



UNIVERSITAT DE
BARCELONA

Treball final de grau

GRAU D'ENGINYERIA
INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Introducció als agents d'IA basats
en LLMs: Disseny d'un sistema
multiagent per a la visualització i
consulta de dades de Fórmula 1

Autor: Carlos Bellanco

Director: Dr. Santi Seguí
Realitzat a: Departament de
Matemàtiques i Informàtica

Barcelona, 10 de juny de 2025

Abstract

The development of artificial intelligence is currently one of the most relevant topics in today's world. This is due, in part, to the emergence of Large Language Models (LLMs) and their applications, such as intelligent agents. Based on these models, they are capable of processing natural language to reason and act in their environment.

Within this context, this project aims to explore the potential of these applications by developing an agent specialized in Formula 1. This agent will have the capability to access and visualize data from different races, thereby supporting enthusiasts in their analysis and understanding of the sport.

The results obtained in this project highlight the strong capabilities of these agents to interact with and adapt to their environment. By using Code Agents, it has been possible to implement a multi-agent system that, based on a user query, is capable of creating graphs from telemetry data, timing, results, etc., and performing an analysis of them.

Resum

El desenvolupament de la intel·ligència artificial és, actualment, un dels temes de major rellevància en el món en el que vivim. Això es degut, en part, a l'aparició dels *Large Language Models* (LLMs) i les seves aplicacions com, per exemple, els agents intel·ligents. Basant-se en aquests models, són capaços de processar llenguatge natural per raonar i actuar en el seu entorn.

En aquest context, aquest projecte té l'objectiu d'explorar el potencial d'aquestes aplicacions mitjançant el desenvolupament d'un agent especialitzat en Fórmula 1. Aquest, disposarà de la capacitat de consultar i visualitzar dades de les diferents carreres per tal de facilitar el seu estudi als entusiastes d'aquest esport.

Els resultats obtinguts en aquest projecte reflecteixen les altes capacitats que tenen aquests agents d'interactuar i adaptar-se al seu entorn. Mitjançant l'ús de *Code Agents* s'ha pogut implementar un sistema multiagent que, basant-se en una consulta d'un usuari, és capaç de crear gràfiques a partir de dades de telemetria, cronometratge, resultats, etc., i de realitzar una anàlisi d'aquestes.

Resumen

El desarrollo de la inteligencia artificial es, actualmente, uno de los temas de mayor relevancia en el mundo en el que vivimos. Esto se debe, en parte, a la aparición de los *Large Language Models* (LLMs) y sus aplicaciones como, por ejemplo, los agentes inteligentes. Basándose en estos modelos, son capaces de procesar lenguaje natural para razonar y actuar en su entorno.

En este contexto, este proyecto tiene como objetivo explorar el potencial de estas aplicaciones mediante el desarrollo de un agente especializado en Fórmula 1. Éste dispondrá de la capacidad de consultar y visualizar datos de las diferentes carreras para facilitar su estudio a los entusiastas de este deporte.

Los resultados obtenidos en este proyecto reflejan las altas capacidades que tienen estos agentes de interactuar y adaptarse a su entorno. Mediante el uso de *Code Agents* se ha podido implementar un sistema multiagente que, basándose en una consulta de un usuario, es capaz de crear gráficos a partir de datos de telemetría, cronometraje, resultados, etc., y de realizar un análisis de estos.

Agraïments

Primerament, vull agrair al meu tutor Santi Seguí per la seva ajuda i per fer-me la proposta inicial d'investigar sobre els agents d'intel·ligència artificial. Gràcies a la realització d'aquest projecte, he pogut descobrir el que realment m'agrada.

Vull agrair també a la meva família i als meus amics per ajudar-me a superar una etapa difícil de la meva vida i demostrar-me el seu suport incondicional. Gràcies a ells ha sigut possible la realització d'aquest treball.

