0 # Turn off engines
19     12 + return 0 # Turn off engines, we're landed

20     13
21     14 - # Correct the angle if necessary
22     15 - if abs(angle) > safe_angle or abs(ang_vel) > safe_angle_vel:
23     16 -     if angle > safe_angle or ang_vel > safe_angle_vel:
24     17 -         return 3 # Push right engine to correct left tilt
25     18 -     elif angle < -safe_angle or ang_vel < -safe_angle_vel:
26     19 -         return 1 # Push left engine to correct right tilt

27     14 + # Correct Orientation
28     15 + if angle > safe_angle or ang_vel > max_ang_vel:
29     16 +     return 3 # Push right engine to rotate left
30     17 + elif angle < -safe_angle or ang_vel < -max_ang_vel:
31     18 +     return 1 # Push left engine to rotate right

32     20 19
33     21 20 # Control vertical speed
34     22 - if y_vel < safe_vertical_speed:
35     23 -     return 2 # Push both engines to slow descent
36     21 + if y_vel < max_safe_speed: # If descending too fast
37     22 +     return 2 # Push both engines to slow down

38     24 23
39     25 - # Control horizontal drift only if angle and angular velocity adjustments are minor
40     26 - if abs(angle) < safe_angle and abs(ang_vel) < safe_angle_vel:
41     27 -     if x_vel > safe_horizontal_speed:

42     24 + # Control horizontal drift only if angle and angular velocity are in safe ranges
43     25 + if abs(angle) < safe_angle and abs(ang_vel) < max_ang_vel:
44     26 +     if x_vel > max_horizontal_drift:
45     27         return 1 # Push left engine to move left
46     29 -     elif x_vel < -safe_horizontal_speed:
47     28 +     elif x_vel < -max_horizontal_drift:
48     30 29         return 3 # Push right engine to move right

```

Figura 4.6: Cambios elaborados en el código agente en la iteración 4 por el individuo que ha conseguido obtener la mejor solución de la población.

Capítulo 5

Discusión

En este capítulo se analizarán los resultados presentados en el capítulo anterior y todos los obtenidos durante la elaboración del proyecto. Se discutirán tanto los aspectos prometedores como aquellos que requieren más desarrollo o no han producido los resultados esperados. Además, se alentará a otros investigadores a continuar la línea de trabajo iniciada en este estudio, proporcionando una serie de recomendaciones basadas en las lecciones aprendidas.

5.1 Análisis de los resultados

Al inicio del proyecto, se enfrentaba una considerable incertidumbre debido a la novedad de incorporar modelos LLM a problemas de aterrizaje espacial, dominados tradicionalmente por metodologías de aprendizaje por refuerzo y por métodos de control clásicos. Además, el enfoque de LLM de Generación de Código para la planificación hacía el problema aún más desafiante. A lo largo del proyecto, no se logró alcanzar de manera consistente el baseline establecido por el entorno Lunar Lander. No obstante, el objetivo no era alcanzar la máxima puntuación en este entorno específico, sino demostrar la capacidad de estos modelos para resolver problemas de aterrizaje basados en una cantidad limitada de registros e indicaciones en lenguaje natural, mostrando un potencial de mejora prometedor.

El análisis muestra que el enfoque de function calling con la API de OpenAI ofrece un rendimiento ligeramente mejor que la opción de salida formateada en JSON. Sin embargo, la alta variabilidad y la falta de consistencia en los resultados de cada ejecución influyen en el resultado final.

Las ejecuciones mediante Zero shot demuestran una rápida capacidad de mejora del modelo a partir de un feedback limitado, sin un punto de partida definido. Este enfoque también evidencia que, independientemente de la metodología escogida, existe un estancamiento al intentar afinar el aterrizaje para conseguir un resultado exitoso.

El enfoque evolutivo con una población de LLMs muestra una rápida convergencia en las primeras iteraciones, debido a la colaboración grupal en la exploración de soluciones y la consolidación de los mejores resultados. Sin embargo, se observa que modelos con malas soluciones tienden a copiar la mejor solución de la población, lo que provoca un estancamiento

generalizado. En este caso, el individuo que alcanza la mejor solución es el que permanece mutando en busca de mejores resultados.

Es evidente en los resultados la dependencia de una buena solución inicial, una característica común también en muchos algoritmos evolutivos. Aunque se observa una mejora en los resultados obtenidos, el afinamiento para lograr un aterrizaje exitoso es complejo y el éxito está fuertemente relacionado con un inicio prometedor. La variabilidad en las salidas obtenidas añade un grado de aleatoriedad mayor del deseado.

No obstante, el objetivo inicial del proyecto era presentar una alternativa a los métodos clásicos de aprendizaje por refuerzo, abordando el desafío de la alta cantidad de episodios de entrenamiento requeridos para converger hacia la solución final. Para destacar los resultados obtenidos, la mejor solución alcanzó más de 200 puntos en la tercera iteración, utilizando un total de cinco episodios: un registro inicial y dos de retroalimentación, además de los dos episodios exitosos, cumpliendo así con un total de 5 episodios (serían siete episodios si contamos las cinco iteraciones). En contraste, el modelo DuelingDQN utilizado para generar ejemplos exitosos de aterrizaje requiere alrededor de 600 episodios de entrenamiento para lograr una puntuación de 200 puntos.

Finalmente, es importante destacar que la alta variabilidad también se debe a la naturaleza del problema. El ajuste de parámetros en el código generado puede tener un impacto significativo, generalmente negativo, en el rendimiento del aterrizaje. Esto puede llevar a que soluciones prometedoras obtengan puntuaciones muy bajas y sean descartadas. En consecuencia, los modelos tienden a hacer cambios pequeños, o incluso a no modificar el código, en lugar de realizar grandes cambios basados en el razonamiento, lo que provoca estancamientos.

