



Universidad Politécnica
de Madrid



**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Resolución de Problemas de Formación
de Equipos en Juegos con "Monte-Carlo
Tree Search" y Modelos de Lenguaje**

Autor: Geovanna Aguilar Avilés

Tutor: José María Barambones Ramírez

Madrid, junio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Resolución de Problemas de Formación de Equipos en Juegos con
"Monte-Carlo Tree Search" y Modelos de Lenguaje

Junio 2025

Autor: Geovanna Aguilar Avilés

Tutor:

José María Barambones Ramírez

Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de
Software

ETSI Informáticos

Universidad Politécnica de Madrid

A mi madre,
por su amor, su paciencia y su confianza en mí.

Resumen

En la última década, el avance de la inteligencia artificial ha abierto nuevas posibilidades en la resolución de problemas complejos, incluidos los juegos de estrategia. En este contexto, VGC AI 2025 ofrece un entorno competitivo en el que cada participante debe crear un agente capaz de seleccionar equipos Pokémon de la manera más efectiva. Aunque las bases del concurso no exigen el uso de modelos de lenguaje, este trabajo aprovecha el marco de VGC AI 2.0 para incorporar dos LLM de última generación, Llama 3.1 y DeepSeek-distill, y evaluar su impacto en la construcción de equipos. También se aprovecha la experiencia de la investigación de la edición anterior, en la que se probaron distintos agentes con otros LLM, para usar ese conocimiento y avanzar hacia soluciones más efectivas.

Para ello, se comparan tres enfoques principales. El primero emplea únicamente Monte Carlo Tree Search para explorar combinaciones de equipos mediante simulaciones aleatorias. El segundo añade a Llama 3.1 como fuente de razonamiento avanzado, utilizando técnicas de text prompting que siguen principios de ingeniería de prompts responsables. El tercero integra DeepSeek-distill, la versión destilada de Llama 3.1, a fin de mantener el mismo tipo de razonamiento estratégico con un coste de inferencia reducido. Además, se aplica fine tuning para simplificar las instrucciones que se envían a los modelos y se registra en cada batalla un historial de equipos ganadores y perdedores junto con el equipo utilizado y la evolución del ELO. Este registro contextual actúa como un esquema de RAG simplificado, alimentando a los LLM con información relevante de partidas anteriores.

Con estas tres estrategias se pretende determinar qué combinación ofrece el mejor equilibrio entre calidad de las decisiones y velocidad de selección dentro del Championship Track de VGC AI 2.0. Los resultados de esta comparación servirán para entender cómo los modelos de lenguaje, cuando se utilizan de forma adecuada, pueden mejorar significativamente el proceso de construcción de equipos en juegos de estrategia.

Abstract

Over the past decade, advances in artificial intelligence have opened new possibilities for solving complex problems, including those found in strategy games. In this context, VGC AI 2025 presents a competitive environment in which each participant must create an agent capable of selecting Pokémon teams as effectively as possible. Although the competition rules do not require the use of language models, this work leverages the VGC AI 2.0 framework to incorporate two state-of-the-art LLMs, Llama 3.1 and DeepSeek, and evaluate their impact on team building. It also draws on the experience gained from the previous edition, in which various agents using other LLMs were tested, in order to build on that knowledge and move toward more effective solutions.

To this end, three main approaches are compared. The first relies solely on Monte Carlo Tree Search (MCTS) to explore team combinations through random simulations. The second adds Llama 3.1 as a source of advanced reasoning, using text prompting techniques aligned with principles of responsible prompt engineering. The third integrates DeepSeek-distill, the distilled version of Llama3.1, in order to maintain a similar level of strategic reasoning at a lower inference cost. In addition, fine-tuning is applied to simplify the instructions sent to the models, and each battle logs a history of winning and losing teams, along with the team used and its ELO progression. This contextual log functions as a simplified Retrieval-Augmented Generation (RAG) mechanism, feeding the LLMs with relevant information from previous matches.

These three strategies aim to identify which combination offers the best balance between decision quality and selection speed within the Championship Track of VGC AI 2.0. The results of this comparison will help demonstrate how, when properly integrated, language models can significantly improve the team-building process in strategy games.

Tabla de contenidos

1	Introducción	1
1.1	Objetivo principal	1
1.2	Solución propuesta	1
1.3	Cambios con respecto a VGC AI 1.0	1
1.4	Elaboración del proyecto	2
2	Bases Conceptuales	3
2.1	Panorama actual de los LLMs	3
2.2	<i>Large Language Model</i> (LLM)	3
2.2.1	Arquitectura <i>Trasformer</i>	4
2.2.2	Llama 3.1	5
2.2.3	DeepSeek R1	5
2.3	Técnicas empleadas	6
2.3.1	Text Prompting	6
2.3.2	<i>Fine-tuning</i>	7
2.3.3	Aplicación concepto RAG	8
3	Desarrollo	9
3.1	VGC AI 2.0	9
3.1.1	Composición del framework	9
3.1.2	Categorías	10
3.1.3	Championship Track	11
3.1.3.1	Parámetros utilizados	11
3.1.3.2	Fase de construcción de equipos	11
3.1.3.3	Rondas de combate y actualizaciones de ELO	12
3.1.3.4	Clasificación final	12
3.1.3.5	Resumen de configuración	12
3.2	Agentes	13
3.2.1	Agente base	13
3.2.2	Agente MCTS	13
3.2.2.1	Esquema MCTS	14
3.2.2.2	Selección y expansión inicial	14
3.2.2.3	Simulación	15
3.2.2.4	Backpropagation	15
3.2.3	Agente híbrido LLM	16
3.2.3.1	MCTS+Llama 3.1	16
3.2.3.2	MCTS+DeepSeek R1 Distill Llama 3.1	17
3.3	Entorno simulaciones	17
4	Resultados	18
4.1	Resultados MCTS	18

4.2	Resultados Llama 3.1	19
4.3	Resultados DeepSeek R1 Distill Llama 3.1	20
4.4	Análisis final	20
4.4.1	Comparación de resultados	21
4.4.2	Discusión de resultados.....	21
5	Conclusiones	23
6	Análisis de Impacto	24
7	Bibliografía	25
8	Anexos.....	27
8.1	Prompt fine tuning en LM Studio	27
8.2	Prompt en script	27
8.3	Ejemplo log DeepSeek	29

1 Introducción

1.1 Objetivo principal

El objetivo principal de este proyecto es diseñar y evaluar agentes capaces de obtener el mejor desempeño posible en la competición *Video Game Championship* (VGC) AI 2025 [1], superando los resultados alcanzados en la edición anterior con VGC AI 1.0 [2], que sirvió de base para la investigación del Trabajo de Fin de Grado realizado en 2024 [3]. Para ello, se explorarán tres enfoques: MCTS puro [4], MCTS combinado con Llama 3.1 [5] y MCTS combinado con DeepSeek-R1 destilada de Llama 3.1 [6]. El propósito es demostrar cómo, aprovechando la capacidad de razonamiento del modelo DeepSeek y las mejoras introducidas en el *framework* VGC AI 2.0 [7], es posible optimizar la selección de equipos, adaptar estrategias en tiempo real y, por último, superar la clasificación obtenida del año pasado.

1.2 Solución propuesta

Para esta edición, se plantea un agente híbrido que combina *Monte Carlo Tree Search* (MCTS) con DeepSeek-R1, en su versión destilada de Llama 3.1. El *prompting* incorpora un enfoque RAG [8] que acumula el historial de equipos ganadores y perdedores, junto con el roster y el resultado anterior, para proporcionar un mejor contexto a DeepSeek. El modelo propone un equipo en base a ese contexto y, si falla, la propia estructura de MCTS genera una alternativa válida.

Así, el agente resultante conjuga la búsqueda de MCTS, la cual simula múltiples escenarios de batalla, con el razonamiento avanzado de DeepSeek. MCTS explora el espacio de soluciones mientras que DeepSeek evalúa, en cada nodo, qué candidatos muestran mayor probabilidad de éxito, apoyándose en experiencias previas. El objetivo es aprovechar este esquema mejorado para obtener un proceso de construcción de equipos más efectivo en VGC AI 2.0 y, de ese modo, superar el rendimiento alcanzado en la edición pasada.

1.3 Cambios con respecto a VGC AI 1.0

En comparación con VGC AI 1.0, el entorno de este año presenta cambios importantes que han obligado a reevaluar el trabajo previo. Aunque se partió del progreso del anterior proyecto, la actualización del *framework* implicó modificar desde la raíz la generación de equipos. Entre las principales diferencias destacan la modificación de la estructura, con nuevos componentes, una nueva función de cálculo de ELO [9], el nuevo método de decisión de los competidores y la estructura interna de las políticas de torneo, lo que invalidó muchas de las heurísticas diseñadas anteriormente. Por ello fue necesario reconstruir un MCTS adaptado al nuevo API y, además, diseñar una versión híbrida que invoque tanto a Llama 3.1 como a DeepSeek-R1 para compensar las limitaciones de cada modelo. Estas modificaciones, junto con ajustes en el manejo del historial de partidos y la gestión de la información de roster,

supusieron un esfuerzo de análisis adicional para garantizar la compatibilidad con VGC AI 2.0 y recuperar lo alcanzado en la edición previa.

1.4 Elaboración del proyecto

Está estructurado en; un análisis previo a la programación de los agentes, donde se estudió el estudio del arte de los LLMs y sus técnicas (Capítulo 2); el desarrollo del campeonato, donde se explica las reglas y diferencias en la versión de este año con respecto al año anterior y la lógica empleada en los agentes MCTS y LLMs (Capítulo 3); finalizando con el análisis de resultados obtenidos (Capítulo 4) y conclusiones (Capítulo 5).

2 Bases Conceptuales

2.1 Panorama actual de los LLMs

En los últimos años, los Modelos de Lenguaje de Gran Escala (LLMs) [10] se han convertido en una herramienta clave en el desarrollo de sistemas inteligentes, con aplicaciones que van desde la generación de texto hasta el razonamiento complejo. Entre los modelos más destacados se encuentran GPT-4 [11], Claude [12], Gemini [13], Qwen [14], y especialmente Llama 3.1, publicado en junio de 2024 por Meta, que ha ganado popularidad en entornos abiertos por su rendimiento competitivo y eficiencia [5].