Índex

1	Introducció	1
1.1	Context i motivació	1
1.2	Objectius	1
1.2.1	Objectius relatius a l'aprenentatge	2
1.2.2	Objectius relatius al desenvolupament	2
1.3	Planificació	3
1.4	Estructura de la memòria	4
2	Marc teòric	6
2.1	Introducció als Large Language Models (LLMs)	6
2.1.1	Definició	6
2.1.2	Funcionament	7
2.1.3	Tokenització i generació del següent token	8
2.2	Agents d'intel·ligència artificial basats en LLMs	9
2.2.1	Definició	9
2.2.2	Integració amb els LLMs	11
2.2.3	Funcionament	12
2.2.4	Com crear bons agents?	16
2.3	Sistemes multiagent	18
2.3.1	Definició	18
2.3.2	Arquitectures	19
2.4	Integració amb sistemes RAG	20
2.4.1	Funcionament	20
2.4.2	Agentic RAG	21
3	Anàlisi dels recursos actuals	23
3.1	Requeriments	23
3.2	Frameworks per a la creació d'agents	24
3.3	Obtenció de dades de la Fórmula 1	25
4	Disseny i implementació del sistema multiagent	27
4.1	Disseny de l'arquitectura	27
4.2	Anàlisi de les llibreries utilitzades	28
4.2.1	Smolagents	28

4.2.2	FastF1	30
4.2.3	LangChain	31
4.2.4	Tractament i visualització de les dades	31
4.3	Disseny de les eines	31
4.3.1	Data Agent	32
4.3.2	Retrieval Agent	33
4.3.3	Web Search Agent	34
4.3.4	Graph Agent	34
4.4	Tria del LLM	38
4.5	Problemes trobats i canvi d'arquitectura	38
4.6	Elecció del system prompt	39
4.7	Connexió de l'agent amb una UI	40
5	Resultats	42
5.1	Verificació dels requeriments	42
5.2	Avaluació del funcionament i exemples d'ús	47
5.2.1	Anàlisi del rendiment	48
5.2.2	Dificultats trobades	50
6	Conclusions i treball futur	51
A	Annex	56

1 Introducció

1.1 Context i motivació

En el món actual en el qual vivim, és ben sabuda la importància de les dades i com, cada segon que passa, s'estan generant al nostre voltant milions de dades de moltíssimes fonts diferents, ja siguin sensors de temperatura, interaccions que fem a les xarxes socials o el simple fet de sortir a caminar pel carrer amb el mòbil a la butxaca.

Només al 2024, el volum de dades global va ser d'uns 149 zettabytes i s'estima que al 2025 aquest augmenti fins els 181 zettabytes. Segons uns estudis recents, s'estima que el 90% de les dades globals han sigut generades en els últims dos anys [1].

La Fórmula 1 no és cap excepció, de fet, es podria dir que és un dels esports més basat en dades de la història. Un cotxe de Fórmula 1 és com un ordinador sobre rodes, té uns 300 sensors que generen aproximadament 1.1 milions de dades de telemetria per segon. Això equival, aproximadament, a 160 terabytes de dades generades en un cap de setmana [2]. Aquestes dades obtingudes són analitzades en temps real (o en qualsevol altre moment) pels equips per prendre decisions estratègiques, realitzar comparatives, ajustar la configuració del cotxe i infinitat d'altres possibilitats que es puguin donar.

Així com els equips tenen accés a la totalitat de les seves dades i realitzen els seus anàlisis, que rarament fan públics, els espectadors només tenen accés, generalment, mitjançant la transmissió de televisió, a detalls de cronometratge i algunes estadístiques avançades.

Probablement, per l'espectador mitjà de la Fórmula 1, això és més que suficient, però per algú aficionat a aquest esport, que gaudeix cada cap de setmana de l'espectacle però que també vol entendre el funcionament dels diferents cotxes i aprofundir en aquesta "*Fórmula 1 de les dades*", no ho és.

És per això que, en aquest projecte, s'ha volgut abordar aquest problema utilitzant una de les eines més potents que existeix avui dia per processar llenguatge natural, els *Large Language Models*. Integrant aquest models amb un agent d'intel·ligència artificial, es vol crear a un expert en Fórmula 1 capaç d'obtenir aquestes dades, generar visualitzacions i aportar context a les seves respostes, tot a partir d'una consulta d'un usuari.

1.2 Objectius

Sota aquest context i considerant les tecnologies actuals, l'objectiu final d'aquest projecte és desenvolupar un agent d'intel·ligència artificial basat en LLMs especialitzat en la consulta i la visualització de dades relatives a la Fórmula 1.

No obstant això, s'ha volgut establir un altre objectiu principal relatiu a l'aprenentatge sobre aquests agents i les seves tecnologies associades. S'ha considerat

que, per tal de crear un bon sistema que combini totes aquestes funcionalitats, és fonamental adquirir prèviament els coneixements necessaris.