5.2 Implicaciones y relevancia

La parte más prometedora de los resultados es la capacidad de razonamiento previo a partir de uno o unos pocos ejemplos de aterrizajes, no necesariamente exitosos. Es capaz de extraer las implicaciones de las acciones en el estado y abstraer las claves de un aterrizaje exitoso para retroalimentar la generación de código. Esta habilidad compite con la humana, que suele depender más de la observación directa del aterrizaje para extraer posibles conclusiones. Además, este hallazgo abre la posibilidad de extrapolar la metodología a otros entornos y proyectos con un mayor número de variables en juego, potenciando la capacidad de extracción de información a partir de registros de texto con la ayuda de un correcto formateo.

Asimismo, la capacidad del modelo para trasladar los conocimientos obtenidos al código generado y aplicar estos conocimientos a las secciones específicas con implicaciones directas en la subtarea deseada dentro del proceso de aterrizaje es igualmente prometedora. Esto demuestra una gran comprensión de las relaciones entre el código y las acciones que este desempeña en el entorno.

5.3 Limitaciones

A pesar de los descubrimientos optimistas del apartado anterior, es importante destacar que existe una deficiencia al materializar el buen razonamiento en un código lo suficientemente elaborado para mejorar la tasa de aciertos. La principal limitación del trabajo radica en el modelo utilizado. Se puede observar un gran salto de calidad entre los dos modelos probados, solucionando por completo los problemas de compilación y comprensión del problema que presenta GPT-3.5 Turbo. Sin embargo, aún existen limitaciones en GPT-4.0 que lo alejan del rendimiento humano en esta tarea.

El uso de estos modelos tiene un coste significativo, ya mencionado en secciones anteriores. A pesar de que la API de OpenAI permite una mayor accesibilidad a estos modelos, el coste económico es un factor que debe considerarse y que puede limitar el proyecto. La cantidad de tokens requeridos para el consumo del modelo en cada iteración es elevada, tanto para los prompts de entrada como para la salida generada. Esto obliga a implementar técnicas de compresión en los registros, como se mencionó en el capítulo 3, y a limitar la cantidad de episodios de aterrizaje que pueden ser probados y utilizados como retroalimentación.

5.4 Recomendaciones y futuras líneas de investigación

Este trabajo abre una nueva línea de investigación en la forma de abordar los problemas de agentes autónomos desde la perspectiva de la planificación con modelos LLM para generación de código. Se anima a los investigadores a continuar poniendo a prueba la solución planteada con nuevos modelos que aparezcan y ofrezcan mejoras en el rendimiento, especialmente aquellos que destaquen en la capacidad de razonamiento y generación de código.

Asimismo, se alienta a profundizar en la metodología de incluir conceptos evolutivos con modelos LLM y a explorar las capacidades de cooperación que estos modelos pueden demostrar dentro de una población orientada a resolver un problema común. Una mayor comprensión de los propios modelos también puede facilitar la forma en que se les suministra la información, mejorando el conocimiento sobre las implicaciones de diferentes prompts y estructuras, con el objetivo de resolver el problema de la variabilidad y falta de estabilidad en las soluciones.

Finalmente, se invita a todos aquellos que trabajen en entornos de agentes autónomos a probar el enfoque planteado y a evaluar su capacidad de adaptación a diferentes entornos y tareas.

Capítulo 6

Conclusiones

Este trabajo ha demostrado que la aplicación de modelos de lenguaje a ámbitos tradicionalmente dominados por metodologías tradicionales dentro y fuera del machine learning es una realidad viable. Se evidencia que su faceta como agentes autónomos es prometedora, presentando una alternativa con mucho por explorar en la navegación espacial. Se plantean importantes desafíos sobre las capacidades relacionadas con el razonamiento humano que un modelo de estas características puede abordar en un problema de aterrizaje, enfocándose en la planificación y desde una perspectiva técnica.

Aunque aún existen limitaciones de acceso a los últimos modelos, el avance de la investigación está provocando que el acceso a herramientas más potentes requiera menos recursos. Esto abre las puertas a la incorporación del enfoque planteado en problemas de menor complejidad que puedan beneficiarse del conocimiento inherente de los LLM para el desempeño de determinadas tareas de gestión y planificación.