Un avance relevante en este contexto es DeepSeek, desarrollado por la empresa china DeepSeek AI [15], que ha introducido capacidades avanzadas de razonamiento como una de sus principales innovaciones en 2025. Estos modelos se entrenan mediante técnicas de refuerzo sin supervisión humana intensiva, lo que permite reducir costes de entrenamiento sin sacrificar calidad. En particular, DeepSeek-R1-Distill-Llama-8B, una versión destilada basada en Llama 3.1, combina eficiencia computacional con un razonamiento sólido, lo que la convierte en una alternativa ideal para entornos con recursos limitados [6].

En este trabajo se trabajará específicamente con MCTS, Llama 3.1 y su versión destilada con DeepSeek-R1, con el objetivo de integrar sus capacidades de razonamiento en el desarrollo de agentes inteligentes. Su uso busca aprovechar las ventajas de los LLMs en contextos computacionalmente restringidos, manteniendo un alto nivel de comprensión, generación de lenguaje y toma de decisiones estratégicas.

2.2 Large Language Model (LLM)

Los Modelos de Lenguaje de Gran Escala (*Large Language Models*, o LLMs) son sistemas basados en aprendizaje automático entrenados para procesar y generar lenguaje natural a partir de grandes cantidades de texto. Su objetivo principal es modelar la probabilidad de aparición de una palabra o símbolo dado un contexto previo, lo que les permite realizar tareas como predicción de texto, resumen automático o generación de respuestas en diálogos.

Uno de los hitos que marcó el desarrollo de los LLM fue la introducción de la arquitectura *Transformer*, que se basa en mecanismos de autoatención o *self-attention*, para identificar relaciones de dependencia entre palabras, incluso a largas distancias [16]. Esta arquitectura ha sido adoptada como base para la mayoría de modelos recientes, incluyendo GPT, LLaMA o DeepSeek.

2.2.1 Arquitectura *Transformer*

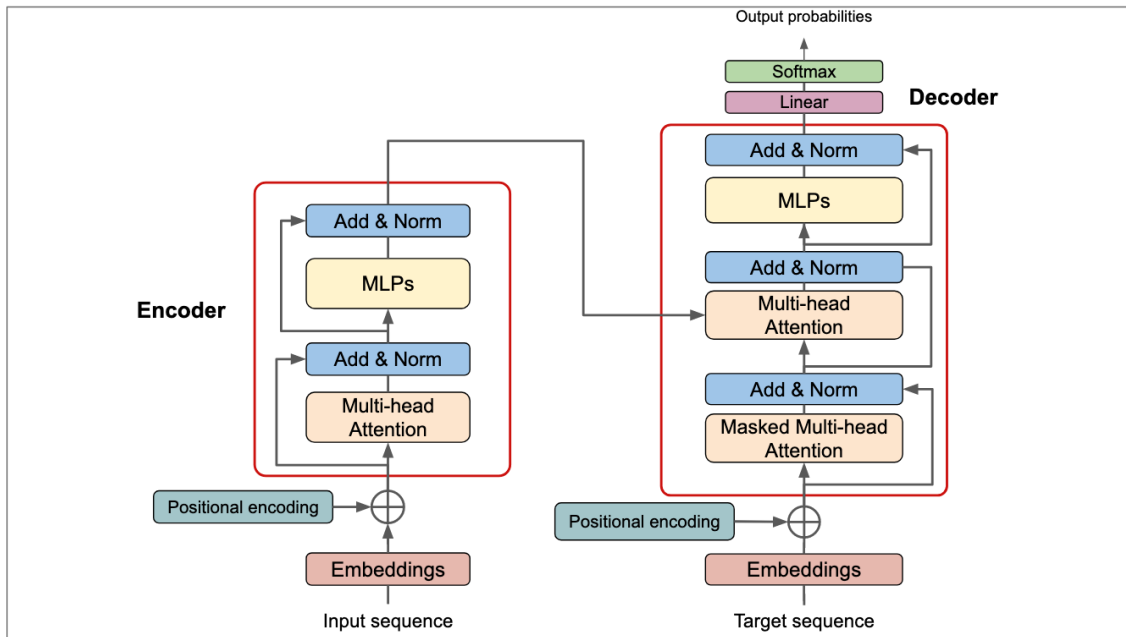


Figura 1. Arquitectura Transformer LLM

El *encoder* recibe una secuencia de entrada, por ejemplo, una frase, y la convierte en una representación contextual, mientras que el *decoder* utiliza esa representación para generar una secuencia de salida [17].

El proceso del *encoder* consta de:

- **Embeddings y Positional Encoding**
Cada palabra se transforma en un vector y se le añade información sobre su posición en la secuencia.
- **Multi-head Attention**
Permite que el modelo atienda a diferentes partes de la entrada al mismo tiempo, capturando relaciones entre palabras.
- **Add & Norm**
Combina la salida anterior con la entrada original (conexión residual) y la normaliza para estabilizar el aprendizaje.
- **MLPs (Feedforward layers)**
Aplican una transformación individual a cada token de forma no lineal.

En el *decoder* se utilizan los mismos componentes, pero para diferentes propósitos:

- **Embeddings y Positional Encoding**
Igual que en el encoder, pero aplicado al texto que se quiere generar.
- **Masked Multi-head Attention**

Aquí se enmascara las posiciones futuras para no ver el siguiente token durante la generación.

- **Multi-head Attention**
Con conexión al *encoder*, lo que permite al *decoder* consultar la información procesada por el *encoder*.
- **Add & Norm y MLPs**
Igual que en el *encoder*, con conexiones residuales y capas de normalización.
- **Capa final**
Consta de una capa, *Linear*, que transforma los vectores del modelo en dimensiones del vocabulario, y *Softmax*, que convierte esas salidas en una distribución de probabilidad sobre todas las posibles palabras.

2.2.2 Llama 3.1

Llama 3.1, desarrollado por Meta AI, es una evolución consolidada dentro de los modelos de lenguaje de gran escala (LLMs), destacando por su equilibrio entre rendimiento, tamaño y eficiencia de inferencia [5]. A diferencia de otros modelos más pesados y cerrados, Llama 3.1 ha sido diseñado para tareas instructivas multilingües, ofreciendo capacidades competitivas en generación de texto, comprensión contextual y razonamiento estructurado. Su versión 8B Instruct representa un punto óptimo para entornos con recursos limitados, permitiendo respuestas coherentes y rápidas, incluso cuando se requiere mantener información compleja en contexto [18].

En este trabajo se reutiliza meta-llama-3.1-8b-instruct con el propósito de mejorar los resultados obtenidos en la edición anterior de VGC AI [3]. Esta vez, se incorpora dentro de un *framework* actualizado, pero con el mismo concepto de la edición pasada [7].

Aprovechando la experiencia del año anterior, este trabajo propone un análisis más profundo y comparativo de Llama 3.1 frente a la técnica de razonamiento introducida por DeepSeek-R1, centrándose en su capacidad real para aportar valor en tareas complejas de toma de decisiones, no solo en generación de lenguaje. Esto permitirá establecer criterios más sólidos para determinar en qué condiciones el uso de LLMs como Llama 3.1 resulta verdaderamente beneficioso en entornos de competición como el VGC AI.

2.2.3 DeepSeek R1

DeepSeek-R1 es un modelo de lenguaje de gran escala (LLM) desarrollado por DeepSeek AI, centrado en el razonamiento avanzado. Entrenado mediante aprendizaje por refuerzo sin supervisión humana directa, ha sido diseñado específicamente para resolver tareas complejas como lógica, matemáticas y programación. Una de sus principales aportaciones es la no necesidad de fine-tuning supervisado, lo que reduce los costes de entrenamiento sin comprometer la calidad del razonamiento [15].

En pruebas comparativas, DeepSeek-R1 ha mostrado un rendimiento sobresaliente en benchmarks, superando a modelos como GPT-4 base en varias tareas específicas de razonamiento estructurado y generación de código, y alcanzando puntuaciones de estado del arte con un menor consumo computacional.

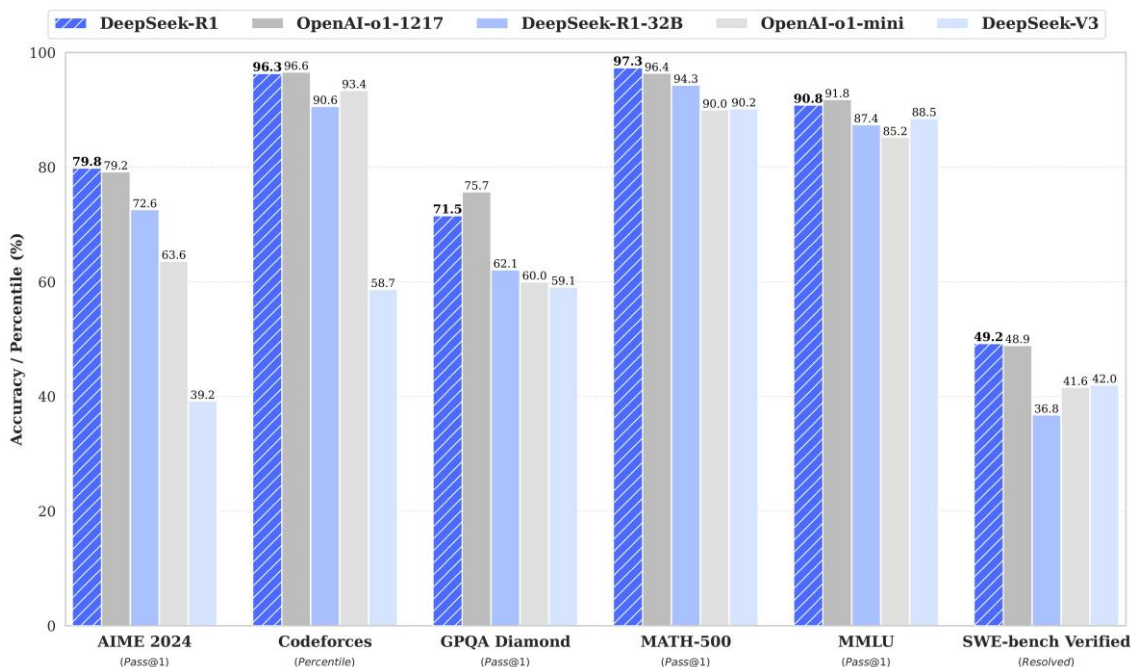


Figura 2. Resultados benchmarks

Para facilitar su uso en entornos con restricciones de hardware, se ha lanzado DeepSeek-R1-Distill-Llama-8B, una versión destilada que transfiere las capacidades de razonamiento del modelo original a una arquitectura basada en Llama 3.1 de 8B parámetros. Esta versión mantiene un buen equilibrio entre velocidad, coste computacional y calidad de inferencia [18], siendo ideal para integraciones en sistemas como el *framework* VGC AI 2.0.