Degut a això, s'han dividit els objectius del treball en dos apartats diferents:

1.2.1 Objectius relatius a l'aprenentatge

En aquesta etapa d'aprenentatge prèvia al desenvolupament, es volen adquirir els coneixements necessaris per poder dissenyar amb criteri un agent d'intel·ligència artificial, entenent les seves diferents arquitectures, la implementació de les diferents funcionalitats i les tecnologies més adients per fer-ho. Es pot fer una divisió en els següents objectius:

- **Estudi del funcionament dels Large Language Models (LLMs):** Investigar sobre el funcionament d'aquest tipus de models de *deep learning* i adquirir una base de coneixement bàsica sobre les seves capacitats.
- **Estudi dels agents d'intel·ligència artificial basats en LLMs:** Investigar sobre aquest tipus d'agents, aprendre el seu funcionament i com s'integren amb els LLMs i analitzar possibles estratègies que puguin ser útils en la fase de desenvolupament.
- **Anàlisi de les tecnologies actuals:** Investigar sobre l'estat actual del desenvolupament d'agents d'intel·ligència artificial i escollir un *framework* adequat a les característiques de l'agent que es vol implementar.

1.2.2 Objectius relatius al desenvolupament

En l'etapa de desenvolupament, com s'ha esmentat prèviament, l'objectiu principal és dissenyar i implementar un agent d'intel·ligència artificial que permeti consultar i visualitzar dades de Fórmula 1 basant-se en els coneixements adquirits en la fase d'aprenentatge. Per fer-ho, es pot dividir aquest objectiu principal en:

- **Obtenció de dades relatives a la Fórmula 1:** Analitzar les fonts existents per a l'obtenció de dades de F1 i dissenyar un sistema de recuperació d'aquesta informació que permeti a l'agent aportar context a les seves respostes.
- **Disseny de l'agent utilitzant els coneixements adquirits:** Utilitzar les diferents estratègies existents per a la creació d'agents d'intel·ligència artificial per tal d'aconseguir un disseny que permeti implementar amb facilitat les funcionalitats de l'agent.
- **Implementació del disseny:** Utilitzant el *framework* escollit, implementar el disseny de l'agent i dotar-ho d'un front-end bàsic per tal d'interactuar amb ell mitjançant una conversa (*chatbot*).

1.3 Planificació

Aquest projecte, des de la decisió del tema fins a l'entrega final, ha estat realitzat en un període de 15 setmanes on s'ha hagut d'establir una planificació inicial i distribuir adequadament la càrrega de treball per tal d'assolir els objectius.

Aquesta planificació inicial es va dividir en les següents etapes:

- **Etapa 1 - Aprenentatge sobre els Large Language Models (LLMs):** Estudiar aquests models per comprendre el seu funcionament i les seves capacitats.
- **Etapa 2 - Aprenentatge sobre els agents d'intel·ligència artificial:** Veure que són, per a què serveixen i entendre el seu funcionament.
- **Etapa 3 - Aprenentatge sobre diferents conceptes útils per crear un bon agent:** Estudiar els sistemes multiagent per al disseny d'arquitectures més complexes i els sistemes RAG (*Retrieval Augmented Generation*) per a la creació d'una base de coneixement controlada.
- **Etapa 4 - Anàlisi dels frameworks existents per a la creació d'agents:** Seleccionar el *framework* amb què es crearà l'agent analitzant les diferents opcions.
- **Etapa 5 - Disseny de l'arquitectura de l'agent:** Utilitzant tots els coneixements adquirits a les etapes prèvies, dissenyar l'arquitectura inicial de l'agent.
- **Etapa 6 - Implementació de l'agent:** Crear l'agent seguint el disseny establert i avaluar el seu funcionament.
- **Etapa 7 - Redacció de la memòria:** Redactar de forma detallada tots els coneixements adquirits en aquest projecte, la fase de disseny de l'agent i els resultats obtinguts.