En definitiva, este es el punto de partida para una línea de investigación que promete una mejora continua y que recibe un gran número de propuestas de evolución directamente relacionadas con el desarrollo acelerado de los LLM. Esto deberá ir acompañado de un mayor entendimiento de los modelos y el perfeccionamiento de técnicas utilizadas en este trabajo para lograr resultados realmente asombrosos en un futuro cercano.

Bibliografía

- [1] Anthropic, 2024. Anthropic api. URL: <https://www.anthropic.com/api>. accessed: 2024-06-18.
- [2] Anthropic, 2024. Claude 3.5: Sonnet - the latest development in language models. URL: <https://www.anthropic.com/news/claude-3-5-sonnet>. accessed: 2024-07-08.
- [3] Devlin, J., Chang, M.W., Lee, K., Toutanova, K., 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. URL: <https://arxiv.org/abs/1810.04805>, [arXiv:1810.04805](#).
- [4] Li, C., Liang, J., Zeng, A., Chen, X., Hausman, K., Sadigh, D., Levine, S., Fei-Fei, L., Xia, F., Ichter, B., 2023. Chain of code: Reasoning with a language model-augmented code emulator. URL: <https://arxiv.org/abs/2312.04474>, [arXiv:2312.04474](#).
- [5] Ma, Y.J., Liang, W., Wang, G., Huang, D.A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., Anandkumar, A., 2024. Eureka: Human-level reward design via coding large language models. URL: <https://arxiv.org/abs/2310.12931>, [arXiv:2310.12931](#).
- [6] Meta AI, 2024. Meta llama 3: Advancing language model capabilities. URL: <https://ai.meta.com/blog/meta-llama-3/>. accessed: 2024-07-08.
- [7] Mistral AI, 2024. Mistral ai technology. <https://mistral.ai/technology/#models>. Accessed: 2024-07-11.
- [8] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. nature 518, 529–533.
- [9] OpenAI, 2023. Gpt-3 apps URL: <https://openai.com/index/gpt-3-apps/>. accessed: 2024-06-15.
- [10] OpenAI, 2024a. Hello gpt-4o URL: <https://openai.com/index/hello-gpt-4o/>. accessed: 2024-06-16.
- [11] OpenAI, 2024b. Lunarlander environment. URL: https://gymnasium.farama.org/environments/box2d/lunar_lander/. accessed: 2024-06-15.
- [12] OpenAI, 2024c. Openai api documentation. URL: <https://platform.openai.com/docs/overview>. accessed: 2024-06-16.

- [13] OpenAI, 2024d. Openai assistants overview. URL: <https://platform.openai.com/docs/assistants/overview>. accessed: 2024-06-19.
- [14] Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L., 2018. Deep contextualized word representations. URL: <https://arxiv.org/abs/1802.05365>, [arXiv:1802.05365](#).
- [15] Rodriguez-Fernandez, V., Carrasco, A., Cheng, J., Scharf, E., Siew, P.M., Linares, R., 2024. Language models are spacecraft operators. [arXiv:2404.00413](#).
- [16] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015. Trust region policy optimization, in: International conference on machine learning, PMLR. pp. 1889–1897.
- [17] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. [arXiv:1707.06347](#).
- [18] Van Hasselt, H., Guez, A., Silver, D., 2016. Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI conference on artificial intelligence.
- [19] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems 30.
- [20] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., Freitas, N., 2016. Dueling network architectures for deep reinforcement learning, in: International conference on machine learning, PMLR. pp. 1995–2003.
- [21] Wang, Z., Zhang, H., Li, C.L., Eisenschlos, J.M., Perot, V., Wang, Z., Miculicich, L., Fujii, Y., Shang, J., Lee, C.Y., Pfister, T., 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. URL: <https://arxiv.org/abs/2401.04398>, [arXiv:2401.04398](#).
- [22] Watkins, C.J., Dayan, P., 1992. Q-learning. Machine learning 8, 279–292.
- [23] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D., 2023. Chain-of-thought prompting elicits reasoning in large language models. URL: <https://arxiv.org/abs/2201.11903>, [arXiv:2201.11903](#).
- [24] Zheng, H.S., Mishra, S., Chen, X., Cheng, H.T., Chi, E.H., Le, Q.V., Zhou, D., 2024. Take a step back: Evoking reasoning via abstraction in large language models. URL: <https://arxiv.org/abs/2310.06117>, [arXiv:2310.06117](#).
- [25] Zhou, Y., Geng, X., Shen, T., Tao, C., Long, G., Lou, J.G., Shen, J., 2023. Thread of thought unraveling chaotic contexts. URL: <https://arxiv.org/abs/2311.08734>, [arXiv:2311.08734](#).

Anexos

.1 Código inicial

El modelo recibe inicialmente el código base junto con los registros generados a partir de su ejecución en el entorno correspondiente. Este código se encapsula en una variable de texto para poder incluirlo en los diversos prompts iniciales, al igual que se hace con los registros.

Listing 1: Código inicial que recibe el modelo..