En este proyecto, se utilizará precisamente DeepSeek-R1-Distill-Llama-8B, aprovechando su bajo coste de inferencia y su razonamiento contextual optimizando así la generación de equipos competitivos en tiempo real.

2.3 Técnicas empleadas

2.3.1 Text Prompting

El *text prompting* es la práctica de formular preguntas o instrucciones para un modelo de lenguaje mediante un mensaje de texto estructurado, es decir, se proporciona al LLM un *prompt*, una instrucción en formato texto, que describe el contexto y la tarea deseada, para que el modelo genere la respuesta más pertinente. Esta técnica se ha desarrollado con metodologías de *Prompt Engineering*, cuyo objetivo es diseñar mensajes claros y completos que orienten al modelo hacia un comportamiento específico [19].

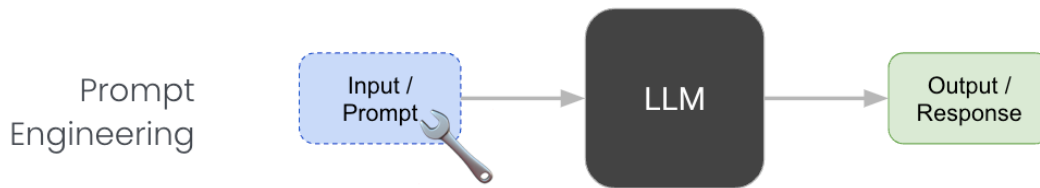


Figura 3. Prompt Engineering

En particular, durante el curso de “IA Responsable y Prompt Engineering” [20] realizado durante el proyecto, se aprendieron buenas prácticas como:

- Incluir en el *prompt* únicamente la información esencial (roster, equipo anterior, resultado de la batalla) para evitar ruido.
- Usar un lenguaje neutro y consistente, de modo que el modelo comprenda sin ambigüedades qué se le solicita. Por ejemplo, utilizar oraciones positivas para ayudar a la IA a pensar directamente en la petición, en lugar de la negación, y así evitar el uso innecesario de negaciones cuando sea posible para evitar la pérdida de contexto.
- Utilizar instrucciones directas y citadas como puede ser el uso de comillas al solicitar el formato de índices (‘---i1, i2, i3---’) que minimicen la probabilidad de respuestas fuera de formato.

En este trabajo, se ha mejorado la técnica de *text prompting* al aplicarse esos principios: cada vez que los LLMs reciben un *prompt*, se les incluye el roster actual de Pokémon (con tipos y movimientos), el equipo usado en la ronda anterior, el resultado (ganar o perder) y datos adicionales como el nuevo ELO. Gracias a ello, el modelo dispone de una visión completa del estado del torneo y puede generar selecciones de equipo más ajustadas sin necesidad de recodificar lógica compleja en el script.

2.3.2 Fine-tuning

El *fine-tuning* es el proceso de refinar un modelo preentrenado utilizando ejemplos concretos para que adapte su comportamiento a una tarea particular. En LM Studio [21], herramienta empleada para utilizar LLMs, se aprovechó esta capacidad para cargar los LLMs, Llama y DeepSeek, ya entrenadas con las instrucciones generales de la competición. De este modo, el código no necesita volver a indicar cada vez las partes fijas del mensaje, sino que solo envía la información que cambia. El modelo, gracias a ese ajuste previo, ya sabe cómo interpretar y completar correctamente las peticiones, consiguiendo así:

- Evitar repetir las instrucciones principales en cada llamada.
- Reducir la carga en el código dado que el script solo necesita pasar la información dinámica (*roster* actualizado, equipo anterior, historial resumido, etc.).

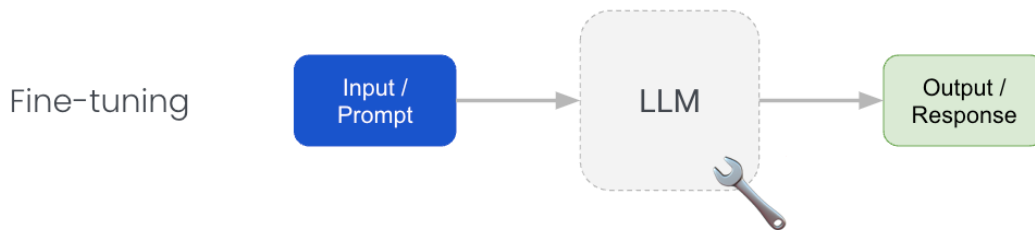


Figura 4. Fine-tuning

Así, el *fine-tuning* aplicado en LM Studio permitió realizar la parte repetitiva de las instrucciones al modelo previamente ajustado, de modo que en el código únicamente se transmiten los datos actualizados de equipo y resultados sin necesidad de repetición ni sobrecarga de *tokens* (unidad de texto en LLMs).

2.3.3 Aplicación concepto RAG

El RAG (*Retrieval-Augmented Generation*) es un enfoque en el que, junto con el prompt principal, se recupera información relevante de una base externa (por ejemplo, un repositorio de documentos, un historial de partidas o datos estructurados) para enriquecer la entrada al modelo. En un escenario típico de RAG, el sistema realiza previamente una búsqueda (*retrieval*) y aporta ese lote de texto como contexto para que el modelo genere resultados basados tanto en su conocimiento interno como en datos actualizados [8].

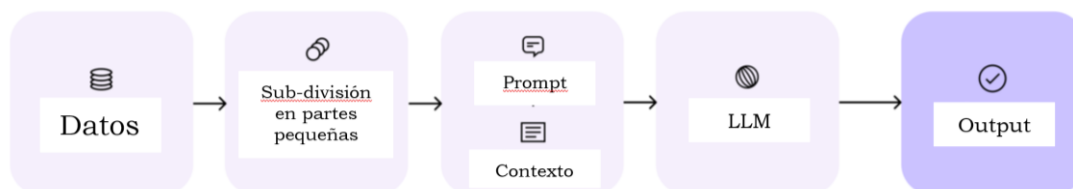


Figura 5. Esquema RAG

En este proyecto no se utilizó RAG, pues LM Studio no ofrece esta funcionalidad vía API. En su lugar, se optó por:

- Registrar en logs los equipos ganadores y perdedores de cada enfrentamiento.
- Mantener el último equipo utilizado y el ELO obtenido.

Aunque no sea RAG “puro” a nivel técnico, este mecanismo cuyo objetivo es tener un historial local cumple la misma función, dado que aporta al modelo datos actualizados que no se encuentran en su conocimiento estático, permitiéndole razonar sobre qué combinaciones han funcionado o fracasado previamente.

3 Desarrollo

3.1 VGC AI 2.0

La edición 2.0 del VGC AI introduce un conjunto revisado de reglas y estructuras con respecto a la versión 1.0, manteniendo la esencia de una competición de agentes que forman equipos de Pokémon para enfrentarse en rondas sucesivas, pero actualizando la lógica de cálculo de ELO, las políticas de entrada y salida de equipos, y el flujo de ejecución de los distintos circuitos. En VGC AI 2.0, cada agente debe construir su equipo de seis Pokémon a partir de un *roster* fijo antes de cada ronda. A continuación, ese equipo compite durante un número determinado de combates, y su rendimiento se traduce en una puntuación ELO que servirá para clasificarlo frente a los demás.

En este trabajo, se utiliza el *framework* proporcionado de la versión 2.0, pero con las reglas del 1.0, es decir, en lugar de realizar los experimentos con equipos de seis Pokémon, se realizan con tres. Esto es, para poder realizar un análisis comparativo con los resultados obtenidos en la anterior versión, de manera que no difieran las reglas, a pesar del cambio de la API empleada.

3.1.1 Composición del framework

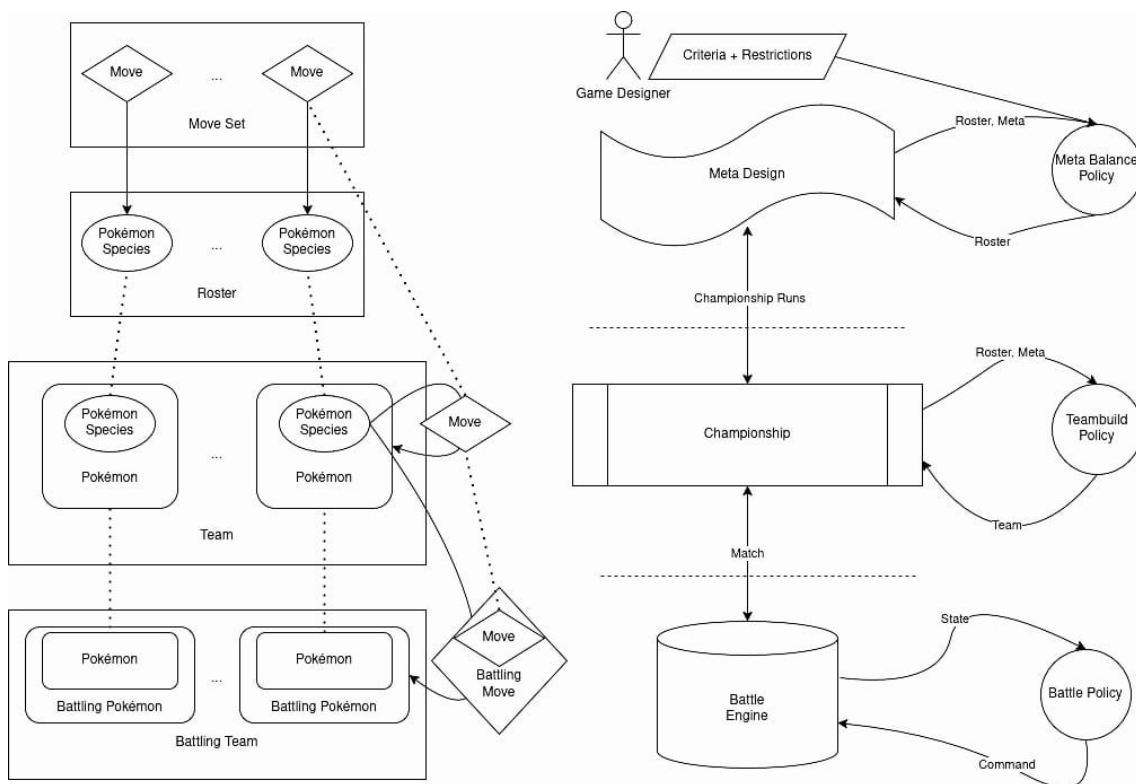


Figura 6. Diagrama flujo framework VGC AI 2.0

Se usa un objeto *Pokémon* que se utiliza tanto para definir el *roster*, el conjunto de opciones disponibles durante la construcción de equipos, como para representar el componente central durante las batallas. Su comportamiento varía en función del contexto: en la fase de *roster* solo se tiene en cuenta la información del ejemplar base; durante la construcción del equipo, deben configurarse estadísticas personalizadas; y finalmente, durante el combate, el *Pokémon* adquiere atributos adicionales relevantes para la dinámica de la batalla.