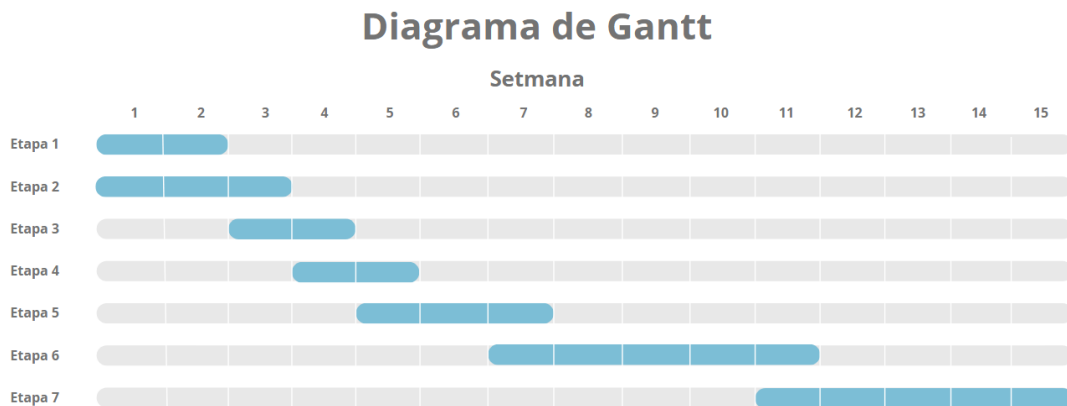


Figura 1: Planificació inicial de les diferents etapes. Font: pròpia

Aquesta planificació inicial ha patit alguna modificació degut a una mala estimació del temps que s'havia d'invertir en cada etapa i de problemes trobats durant l'etapa de disseny i implementació.

Nota: Aquests problemes es detallaran més endavant en l'apartat 4.

Es per això, que s'ha hagut de replanificar la duració de l'etapa de selecció de *framework*, juntament amb l'etapa de disseny de l'arquitectura. També, com a conseqüència, l'etapa d'implementació s'ha hagut de replanificar i, degut a problemes trobats al final d'aquesta, s'ha afegit una nova fase de revisió de funcionament i correcció d'errors.

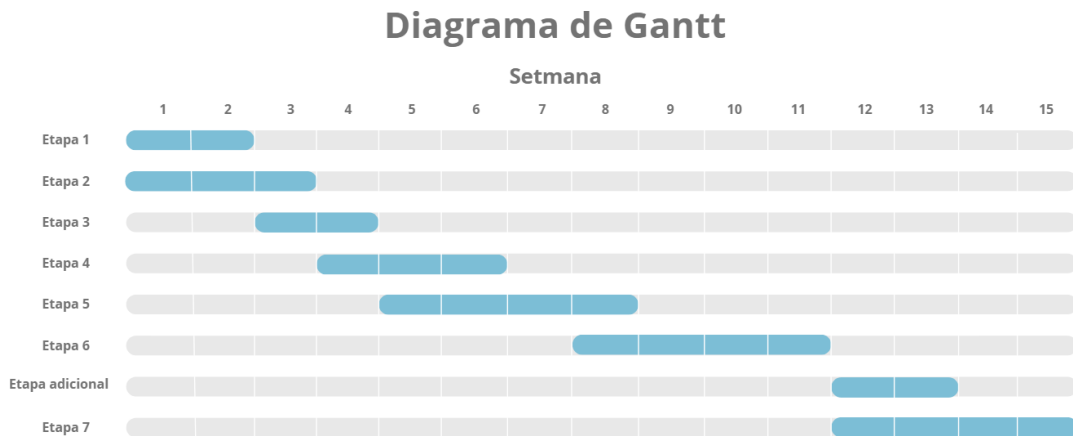


Figura 2: Planificació final de les diferents etapes. Font: pròpia

1.4 Estructura de la memòria

La redacció d'aquest projecte està estructurada en sis apartats, a més de la bibliografia i l'annex. A continuació es pot veure en detall el contingut de cadascun d'aquests apartats:

- **Introducció:** Es descriu el context i la motivació del projecte, establint l'estat actual de l'anàlisi i visualització de dades de la Fórmula 1 i introduint els agents d'intel·ligència artificial com a solució. Posteriorment, es descriuen els objectius de les diferents fases del projecte i una planificació detallada per assolir-los.
- **Marc teòric:** Es desenvolupen els conceptes clau relatius als agents d'intel·ligència artificial basats en LLMs. S'introdueixen aquests LLMs, s'estudia el funcionament dels agents i es veuen algunes estratègies importants que cal considerar en l'etapa de disseny. També s'introdueixen els sistemes multiagent i els sistemes RAG que seran necessaris per la implementació de l'agent.
- **Anàlisi dels recursos actuals:** Es defineixen els requeriments del sistema i s'analitza l'estat actual dels *frameworks* de creació d'agents per tal de

seleccionar un adequat a les característiques del projecte. També, s'estudien les diverses formes d'obtenir dades de Fórmula 1 i de crear una base de coneixement.