```
import random

def act(observation):
    '''
        The function that codifies the action to be taken in each
        instant of time.

        Args:
            observation (numpy.array):
                "description": "The state of the environment after the
                    action is taken.",
                "positions": {
                    "0": "X position",
                    "1": "Y position",
                    "2": "X velocity",
                    "3": "Y velocity",
                    "4": "Angle",
                    "5": "Angular velocity",
                    "6": "Left contact sensor",
                    "7": "Right contact sensor"
                },
                "min_values": [-1.5, -1.5, -5.0, -5.0, -3.14, -5.0, 0,
                    0],
                "max_values": [1.5, 1.5, 5.0, 5.0, 3.14, 5.0, 1, 1]

        Returns:
            Integer : The action to be taken.
```

```

        "options": {
            '0': "Switch off engines",
            '1': "Push left engine",
            '2': "Push both engines (upwards)",
            '3': "Push right engine"
        },
        ...
    return random.randint(0, 3)

```

.2 Ejemplos de prompt

En esta sección se presentan ejemplos de los prompts utilizados tanto para la creación de los agentes como para la interacción con ellos.

Para inicializar el asistente, es necesario incluir una descripción detallada de la tarea que debe desempeñar, así como las instrucciones precisas para llevarla a cabo de manera correcta. Una vez el asistente es creado, se le alimenta con un mensaje inicial proporcionado por el usuario. En el caso de **Zero Shot**, este mensaje no contiene información adicional sobre otras ejecuciones en el entorno, como se puede observar en el código 2.

Listing 2: Prompt para el modelo con Zero shot

```

DESCRIPTION = "You are an expert programmer in Python. Your
specialty is to generate the code responsible for making
decisions about actions to be taken in various spacecraft
landing environments. The objective is to land the spacecraft
within a target zone in the shortest possible time and very
gently. A scoring system is used to evaluate the landings,
which must be maximized."

INSTRUCTIONS = f"""Your task is:
1. Analyze and reason about the results received in the
last landing attempts. This is the schema of the landing
logs: {log_description}.
2. Your goal is to be able to make the appropriate decision
based on the results of previous iterations. You must
code the decision making based on your reasoning in a
Python function.
3. Use these tips to get a good landing:
- First you have to stabilize the falling ship and then
land gently.
- Use all observations when deciding what action to
take at any given moment. Both position and
velocities must be taken into account to know how the
ship is doing and towards which states it is heading

```

```

        . All of these must be considered to achieve
        stability.
    - Landing zone is in the central area of the x-axis.
    - Learn how actions taken affect the future states of
      the spacecraft in the logs of past events so that you
      can take this into account when developing code to
      reach the landing zone.
4. Based on the logs you should improve the code generated
   in the 'act' function in the last iteration, seeking to
   maximize the score received and generate a higher quality
   code.
5. Save the code of the act function in the file 'Action.py
   ' using the tools given.
6. Improve your results and correct any errors you may have
   generated in your last code if they exist."""

NAME = "Spacecraft Landing Master"

initial_msg = f"Take a deep breath and think step by step.
  Analyze the results and improve the code."

```

El siguiente ejemplo (3) contiene un prompt más primitivo utilizado durante el desarrollo del proyecto. Este prompt incluye una serie de consejos generales sobre el objetivo del aterrizaje, en lugar de la descripción del score. Esta metodología fue descartada posteriormente para evaluar correctamente el razonamiento completo del modelo en base a los registros.

Listing 3: Prompt de fases de desarrollo anteriores

```

DESCRIPTION = "You are an expert programer in Pyhton. Your
  specialty is to generate the code responsible for making
  decisions about actions to be taken in various spacecraft
  landing environments.The objective is to land the spacecraft
  within a target zone in the shortest possible time and very
  gently. A scoring system is used to evaluate the landings,
  which must be maximized."
INSTRUCTIONS = f"""Your task is:
1. Analyze and reason about the results received in the
   last landing attempts. This is the schema of the landing
   logs: {log_description}.
2. Your goal is to be able to make the appropriate decision
   based on the results of previous iterations. You must
   code the decision making based on your reasoning in a
   Python function.
3. Use these tips to get a good landing:
   - First you have to stabilize the falling ship and then
     land gently.

```

```

- Use all observations when deciding what action to
  take at any given moment. Both position and
  velocities must be taken into account to know how the
  ship is doing and towards which states it is heading
  . All of these must be considered to achieve
  stability.
- Landing zone is in the central area of the x-axis.
- Learn how actions taken affect the future states of
  the spacecraft in the logs of past events so that you
  can take this into account when developing code to
  reach the landing zone.
4. Based on the logs you should improve the code generated
  in the 'act' function in the last iteration, seeking to
  maximize the score received and generate a higher quality
  code.
5. Save the code of the act function in the file 'Action.py
  ' using the tools given.
6. Improve your results and correct any errors you may have
  generated in your last code if they exist."""