El sistema se organiza en tres capas de abstracción. La primera, y más baja, es la capa de batalla, *Battling Team*, donde se ejecuta el motor encargado de gestionar los enfrentamientos. Esta capa es utilizada por la capa de campeonato, que se encarga de coordinar y ejecutar las batallas requeridas durante el torneo. Por encima de ambas se encuentra la capa de metajuego, *Move Set*, cuyo propósito es ejecutar múltiples épocas de campeonatos, cada una con una versión modificada del *roster*, con el fin de estudiar o mantener el equilibrio estratégico del entorno competitivo.

Para cada una de estas capas se han diseñado interfaces específicas que permiten definir el comportamiento deseado en función del nivel en el que se opera. Esto proporciona una separación clara entre las responsabilidades de cada componente del sistema.

El *framework* permite un uso flexible de todos estos elementos. Es posible ejecutar competiciones completas, simular batallas individuales o incluso extraer datos útiles para entrenar modelos o desarrollar otras soluciones. Esta flexibilidad se mantiene sin perder la filosofía modular del sistema: el *roster* tiene un papel central en el equilibrio del metajuego, aunque también puede influir en la construcción de equipos; mientras que los *Pokémon* y sus movimientos solo resultan relevantes dentro del contexto de una batalla [22].

3.1.2 Categorías

La competición VGC AI 2.0 se organiza en tres circuitos (tracks) independientes, cada uno evaluando un aspecto distinto de la inteligencia artificial aplicada a la formación de equipos y la simulación de batallas:

- **Battle Track:** En esta categoría, cada agente recibe equipos generados aleatoriamente y debe vencer a su oponente exclusivamente a través de la política de combate *Battle Policy* que implemente. El objetivo es construir agentes capaces de adaptarse a cualquier equipo dado y maximizar su ratio de victorias, sin intervenir en la construcción del propio equipo.
- **Championship Track:** Aquí los agentes son evaluados en la fase de *Team Building*. Cada participante debe elegir un equipo de seis *Pokémon* antes de que empiece la serie de combates contra los demás competidores. Tras formarse los equipos, se celebran varios enfrentamientos, cada uno consistiendo en un número fijo de batallas, y a partir de los resultados se asigna o se resta ELO a cada agente. El *ranking* final se calcula según esa puntuación ELO, y sirve para determinar el ganador del circuito.
- **Meta-Game Balance Track:** En este track, cada agente no solo construye un equipo, sino que además propone cambios en las condiciones del

torneo, como por ejemplo, calibrar estadísticas de uso de movimientos o ajustar restricciones en la composición de los equipos con el fin de equilibrar el metajuego global. Se evalúa la habilidad de medir estadísticamente qué ajustes conducen a un pool de equipos más diverso o competitivo.

3.1.3 Championship Track

Se ha optado por el *Championship Track* porque constituye el entorno más apropiado para aplicar y evaluar los enfoques basados en MCTS y LLMs. En este campeonato, cada agente sólo debe entregar un único script en Python que implemente la lógica completa de construcción de equipo y simulación de batallas. Sin embargo, para los experimentos realizados en este trabajo se emplearán los modelos de lenguaje DeepSeek-R1-Llama3.1-Distill y Llama 3.1, cuyo uso requiere conectarse a través de la herramienta externa LM Studio. De esta manera, aunque la competición en sí solamente permita la entrega de un script autónomo, en nuestro análisis los LLMs serán invocados desde LM Studio para enriquecer la toma de decisiones de construcción de equipos en tiempo real.

3.1.3.1 Parámetros utilizados

- Se fija de antemano el número total de agentes, en nuestro caso lo fijaremos a 20 agentes, siendo uno de ellos nuestro agente (MCTS o LLM) y el resto los agentes base proporcionados en el *framework*.
- Cada participante dispone de un roster común consistente en un listado de 50 Pokémon permitidos.
- El tamaño del equipo a enviar a cada ronda siempre será de 3 Pokémon.
- Se establecen parámetros de batalla (por ejemplo, número máximo de movimientos por Pokémon, número de batallas por enfrentamiento), idénticos para todos.

3.1.3.2 Fase de construcción de equipos

Antes de que dé comienzo la serie de enfrentamientos, cada agente ejecuta su *Team Build Policy*, política con reglas para creación de equipos, y devuelve un *TeamBuildCommand*, es decir, tres índices del 0 al 49 que eligieron del *roster*.

Si un agente envía un equipo inválido, como puede ser índices fuera de rango o *Pokémon* repetidos, el *framework* le asigna un equipo aleatorio legal para esa ronda.

3.1.3.3 Rondas de combate y actualizaciones de ELO

Una vez todos los agentes han enviado sus equipos, comienza un torneo consistente en *epochs*, estos son, rondas por campeonato. Cada agente se enfrenta a cada uno de los demás una cantidad predefinida de veces, en este caso serán de 10 rondas por campeonato.

En cada ronda se disputan varias batallas, manteniendo en este proyecto el caso base de 3 batallas, y así, manteniendo la regla de desempate con dos victorias en caso de empate. Esta implementación queda oculta en el motor de batalla, pero cada victoria o derrota se va acumulando.

Al terminar todas las batallas entre dos agentes A y B, se calculan las nuevas puntuaciones ELO: si A gana más batallas que B, el ELO de A sube y el de B baja.

3.1.3.4 Clasificación final

Al concluir todos los enfrentamientos, cada agente tiene un ELO resultante. Se ordena la tabla de participantes de mayor a menor ELO, y la posición en ese *ranking* determina la posición final del agente en el *Championship Track*.

El agente que figure en primer lugar en ELO es declarado ganador de este circuito.

En el VGC AI 2.0, el *Championship Track* presenta dos novedades más con respecto a la versión 1.0:

- **Cálculo de ELO revisado:** la fórmula ELO mantiene sus fundamentos clásicos, pero se han ajustado las constantes de ganancia/pérdida. En esta versión 2.0, se ha podido observar que, para la configuración utilizada en esta investigación, el rango de ELO que se puede obtener estará comprendido entre los valores 988.44 y 1411.56. Cuando se pierde todas las 10 rondas, se obtendrá 988.44 y cuando se gana todas las rondas posibles, 1411.56.
- **Nuevo flujo de decisión:** la función de *decision* de cada agente recibe ahora un contexto más completo, ya no solo el último equipo y el ELO anterior, sino también estadísticas de uso de cada *Pokémon* y movimientos, así como un registro de equipos previos), de modo que las políticas de *team building* puedan apoyarse en información histórica y no únicamente en heurísticas simples.

3.1.3.5 Resumen de configuración

La configuración utilizada en el análisis quedaría en:

- 10 *epochs*, o rondas
- 100 movimientos *n_moves* predeterminados
- 50 *Pokémon* disponibles en el *roster*
- 20 competidores, o agentes, por campeonato
- 4 movimientos disponibles por *Pokémon*
- 3 batallas por ronda

3.2 Agentes

3.2.1 Agente base

El agente proporcionado por el *framework* de VGC AI 2.0, es un agente aleatorio el cual aplica las políticas predeterminadas para esta versión. Está comprendido en *Competitor.py* y lo utilizaremos en los 19 agentes a los que nos enfrentaremos en cada campeonato.

Esta es la estructura, la cual utiliza las políticas de esta versión ya explicados, además de otros como *Team Predictor*, *Team Valuator*, *Data Aggregator* y *Balance Policy*, definidas para el control de los resultados a modo de reglamento:

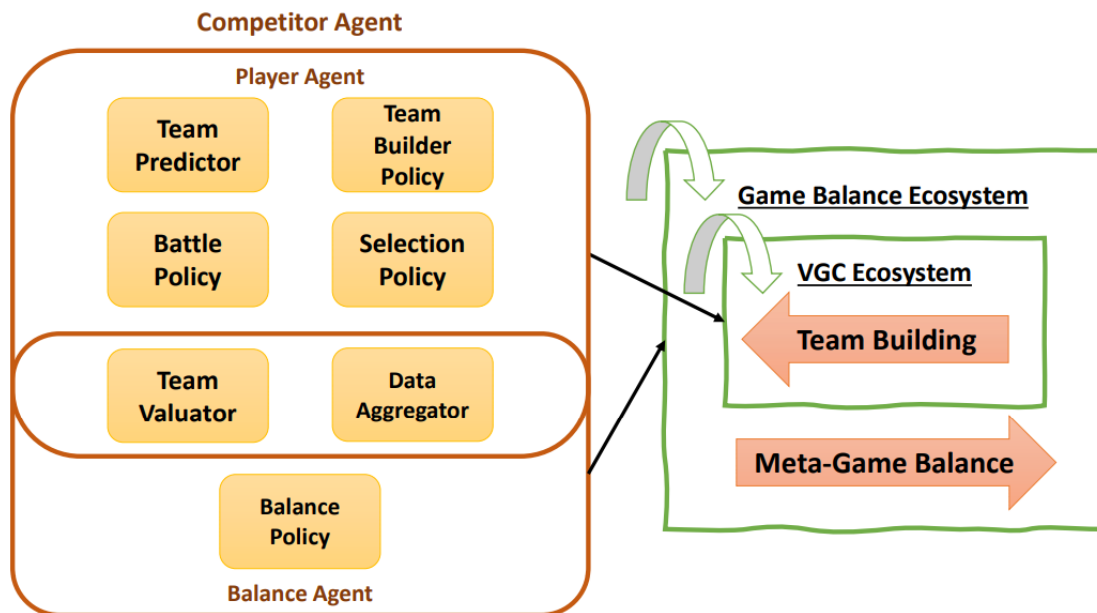


Figura 7. Flujo Agente Base

3.2.2 Agente MCTS

Dado que el *framework* actual es bastante diferente al antiguo, el agente MCTS está desarrollado en base a la nueva configuración, no pudiendo partir del agente MCTS-Rng creado en la investigación previa de *Alfedi* [23].