- **Disseny i implementació del sistema multiagent:** Es defineix l'arquitectura del sistema, establint els diferents agents i les seves corresponents eines. També, s'analitzen amb detall les llibreries utilitzades i s'explica el procés d'implementació.
- **Resultats:** Es verifica el compliment dels requisits mostrant els resultats obtinguts i s'avalua el rendiment del sistema.
- **Conclusions i treball futur:** Es comprova l'assoliment dels objectius del projecte i s'estudien possibles millores del sistema que es podrien implementar en treballs futurs.

2 Marc teòric

Tal com s'indica a la planificació del projecte, una part significativa del temps invertit en aquest treball ha estat dedicada a l'aprenentatge sobre els agents d'intel·ligència artificial per tal d'entendre el seu funcionament, la seva presa de decisions i les bases per crear un bon agent. Conseqüentment, també s'ha dedicat bastant temps a entendre com funcionen els LLMs (Large Language Models), especialment en el context dels agents, ja que aquests serveixen com a 'cervell' i els hi donen la capacitat de processar el llenguatge natural, de raonar i de prendre decisions.

Per tant, per tal d'entendre millor la posterior fase de disseny, és convenient dedicar una secció d'aquest treball a explicar amb detall aquests conceptes que han estat claus a l'hora de dissenyar un bon sistema multiagent.

2.1 Introducció als Large Language Models (LLMs)

Abans d'aprofundir en el concepte dels agents, és important comprendre què són aquests mecanismes que utilitzen per poder processar el llenguatge natural, raonar de forma correcta i generar una resposta coherent. Tal com s'ha esmentat prèviament, aquests mecanismes són els anomenats *Large Language Models* (LLM) que, de forma molt resumida, són un tipus de model d'intel·ligència artificial que funciona molt bé a l'hora d'entendre i generar llenguatge natural.

Nota: L'objectiu d'aquest apartat és entendre de forma superficial que són aquests models i el seu funcionament ja que, si s'entra en detall, ràpidament es pot veure que és un tema d'una complexitat força elevada i la seva comprensió queda fora dels objectius d'aquest projecte.

2.1.1 Definició

Es poden donar diverses definicions dels LLMs, totes correctes, adaptades segons el grau de coneixement. Per exemple, una definició inicial que es podria donar és la següent:

'Un LLM és una funció matemàtica sofisticada que prediu quina paraula ve a continuació per a qualsevol fragment de text.' [3]

Aquesta definició, tot i que és correcta, no és gaire precisa. Què és aquesta funció matemàtica sofisticada? Com prediu la següent paraula? La resposta a aquestes preguntes és clau per entendre el seu funcionament intern. És per això, que és necessari donar una definició una mica més detallada:

'Un LLM és un tipus de model d'aprenentatge profund dissenyat per a tasques de processament del llenguatge natural com ara la generació de llenguatge. Són models de llenguatge amb molts paràmetres i que s'entrenen amb aprenentatge auto-supervisat en una gran quantitat de text.' [4]

A partir d'aquesta definició més rigorosa, es pot deduir que la 'funció matemàtica sofisticada' és, en realitat, un tipus de model d'aprenentatge profund que, gràcies al seu elevat nombre de paràmetres i a l'entrenament sobre quantitats massives de dades textuais, és capaç d'aprendre patrons i estructures pròpies del llenguatge natural per poder predir la següent paraula donat un fragment de text.

2.1.2 Funcionament

Gairebé tots aquests models d'aprenentatge automàtic estan basats en l'arquitectura *Transformer*, presentada per Google en un article d'investigació anomenat *Attention Is All You Need* l'any 2017 [5]. Aquesta arquitectura d'aprenentatge profund està basada en l'algorisme d'atenció, el qual, a l'hora de predir la següent paraula donat un text, assigna pesos diferents a cada paraula segons el seu context [6].

Per exemple, per determinar la següent paraula de la frase 'El cel és de color...', les paraules 'cel' i 'color' tindran més importància que 'El' o 'de' ja que contenen gran part del significat d'aquesta.