NAME = "Spacecraft Landing Master"

initial_msg = f"Take a deep breath and think step by step. This
  is the record of an example of a successful landing in this
  environment, but under other conditions: {success_logs}. This
  is the code of the initial function: {agent_initial_code}
  and these are the execution logs of one landing attempt: {
  initial_logs}. Analyze the results and improve the code."

```

Para utilizar el modo JSON, las indicaciones que recibe el asistente en el momento de su creación difieren ligeramente. En lugar de solicitar al modelo que lleve a cabo una llamada a la función al final del proceso, se le debe solicitar una respuesta en formato JSON (ver 4).

Listing 4: Prompt utilizado en la generación de agente con JSON mode

```

DESCRIPTION = "You are an expert spacecraft landing agent. Your
  specialty is to take the appropriate action at each instant
  of time based on the environment and state of the spacecraft.
  To achieve this, you put your knowledge base and the
  knowledge you acquire by analyzing each execution into a
  python function in charge of directing the landing."
INSTRUCTIONS = f"""To complete the task you must follow the
  following steps and indications:
1. Analyze and reason about the records received in the last
  landing attempts to learn how the spacecraft behaves in the
  environment. Keep in mind that conditions may vary.

```

```

2. To measure how good a landing is, a scoring system that
   appears in the logs is used, your role is to maximize it. To
   do so, keep the following in mind:
   - Increases/decreases the closer/further the spacecraft is
     from the landing area (both axes).
   - Increased/decreased the slower/faster the spacecraft is
     moving.
   - Decreased the more the spacecraft is tilted.
   - Decreased by 0.03 points each frame a side engine is
     firing and 0.3 points for the center engine.
   - Receive 100 points for a successful landing and lose them
     for crashing.
   - A succesful landing must exceed 200 points.
3. This is the log schema:\n {log_description}
4. Add your decision code to the 'act' method and locate errors
   in it if the landing is unsuccessful.
5. Provide output in valid JSON format. The data schema should
   be as follows: {json.dumps(json_shcema_response)}.
6. Generate the code of the act function in the file 'Action.py'
'''

NAME = "Spacecraft Landing Master"

initial_msg = "This is the record of an example of a successful
landing in this environment, but under other conditions: {
success_logs}. This is the code of the initial function: {
agent_initial_code} and these are the execution logs of one
landing attempt: {initial_logs}. What does each value and
each change in the observations mean? What effects does each
action have on the current state observations? What are the
steps to follow to achieve a successful landing?. Take a deep
breath and reason step-by-step. After reasoning analyze the
results, learn and make better code."

```

Tras cada iteración, es importante devolverle al modelo los resultados del código generado, junto con las indicaciones pertinentes para que evolucione el código en base a sus observaciones (ver 5). En el caso de estar ejecutando la opción que incluye un enfoque evolutivo con una población de asistentes (texto 6, debe incluir las instrucciones con la mejor solución obtenida dentro de la población junto con sus indicaciones de mejora individual vistas en el mensaje anterior.

Listing 5: Retroalimentación tras cada iteración

```

msg = f"""These are the logs generated by your last code: {logs
}. Analyze the performance of the spacecraft and how it

```

```
differs from the desired result. Be guided by the scoring
system. Identify the cause of errors in your code and modify
it without fear of making major changes. Think deeply about
the priorities of your code and how to order and combine them
correctly to achieve success. Remember that the code is a
reflection of your reasoning and the way you approach the
problem."""
```

Listing 6: Retroalimentación tras cada iteración con el engoque genético

```
msg_resultado = f"The best result for the population was {
    resultados[1]} achieved with the following code: {resultados[
    2]}\ and the following logs:{resultados[3]}".
Compare it with yours and draw conclusions. Re-generate the
code step by step and think deeply to improve both results."
```