Aquí, la simulación se ha refinado para que el resultado de cada partida simulada dependa de la suma de las estadísticas base de los *Pokémon* en cada plantilla. Esta modificación permite que la fase de simulación tienda a favorecer las plantillas con mayor fuerza bruta. En caso de empate, se recurre al azar para romper la igualdad. Esto reduce el ruido de las simulaciones y suele acortar el número de iteraciones necesarias para distinguir buenas opciones de malas.

3.2.2.1 Esquema MCTS

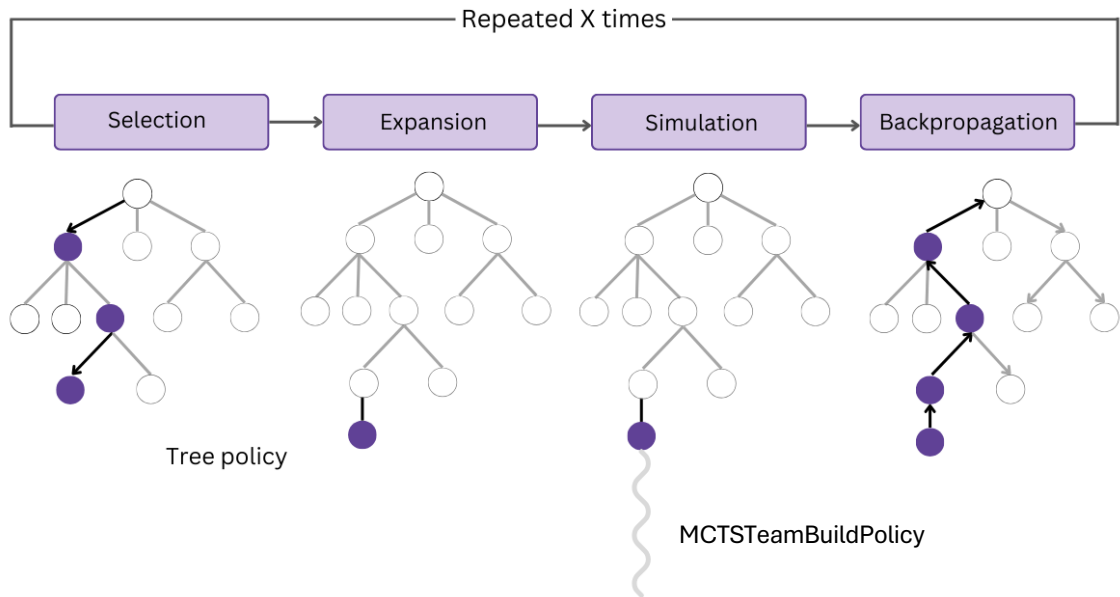


Figura 8. Esquema MCTS VGC AI 2.0

El esquema seguido es el mismo que tiene un MCTS, pero la política a seguir en la fase de simulación, es el *MCTSTeamBuildPolicy*, el cual se ha creado con nuestras propias reglas.

3.2.2.2 Selección y expansión inicial

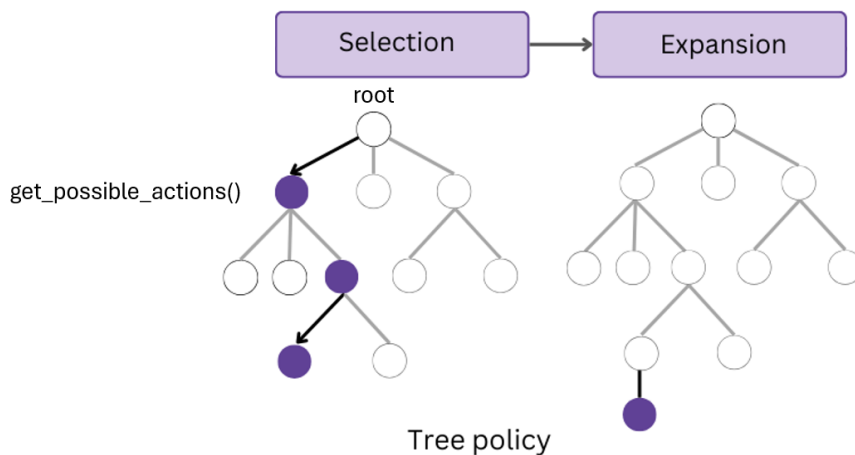


Figura 9. Flujo selección y expansión

Se crea un nodo raíz vacío, *root*, bajo el cual se expanden todos los hijos posibles en la primera capa. Cada hijo corresponde a una combinación de índices de Pokémon de longitud igual al tamaño de equipo que en este caso serán de tres.

Después, con el método *get_possible_actions()*, se consigue el número de acciones posibles que genera todas las tuplas de tres índices. De esa forma, si el *roster* tiene 50 especies, se generarán (de 50 se elige 3) nodos hijos, cada uno representando esa triple de índices.

3.2.2.3 Simulación

Una vez que el árbol de un solo nivel está construido, es decir que tenemos todos los movimientos iniciales posibles, se lanzan N simulaciones. En este caso, 100 iteraciones.

En cada iteración se selecciona uno de los hijos de la raíz según la fórmula de UCT (*Upper Confidence Bound applied to Trees*), adaptada para explorar [4].

- El valor UCT se calcula como:

$$\mathbf{UCT} = \frac{\text{wins}}{\text{visits}} + c \sqrt{\frac{\ln(\text{total_simulations})}{\text{visits}}}$$

Figura 10. Fórmula UCT

donde:

- *wins/visits* es la parte de explotación, es decir, frecuencia de victorias observadas y, el sumando de la derecha, es la parte de exploración.

El hijo con mayor valor UCT, o uno aleatoriamente escogido entre aquellos que empatan, es el nodo seleccionado para simular.

La simulación consiste en comparar ese equipo candidato con otro equipo rival escogido al azar:

1. Se calcula la fuerza de cada plantilla sumando las estadísticas base de los tres *Pokémon*.
2. Si la suma del candidato es mayor que la del oponente aleatorio, la simulación retorna 1 (victoria); si es menor, retorna 0 (derrota); y si hay empate, se elige ganador al azar.

3.2.2.4 Backpropagation

Una vez que se obtiene el resultado (*reward*) de la simulación para el nodo seleccionado, se incrementan sus contadores *node.visits* += 1 y *node.wins* += *reward*. De igual forma, el nodo raíz también incrementa *root.visits* += 1 para contabilizar el total de simulaciones.

Tras completar todas las iteraciones, se elige el hijo cuya tasa de victorias (*wins/visits*) sea la mayor. Ese hijo *best* contiene la tupla de índices *best.move* que representa el equipo elegido para la competición.

Finalmente, se construye las instrucciones requerido por el *framework*, donde cada índice que representa a un *Pokémon* del roster contiene:

- iv: valores fijos de IVs, o Valores Individuales, a 31 en cada estadística.
- ev: distribución aleatoria de puntos EV, o Valores de Esfuerzo, entre las seis estadísticas (suma total 510).
- nat: naturaleza aleatoria elegida por número entero.
- mv: lista de posiciones de movimientos (toma los primeros *max_pkm_moves*).

El resultado final es una lista de tuplas (idx, ev, ivs, nat, mv), que VGC AI interpreta como la formación completa del equipo.

3.2.3 Agente híbrido LLM

Para el agente híbrido, se utiliza el mismo script de `competidor_llm.py` con la diferencia que dependiendo que LLM queramos usar, solo se cambiará el nombre del modelo. En caso de no proporcionarse un equipo válido, el código base del MCTS empezará a utilizarse.

La función para conectarse a la API de LM Studio consta de una función `query_llm()`, donde se le pasarán un *prompt* y el nombre del modelo LLM (deepseek-r1-distill-llama-8b / meta-llama-3.1-8b-instruct). Se pueden ajustar parámetros como la temperatura empleada (balance entre el determinismo y la aleatoriedad de las respuestas), en nuestro caso de 0.7, y el máximo de *tokens* que se utilizarán.

Cuando se utiliza DeepSeek, debido a que se recopila su razonamiento, se utiliza una función auxiliar, donde se detecta el razonamiento “think” y lo elimina, filtrando así la respuesta esperada con los índices que queremos.

Otro aspecto relevante es, la implementación de la función `decision()`, en la cual buscaremos la última partida que el equipo jugó y, si ganamos, lo añadimos a la constante de equipos ganadores y si perdemos viceversa. Luego se determina si se ha ganado o perdido y el nuevo ELO obtenido. Con estos datos obtenidos, podemos añadir más contexto al LLM en la siguiente iteración, de manera que sus respuestas puedan ser más fiables.

3.2.3.1 MCTS+Llama 3.1

A pesar de su rapidez, Llama 3.1 carece del proceso de razonamiento profundo que integra DeepSeek, lo que significa que las respuestas tienden a seguir rutas más convencionales en lugar de explorar alternativas novedosas. Esto provoca que, aunque ofrezca un tiempo de respuesta muy bajo y estable, sus sugerencias de equipo carezcan en ocasiones de creatividad estratégica. Sin embargo, el tiempo que requiere lo hace idóneo para escenarios donde se prioriza velocidad sobre profundidad de análisis, y sirve como punto de partida para comparar mejoras introducidas por DeepSeek.