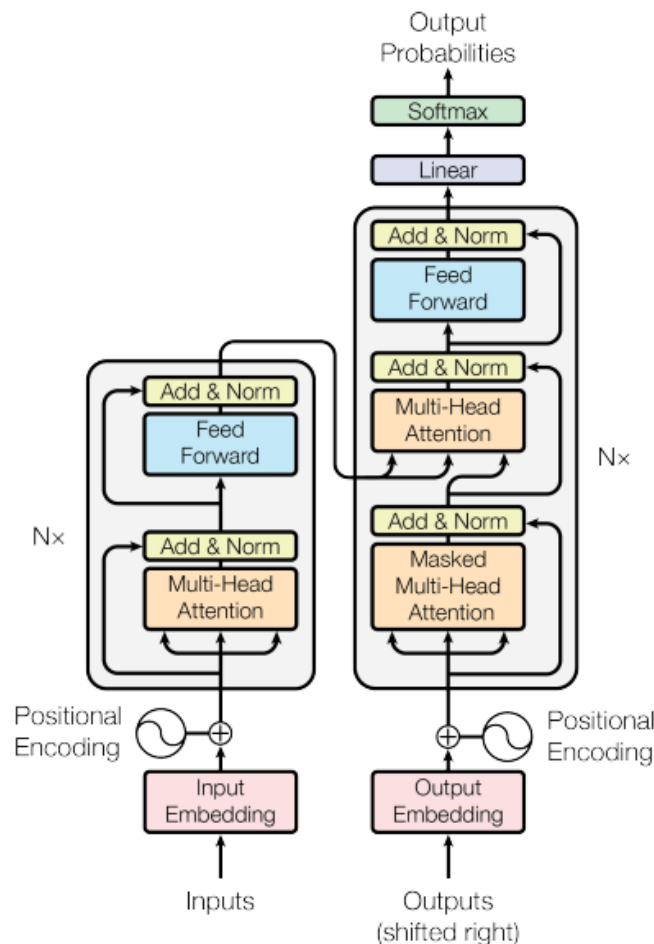


Figura 3: Arquitectura original del model *Transformer*. Font: [5]

En aquesta imatge, es pot veure l'arquitectura original del model *Transformer*

presentada en l'article prèviament esmentat, on es pot observar un codificador a l'esquerra i un descodificador a la dreta. En el cas dels LLMs, la gran majoria d'aquests, estan basats en una arquitectura *decoder-based Transformer*, que només utilitza la part del descodificador.

Degut a la complexitat del funcionament i de les operacions internes d'aquesta estructura, per simplicitat, en aquest projecte, es tractarà com una 'capsa negra' que, donada una entrada de text, retorna una llista amb les possibles següents paraules i les seves corresponents probabilitats. Sense entrar en molt detall, però, es veurà el seu funcionament.

Primer de tot, el text d'entrada es fragmenta en tokens, com s'explica al següent apartat, i es passen a una matriu anomenada *embedding matrix* que mapeja cada token a un vector d'un espai vectorial de dimensió molt elevada (p.ex. 12288 dimensions per GPT-3). Després, es passa per una etapa on s'aplica codificació posicional per preservar l'ordre d'aquests tokens en la seqüència i, posteriorment, s'envia tot a una pila de blocs de *decoder-based Transformers*.

Dins d'aquesta estructura, aquests tokens passen per diferents capes i, mitjançant l'algorisme d'auto-atenció, són capaços 'd'atendre' a tokens que apareixen prèviament, modificant així el seu significat o la seva importància dins el text. Per exemple, no és el mateix un 'banc de peixos' que un 'banc per seure' o un 'banc financer'. L'última capa, la *feed forward layer*, és l'encarregada d'extreure la informació d'aquests vectors mitjançant una xarxa neuronal *feed-forward* on es produeixen transformacions no lineals que contribueixen al refinament dels valors de cada token segons el seu context.

Després de varies iteracions d'aquest procés, utilitzant una matriu semblant a la matriu d'embeddings (però transposada), es mapeja l'últim vector de la seqüència generada (el que conté els pesos del següent token) a un vector que determina, per cada token del vocabulari, una puntuació anomenada *logit*. Finalment, aplicant una funció softmax, es transformen aquestes puntuacions en una distribució de probabilitats sobre tots els tokens.