3.2.3.2 MCTS+DeepSeek R1 Distill Llama 3.1

DeepSeek con Llama 3.1 combina la potencia de Llama con un proceso de razonamiento interno. Esto a veces alarga su tiempo de respuesta, pero evita que el modelo pierda todas las rondas, ya que al menos gana una. Siempre analiza los equipos que le derrotaron y suele optar por jugadas menos arriesgadas.

3.3 Entorno simulaciones

Para la realización del experimento se ha utilizado el programa LM Studio en su versión 0.3.15 (Build 11) en una máquina con 32GB de RAM, 14 núcleos, un procesador Intel i7 13700H de 13^a Generación y una tarjeta gráfica NVIDIA GeForce RTX 4070 8GB GDDR6.

Para los modelos LLM, se han ejecutado con los mismos ajustes: una temperatura de 0.7 y una ventana de contexto de un máximo de 8000 *tokens*.

Dependiendo del agente, cada *Championship* tiene una duración diferente:

- meta-llama-3.1-8b-instruct: media de 10 minutos
- deepseek-r1-distill-llama-8b: media de 17 minutos

Todo el código de los agentes creados se encuentra público en GitHub [24].

4 Resultados

Para calcular el tamaño de la muestra necesario, con un porcentaje de victorias, nivel de confianza y margen de error, se utiliza una fórmula estadística. El tamaño de la muestra (n) se calcula con:

$$n = (Z^2 * p * (1-p)) / E^2$$

- **Z:** Valor crítico asociado al nivel de confianza deseado (con 1.96 para 95% de confianza).
- **p:** Estimación del winrate como decimal.
- **E:** Margen de error deseado (0.05 para 5%).

Como queremos saber el tamaño máximo de muestra, se puede utilizar un valor de $p = 0.5$. Si queremos un nivel de confianza del 95% con una tasa de error del 5%, siguiendo la fórmula obtenemos que:

$$n = (1.96^2 * 0.5 * (1-0.5)) / 0.05^2 = \mathbf{384.16}$$

Por lo que, necesitamos 384 resultados de cada campeonato para poder analizar adecuadamente los resultados de los agentes.

Para un nivel de confianza del 95% con una tasa de error del 10%:

$$n = (1.96^2 * 0.5 * (1-0.5)) / 0.1^2 = \mathbf{96.04}$$

Así, se intentará llegar al mayor número posible de muestras, teniendo en cuenta que se está asumiendo una tasa de victorias del 50% y que el tiempo que requiere cada agente difiere entre ellos.

4.1 Resultados MCTS

Para el agente MCTS, se ha obtenido una tasa de victoria del 10,07% tras ejecutar 139 campeonatos. Dado que el tamaño de muestra es inferior al requerido para garantizar un margen de error del 5% con un 95% de confianza, se ha calculado el intervalo de confianza real para esta muestra.

El intervalo de confianza del 95% para la tasa de victoria es de entre el 5,03% y 15,11%, lo que implica un margen de error aproximado de $\pm 5,04\%$. Este resultado indica que, con un nivel de confianza del 95%, la tasa real de victoria del agente se encuentra dentro de ese rango.

Aunque el margen de error es ligeramente superior al umbral deseado, la muestra sigue siendo razonablemente representativa. Sin embargo, una mayor cantidad de campeonatos podría aportar una estadística más precisa.

Competidor	MCTS
Victorias (%)	10,07
Mejor ELO	1411,56
Peor ELO	988,44
\bar{x} ELO	1197,37
\bar{x} ELO al Ganador	139,93
σ ELO	97,83
\bar{x} Posición	10,94
σ Posición	6,77

Figura 11. Resultados Agente MCTS

4.2 Resultados Llama 3.1

Se ha obtenido una muestra de 327 campeonatos, en los que el agente ha alcanzado una tasa de victoria del 25,38%. A partir de estos datos, se ha calculado el intervalo de confianza del 95%, que sitúa la tasa real de victoria entre 20,66% y 30,10%. Esto implica un margen de error aproximado de $\pm 4,72\%$, lo que permite considerar esta muestra como fiable, estadísticamente, para la evaluación del rendimiento del agente con Llama 3.1.

Dado que el margen de error se encuentra por debajo del umbral del 5%, se puede afirmar con un alto grado de confianza que los resultados obtenidos reflejan con precisión el comportamiento esperado del modelo dentro de nuestro entorno competitivo.

Competidor	MCTS+Llama3.1
Victorias (%)	25,38
Mejor ELO	1411,56
Peor ELO	988,44
\bar{x} ELO	1230,58
\bar{x} ELO al Ganador	114,2
σ ELO	115,84
\bar{x} Posición	8,82
σ Posición	7,24

Figura 12. Resultados Agente MCTS+Llama3.1

4.3 Resultados DeepSeek R1 Distill Llama 3.1

Se ha obtenido una muestra de 222 campeonatos, en los que el agente ha alcanzado una tasa de victoria del 39,19%. A partir de estos datos, se ha calculado el intervalo de confianza del 95 %, que sitúa la tasa real de victoria entre 32,78% y 45,60%. Esto implica un margen de error aproximado de $\pm 6,41$ %.

Aunque el número de muestras es considerable, el margen de error supera ligeramente el umbral del 5 %, por lo que los resultados, pueden variar con más cantidad de muestras. Aun así, el rendimiento del modelo destaca claramente por encima del resto, lo que indica que el modelo ha mostrado un buen rendimiento en comparación al agente base con MCTS puro y al agente con Llama 3.1.

Competidor	MCTS+DeepSeek R1
Victorias (%)	39,19
Mejor ELO	1411,56
Peor ELO	1018,44
\bar{x} ELO	1268,61
\bar{x} ELO al Ganador	83,43
σ ELO	109,95
\bar{x} Posición	6,67
σ Posición	6,79

Figura 13. Resultados Agente MCTS+DeepSeek R1

4.4 Análisis final

Además de analizar los resultados obtenidos por los tres agentes creados, es interesante ver la comparación respecto a los agentes del año pasado. Los más relevantes a comparar son MCTS-Rng (agente MCTS puro), MCTSo-L327b (versión optimista de Llama 3.1) y MCTSp-L327b (versión pesimista de Llama 3.1), todos ellos elaborados en el VGC AI 1.0 [3].

Sin embargo, y debido a los cambios en la estructura del *framework* y reglas en VGC AI 2.0, los valores de ELOs obtenidos el año pasado, no son comparables dado que salen del intervalo de ELO [988,44; 1411,56] que aplica para la convocatoria de 2025. Se han tomado en cuenta la tasa de victorias y la posición media obtenidas debido a que para ambos casos el número de competidores es el mismo.

4.4.1 Comparación de resultados

Como se puede apreciar en la siguiente tabla, los resultados de los agentes han aumentado considerablemente, siendo la versión MCTS pura de la anterior edición, un peor resultado. El agente con DeepSeek, no obstante, tiene una tasa de victorias del 39,19%, representando una mejora de rendimiento frente al resto de agentes.

En cuanto a la posición, DeepSeek obtiene una media de obtención del 6,79, casi entrando en el top 5 del total de 20 competidores.

Competidor	Victorias (%) ▼	Mejor ELO	Peor ELO	\bar{x} ELO	\bar{x} ELO al Ganador	σ ELO	\bar{x} Posición	σ Posición
MCTS+DeepSeek R1	39,19	1411,56	1018,44	1268,61	83,43	109,95	6,67	6,79
MCTS+Llama3.1	25,38	1411,56	988,44	1230,58	114,2	115,84	8,82	7,24
*MCTSo-L327b	11,34						9,77	6,59
MCTS	10,07	1411,56	988,44	1197,37	139,93	97,83	10,94	6,77
*MCTSp-L327b	8,25						10,62	6,24
*MCTS-Rng	8,25						10,74	5,81

Figura 14. Comparación resultados agentes

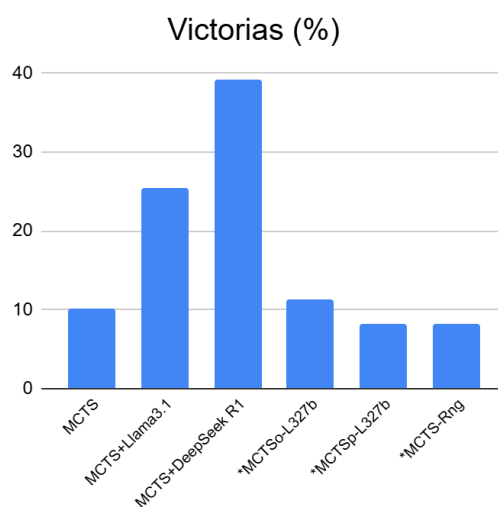


Figura 15. Comparación tasa de victorias

4.4.2 Discusión de resultados

Después de observar los resultados, queda claro la gran eficiencia de DeepSeek. Además, a pesar de haber mejorado el rendimiento de Llama 3.1 con respecto a ambos agentes de Llama 3.1 de la competición anterior, no llega a la gran tasa de éxito de su versión destilada con DeepSeek-R1.

Se puede apreciar también que, el aumento de rendimiento del agente Llama 3.1 de este año, se debe a la aplicación de las tres técnicas: *text prompting*, *fine-tuning* y RAG. Se ha podido comprobar que, en efecto, la unión de estos tres

métodos permite mejorar la calidad de las respuestas generadas, reducir los errores en la toma de decisiones y adaptar el modelo de forma más eficaz al contexto dinámico del entorno competitivo [25].

A pesar del aumento de rendimiento con DeepSeek, durante las pruebas se observó que, al intentar no correr riesgos, el modelo podía llegar a “sobrepensar”, es decir, analizar un resultado y retractarse al estimar una posible derrota, para luego finalmente optar por el primer resultado escogido. Esto produjo que pudiera llegar a tardar demasiado. Para corregirlo, se ajustó el *prompt* inicial de modo que, si insistía en volver a usar un equipo que ganó, pero luego pierde más de una vez, deje de reutilizar esa formación después de dos derrotas. Con esta nueva regla, se ceñía a lo ordenado y no divagaba por su cuenta.

5 Conclusiones

Tras realizar el análisis de los distintos enfoques, queda claro que los modelos de lenguaje pueden aportar un valor significativo en entornos de decisión estratégica, siempre que se integren de forma controlada y bajo supervisión. A lo largo del proyecto, se ha demostrado que técnicas como *text prompting*, *fine-tuning* y RAG permiten mejorar la calidad de las decisiones generadas por los modelos, ofreciendo un apoyo real en tareas complejas como la construcción de equipos, en este caso, en VGC AI 2.0.

Sin embargo, también se ha observado que, al introducir mecanismos de razonamiento más sofisticados, como los de DeepSeek, los modelos tienden a comportarse de forma más parecida al pensamiento humano. Esto puede ser útil, pero también conlleva riesgos, como el sobreanálisis o la pérdida de consistencia en las decisiones. Durante las pruebas, fue necesario intervenir y ajustar el comportamiento del modelo mediante cambios en los *prompts*, lo que puso de manifiesto la importancia de que estas herramientas sigan siendo supervisadas. Hoy en día, los LLMs deben considerarse como un complemento al criterio humano, y no como entidades autónomas en las que confiar plenamente.

Además de mejorar el rendimiento en un entorno competitivo, este trabajo ha servido para reflexionar sobre el uso práctico de los LLMs como herramientas de asistencia. Supervisar activamente el razonamiento de DeepSeek ha resultado una mecánica útil, no solo para ajustar su comportamiento, sino también para entender mejor cómo aprovechar su potencial sin perder el control sobre las decisiones finales. En el futuro, seguirá siendo clave encontrar el equilibrio entre delegar tareas complejas y mantener la supervisión humana en el uso de modelos avanzados como estos.

De cara a la próxima edición de VGC AI, la organización plantea la inclusión de nuevas reglas de torneo generadas automáticamente, así como la introducción del tiempo de decisión como criterio de desempate. Ante este nuevo escenario, será importante explorar mejoras como la optimización del razonamiento mediante modelos aún más ligeros, la incorporación de restricciones temporales en los *prompts* o el uso de técnicas que eviten sobreanálisis. Estas estrategias podrían contribuir a mantener un rendimiento competitivo sin comprometer la eficiencia, alineándose mejor con los objetivos futuros del entorno VGC AI.

Por último, sería interesante estudiar el rendimiento de la nueva versión de DeepSeek-R1-0528 publicada recientemente, y analizar si los cambios introducidos afectan a la toma de decisiones en contextos similares. Además, dado que la organización de DeepSeek AI ha insinuado el desarrollo de una futura versión, posiblemente bajo el nombre de DeepSeek-R2, su análisis podría representar una línea de trabajo prometedora dado que su potencial es superior a DeepSeek-R1 y equiparable a OpenAI-o3 [26]. Esto supone, una vez más, un recordatorio de que la inteligencia artificial, y en particular los modelos de lenguaje, están evolucionando a un ritmo exponencial. Por ello, estar al día con los avances en modelos de lenguaje no es solo recomendable, sino imprescindible para asegurar un uso eficaz, responsable y alineado con el ritmo de evolución tecnológica.

6 Análisis de Impacto

La realización de este Trabajo de Fin de Grado ha tenido un impacto personal relevante, ya que ha requerido investigar por cuenta propia conceptos que, además de ser técnicamente exigentes, están muy presentes en el desarrollo actual de la inteligencia artificial. Esto ha permitido aprender herramientas y metodologías que no solo han sido útiles para el proyecto, sino que también resultan aplicables en muchos otros contextos. Además, el proceso de comparar enfoques, ajustar el comportamiento del agente y analizar resultados ha sido enriquecedor tanto a nivel técnico como en lo que respecta a la forma de enfrentarse a un proyecto complejo de principio a fin.

En cuanto a su proyección práctica, muchas de las técnicas empleadas, como el uso de modelos destilados, el ajuste de *prompts* o el uso de esquemas tipo RAG, podrían trasladarse a entornos reales donde se necesite automatizar decisiones o generar contenido contextualizado. Este tipo de soluciones resulta especialmente útil en sectores que manejan grandes volúmenes de datos o que requieren respuestas adaptadas en tiempo real, como pueden ser entornos empresariales o herramientas de soporte.

Desde una perspectiva más general, este trabajo también permite reflexionar sobre cómo deberían utilizarse los modelos de lenguaje. Aunque han demostrado ser muy útiles como herramientas de apoyo, sigue siendo importante mantener una supervisión activa, sobre todo cuando se les asignan tareas con cierto nivel de interpretación o razonamiento. En ese sentido, se ha comprobado que los modelos pueden llegar a dar respuestas inconsistentes si no se delimitan bien sus instrucciones, lo que refuerza la idea de que estos sistemas deben acompañar el criterio humano, no sustituirlo.

Además, el uso de modelos ligeros como DeepSeek-R1-distill ayuda a reducir el consumo computacional frente a otras alternativas más pesadas, lo que se alinea con una visión más sostenible del desarrollo de inteligencia artificial. Este aspecto conecta con algunos de los Objetivos de Desarrollo Sostenible de la Agenda 2030, como el ODS 12 (producción y consumo responsables), el ODS 9 (industria, innovación e infraestructura), o el ODS 4 (educación de calidad), al tratarse de un proyecto que promueve la innovación responsable dentro del marco de una formación académica avanzada.

Por último, dado que el organizador de la competición ha planteado para futuras ediciones nuevas reglas de balance automático y la incorporación del tiempo de decisión como criterio de desempate, este trabajo también sienta una base útil para futuras mejoras. Entre ellas, sería interesante explorar técnicas que permitan reducir los tiempos de inferencia, como el uso de respuestas condicionadas a confianza, o limitar la longitud de razonamiento sin afectar a la calidad final. Todo ello permitirá avanzar hacia agentes más rápidos, eficientes y ajustados a los nuevos retos del entorno competitivo.

7 Bibliografía

- [1] IEEE 2025 Conference Of Games, «VGC AI Competition 2025 Edition,» 2025. [En línea]. Available: <https://cog2025.inesc-id.pt/vgc-ai-competition/>.
- [2] S. Reis, L. P. Reis y N. Lau, «VGC AI 1.0 Framework,» [En línea]. Available: https://gitlab.com/DracoStriker/pokemon-vgc-engine/-/tree/legacy_vgc_ai_1.
- [3] Á. F. Díez, "Explorando la integración entre LLMs y MCTS en los problemas de formación de equipos a través de la competición VGC AI", Universidad Politécnica de Madrid, 2024.
- [4] Browne, Cameron B, et al, "A survey of Monte Carlo Tree Search Methods", vol. 4, IEEE Transactions on Computational Intelligence and AI in Games: IEEE, 2012, p. 43.
- [5] Meta AI, «Introducing Llama 3.1: Our most capable models to date,» 2024. [En línea]. Available: <https://ai.meta.com/blog/meta-llama-3-1/>.
- [6] DeepSeek AI, «DeepSeek LLaMA: An Efficient Distilled Language Model,» 2024. [En línea]. Available: <https://deepseek.com/llama-distilled>.
- [7] S. Reis, L. P. Reis y N. Lau, «VGC AI 2.0 Framework,» 2025. [En línea]. Available: <https://gitlab.com/DracoStriker/pokemon-vgc-engine>.
- [8] IBM, «What is retrieval-augmented generation?,» 2023. [En línea]. Available: <https://research.ibm.com/blog/retrieval-augmented-generation-RAG>.
- [9] S. Olesker-Taylor y L. Zanetti, «An Analysis of Elo Rating Systems via Markov Chains,» 2023. [En línea]. Available: <https://arxiv.org/pdf/2406.05869>.
- [10] IBM, «¿Qué son los grandes modelos de lenguaje (LLM)?,» 2023. [En línea]. Available: <https://www.ibm.com/es-es/think/topics/large-language-models>.
- [11] OpenAI, «GPT-4 | OpenAI,» 2023. [En línea]. Available: <https://openai.com/es-ES/index/gpt-4/>.
- [12] Anthropic, «Meet Claude \ Anthropic,» 2025. [En línea]. Available: <https://www.anthropic.com/claude>.
- [13] Google DeepMind, «Gemini 1: our largest and most capable model,» 2023. [En línea]. Available: <https://deepmind.google/technologies/gemini/>.
- [14] Alibaba Cloud, «QwenLM/Qwen3,» 2025. [En línea]. Available: <https://github.com/QwenLM/Qwen3>.
- [15] DeepSeek AI, «DeepSeek-R1,» 2025. [En línea]. Available: <https://github.com/deepseek-ai/DeepSeek-R1>.
- [16] Vaswani et al., «Attention Is All You Need,» 2017. [En línea]. Available: <https://arxiv.org/pdf/1706.03762>.

- [17] Jean Nyandwi, «The Transformer Blueprint,» 2023. [En línea]. Available: <https://deepreversion.github.io/posts/001-transformer>.
- [18] Meta, «Llama 3.1 Instruct 8B: Intelligence, Performance & Price Analysis,» 2024. [En línea]. Available: <https://artificialanalysis.ai/models/llama-3-1-instruct-8b>.
- [19] OpenAI, «Text Generation and Prompting,» 2024. [En línea]. Available: <https://platform.openai.com/docs/guides/text?api-mode=responses>.
- [20] Founders & Microsoft, «IA Responsable y Prompt Engineering,» 2025. [En línea]. Available: <https://founderz.com/es/programa/curso-generacion-ia-comunidad-madrid/>.
- [21] Y. Burowski, «LM Studio,» 2024. [En línea]. Available: <https://lmstudio.ai/>.
- [22] S. Reis, L. P. Reis y N. Lau, «VGC AI 2.0 Framework Architecture,» 2025. [En línea]. Available: <https://gitlab.com/DracoStriker/pokemon-vgc-engine/-/wikis/Documentation/Framework-Architecture>.
- [23] Á. F. Díez, «PokeHit/Alfedí,» 2024. [En línea]. Available: <https://github.com/Alfedí/PokeHit/blob/main/mcts.py>.
- [24] G. A. Avilés, «geodrns/pokemon-vgc-ai,» 2025. [En línea]. Available: <https://github.com/geodrns/pokemon-vgc-ai>.
- [25] A. Barrow, M. El-Assady, B. Bach et al., «A Systematic Evaluation of LLM Strategies for Mental Health Text,» 2025. [En línea]. Available: <https://arxiv.org/pdf/2503.24307>.

8 Anexos

8.1 Prompt fine tuning en LM Studio

You are a team building machine for a simplified version of Pokémon.

Before each call you will receive:

- Roster: the full list of allowed Pokémon templates, each with Type, Max_HP and Moves.
- Previous Team: the three indices you used last time.
- Last Result: an English summary—either “This is your first battle; no prior result.” or “Your last team won. Your new Elo is XXXX.” or “Your last team lost. Your new Elo is XXXX.”

Using only that info, choose a new 3 Pokémon team from the Roster that will maximize your chances in the next battle. If a team combination lost two or more battles, don't continue to choose it, but if it's winning more than losing, you can try again.

****Instructions:****

- Return exactly three indices (no more, no less), comma-separated.
- Use the format `---i1, i2, i3---` .
- Do ****not**** include any extra text, explanation, punctuation or Markdown.
- Only simple text within the triple-dash markers.

8.2 Prompt en script

```
def build_prompt(self,
                 roster: list,
                 current_team: list[int],
                 match_outcome: str,
                 new_elo: float | None,
                 winning_indices: list[list[int]],
                 losing_indices: list[list[int]]) -> str:
    # Cabecera de resultado anterior
    if "first battle" in match_outcome:
        header = "This is your first battle; no prior result.\n"
    elif "unknown" in match_outcome:
        header = "Your last result could not be determined.\n"
    else:
        header = f"Your last team {match_outcome}."
        if new_elo is not None:
            header += f" Your new Elo is {new_elo:.2f}."
        header += "\n"
```

```

# Lista de indices ganadores previos
if winning_indices:
    wins_lines = "\n".join(
        "[" + ", ".join(str(i) for i in win_list) + "]"
        for win_list in winning_indices
    )
    wins_line = f"Winning team indexes in past matches:\n{wins_lines}\n"
else:
    wins_line = "No past winning-team data available\n"

# Lista de idices perdedores previos
if losing_indices:
    loses_lines = "\n".join(
        "[" + ", ".join(str(i) for i in lose_list) + "]"
        for lose_list in losing_indices
    )
    loses_line = f"Losing team indexes in past matches:\n{loses_lines}\n"
else:
    loses_line = "No past losing-team data available\n"

# roster formateado
roster_lines = "\n".join(
    f" {i}: {fmt_pokemon_species(sp)}"
    for i, sp in enumerate(roster)
)

# Si es la primera batalla no incluimos los equipos previos ya q no hay
if "first battle" in match_outcome:
    prev_line = ""
else:
    prev_team_str = ", ".join(str(i) for i in current_team)
    prev_line = f"Previous team indices: [{prev_team_str}]\n"

return (
    header
    + wins_line
    + loses_line
    + "Now, given the following:\n"
    + "Roster:\n"
    + roster_lines + "\n"
    + prev_line
    + "Please return exactly the new team indices following this format: ---id1, id2,
id3---" #repetimos el formato para asegurarnos q se respondera asi
)

```

8.3 Ejemplo log DeepSeek

```
2025-06-02 09:04:06,601 INFO: Enviando prompt a LM Studio (deepseek-r1-distill-llama-8b):
Your last team won. Your new Elo is 1343.78.
Winning team indexes in past matches:
[5, 7, 17]
[5, 7, 14]
[5, 7, 14]
[5, 7, 14]
[5, 14, 17]
[5, 7, 17]
[5, 7, 17]
[5, 14, 17]
Losing team indexes in past matches:
[5, 7, 14]
Now, given the following:
Roster:
  0: PkmTemplate(Type=BUG, Max_HP=124, Moves=[GHOST, Max PP 10, OTHER, Priority 1, Self Switch, FIRE, Accuracy
0.83, Max PP 11, OTHER, Priority 1, Protect, ELECTRIC, Power 117, Accuracy 0.86, Max PP 11, PHYSICAL,
Probability 0.16, STEALTH_ROCK, DARK, Power 137, Accuracy 0.86, Max PP 9, SPECIAL, Probability 0.02,
Trickroom])
  1: PkmTemplate(Type=BUG, Max_HP=108, Moves=[ELECTRIC, Accuracy 0.76, Max PP 10, OTHER, Priority 1, Disable,
FAIRY, Power 104, Accuracy 0.97, Max PP 8, PHYSICAL, Probability 0.26, SUN, GRASS, Accuracy 0.80, Max PP 6,
OTHER, Protect, DARK, Accuracy 0.86, Max PP 10, OTHER, Target Boosts (0, -1, 0, -1, 0, 0, 0, 0)])
...
[Resto de contenido del roster]
...
  49: PkmTemplate(Type=WATER, Max_HP=122, Moves=[FAIRY, Power 52, Max PP 12, SPECIAL, Probability 0.63, Heal
0.34, GHOST, Max PP 9, OTHER, Priority 1, Ignore Evasion, ICE, Power 51, Accuracy 0.90, Max PP 9, PHYSICAL,
Probability 0.64, Protect, NORMAL, Power 48, Max PP 11, PHYSICAL, Probability 0.66, Force Switch])
Previous team indices: [5, 14, 17]
Please return exactly the new team indices following this format: ---id1, id2, id3---
2025-06-02 09:04:06,601 DEBUG: Starting new HTTP connection (1): 192.168.1.131:1234
2025-06-02 09:04:55,126 DEBUG: http://192.168.1.131:1234 "POST /v1/chat/completions HTTP/1.1" 200 6278
2025-06-02 09:04:55,126 INFO: deepseek-r1-distill-llama-8b respondio: <think>
Alright, so I need to figure out the best new Pokémon team from the given roster based on the last battle's
outcome. Let me start by understanding the problem step by step.

First, the user provided some context about previous battles. The last result was a win with an Elo of
1343.78. The winning teams in past matches were [5,7,17], [5,7,14], and so on. The losing team indices include
[5,7,14]. So, my task is to choose a new team that can continue the winning streak.

The user also mentioned that if a team combination lost two or more battles, we shouldn't continue using it.
But since the last result was a win, I need to find a team that's more likely to win again.

Looking at the Roster, there are 49 Pokémon templates with their types, Max_HP, and Moves. Each index
corresponds to a unique Pokémon. The previous team used indices [5,14,17], which are PkmTemplates 5, 14, and
17. Let me check what those are:
```

- Index 5: Type GHOST/ELECTRIC
- Index 14: Type DARK/GHOST
- Index 17: Type STEEL/ROCK

I need to evaluate if these types are effective together or if switching them could lead to better performance.

First, let's consider the previous team's composition. Ghost and Electric types usually have good coverage against various Pokémon, especially since Ghost resists many physical attacks and Electric covers Ground and Rock moves. The addition of Steel in index 17 adds more defense and resistances, which is always beneficial.

Now, considering the last battle was won, I might want to keep a similar composition but maybe tweak one or two members for better versatility or higher effectiveness.

Looking at the losing team indices, [5,7,14], that's another combination. So, if the current team includes 5 and 14, perhaps adding 17 is better because it adds Steel which can help against Steel-type Pokémon, which might have been problematic before.

But wait, let me think about each Pokémon's role in the team. Ghost types are great for support with moves like Protect and Stealth Rock. Electric can deal good damage and set up hazards. Steel adds more defense but also can contribute through moves like Reflect or Sand Stream (if available).

I should check what moves these Pokémon have that could synergize well. For example, if one Pokémon has a move that boosts the team's effectiveness, that would be useful.

Looking at index 5: Moves include Force Switch and Protect. These are solid support moves.

Index 14: Moves like Target Boosts which can enhance allies' attacks or defenses.

Index 17: Moves include PSYCHIC_TERRAIN, which could boost the power of Psychic-type Pokémon, but since it's a Steel/Rock type, maybe it's used to set up terrain.

Now, considering the losing team [5,7,14], that would be GHOST/ELECTRIC and DARK/GHOST. Maybe the combination was too slow or lacked coverage against certain types. Perhaps replacing one member with a more versatile Pokémon could help.

Alternatively, perhaps adding a Psychic type could benefit from the Psychi terrain if index 17's move is used. But since index 14 is Dark/Ghost, which already resists many attacks, maybe it's better to keep that for defensive purposes.

I also need to consider the types' effectiveness against each other. For example, Electric vs. Ground or Rock would super effective, so having both Electric and Rock types could be good.

Wait, in the current team, index 17 is Steel/Rock, which can cover Rock-type resistances. Maybe adding a Rock type Pokémon would help more. But let me check if any other indices have better Rock coverage with more powerful moves.

Looking through the roster, I see that some Rock types have strong moves like Stone Edge or Rock Slide. For instance, index 15 is Rock with Power 97 and Max PP 9. Maybe replacing index 17 (Steel/Rock) with a pure Rock type could be beneficial if it has better offensive moves.

But without knowing the exact effectiveness of each move, it's hard to say. Alternatively, perhaps keeping the current team is still the best because they've been winning. But since the user mentioned that if a combination lost two or more times, we shouldn't continue using it, but in this case, the last result was a win.

I should also consider the types' defensive strengths. Ghost and Electric have good resistances, and adding Steel/Rock adds more layers of defense. This makes the team more resilient against various attacks, which is good for maintaining a winning streak.

Another consideration is the team's movesets. If the current team has effective moves like Protect, Stealth Rock, and Reflect, they can set up hazards effectively, making it harder for opponents to attack. That's a strong strategy that contributed to their previous success.

I also need to look at the user's instruction: choose three indices from the roster that maximize chances of winning. So, I should focus on combinations that are effective in battles based on type coverage and move synergy.

Perhaps trying a different combination could lead to better performance. For example, using a Fire-type Pokémon for offensive boost or adding a Psychic for terrain manipulation. But without knowing the specific opponent types they'll face next, it's tricky.

Alternatively, maybe the current team is already optimal because of their defensive moves and resistances. It's possible that changing one member could introduce vulnerabilities if not done carefully.

In conclusion, considering the previous success with indices 5,14,17, which are Ghost/Electric, Dark/Ghost, and Steel/Rock, it seems like a solid team. Unless there's a specific reason to replace one for better synergy or coverage, sticking with this combination might be the best choice.

</think>

---5, 14, 17---

2025-06-02 09:04:55,126 INFO: Contenido limpio: ---5, 14, 17---

2025-06-02 09:04:55,126 INFO: Extracción de índices: [5, 14, 17]

2025-06-02 09:04:55,126 INFO: Final team indices: [5, 14, 17]