2.1.3 Tokenització i generació del següent token

Prèviament, s'ha dit que aquests models generen la següent paraula donat un text. Aquesta afirmació, però, no és del tot correcta. Això és degut a que els models, per raons d'eficiència, treballen amb 'tokens' que, generalment, són divisions d'una paraula. Per tant, el que realment fan és generar el següent token que, eventualment, generarà la següent paraula.

Cada LLM pot tenir un tokenitzador diferent que generarà diferents tokens per la mateixa frase. Per exemple, si es tokenitza la frase 'Els models de llenguatge són molt interessants', obtenim diferents resultats depenent del model utilitzat:

- GPT-4 (13 tok.): E/ls/ models/ de/ ll/engu/at/ge/ s/ón/ molt/ interess/ants
- GPT-3 (15 tok.): Els/ models/ de/ ll/engu/at/ge/ s/ón/ m/olt/ int/e/-ress/ants

- Grok-1 (11 tok.): Els/ models/ de/ ll/engu/at/ge/ són/ molt/ interess/ants

Per tal de controlar la generació de text es fan servir tokens especials, específics per a cada model, que s'utilitzen, per exemple, per indicar el principi o el final d'un missatge. El més important d'aquests tokens és el token que determina el final d'una seqüència (EOS token) [7].

Per tant, donat un text d'entrada, aquest es tokenitza i es passa al model que retorna, per cada token del seu vocabulari, les probabilitats de ser el següent token de la seqüència. Una vegada seleccionat aquest token, aquest s'inclou en el text d'entrada i es torna a passar al model, és a dir, el model és autoregressiu. Aquest procés continua fins que es genera el token EOS.

Si sempre s'agafés el token amb més probabilitat es tindria un model determinista que sempre produiria la mateixa sortida, per evitar-ho, es pot incloure al model un paràmetre especial. Aquest paràmetre és la temperatura, que dona un cert grau d'aleatorietat a l'hora de seleccionar el següent token. D'aquesta forma, es pot controlar el model i, amb temperatures properes a 0, aquest produirà sortides més deterministes. Si s'augmenta aquest valor, el model tindrà més 'creativitat' i, en dos execucions diferents del mateix text, es poden obtenir resultats molt diferents. S'ha d'anar amb compte, però, de no augmentar massa aquest valor ja que el model podria començar a generar text sense massa sentit [8].

2.2 Agents d'intel·ligència artificial basats en LLMs

Una vegada introduïts els *Large Language Models* i observades les seves capacitats, en aquest apartat s'explorà una de les seves aplicacions directes més importants avui dia, els agents d'intel·ligència artificial. Primerament, però, abans d'explicar el que són aquests tipus d'agents, s'ha de definir el seu concepte general.

2.2.1 Definició

Si es consulta el diccionari (DIEC2) la definició de la paraula 'agent', es poden trobar aquestes definicions:

- *'Poder o causa activa, allò que té el poder de produir un efecte.'*
- *'Reactiu químic.'*
- *'Persona que té la missió d'obrar per algú.'*

Es pot observar com la definició pot variar depenent del context en el que s'utilitzi la paraula però totes comparteixen la mateixa essència. Un agent és una entitat que té la capacitat de realitzar accions que produeixen, o no, un efecte.

En el context de la intel·ligència artificial, més precisament, dels agents basats en LLMs, aquesta definició segueix sent correcta, tot i que s'hauria d'adaptar al domini que representa. Russell i Norvig van definir un agent d'IA com *'qualsevol*

entitat capaç de percebre el seu entorn mitjançant sensors i actuar en ell mitjançant actuadors' [9].

Si es porta aquesta definició a l'àmbit dels agents basats en LLMs, ràpidament es pot veure com aquests 'sensors', que capten l'entorn, són en realitat el text d'entrada que reben. Seguint aquesta analogia, els 'actuadors' serien les accions que pot fer per actuar sobre l'entorn, com ara generar text, cridar funcions, navegar per internet, etc. L'entorn en aquest cas podria estar format per l'usuari, el sistema, un conjunt de funcions, documents, etc. Totes aquestes diferents parts d'un agent es veuran en detall en els pròxims apartats.

El que fa especials a aquests tipus d'agents és la forma que tenen d'interactuar amb el seu entorn. Mitjançant la integració d'un LLM com a un 'cervell', són capaços de raonar, planejar i prendre accions en funció del seu estat. Per tal de visualitzar el seu funcionament, a continuació es mostra un petit exemple simplificat d'un agent del temps:

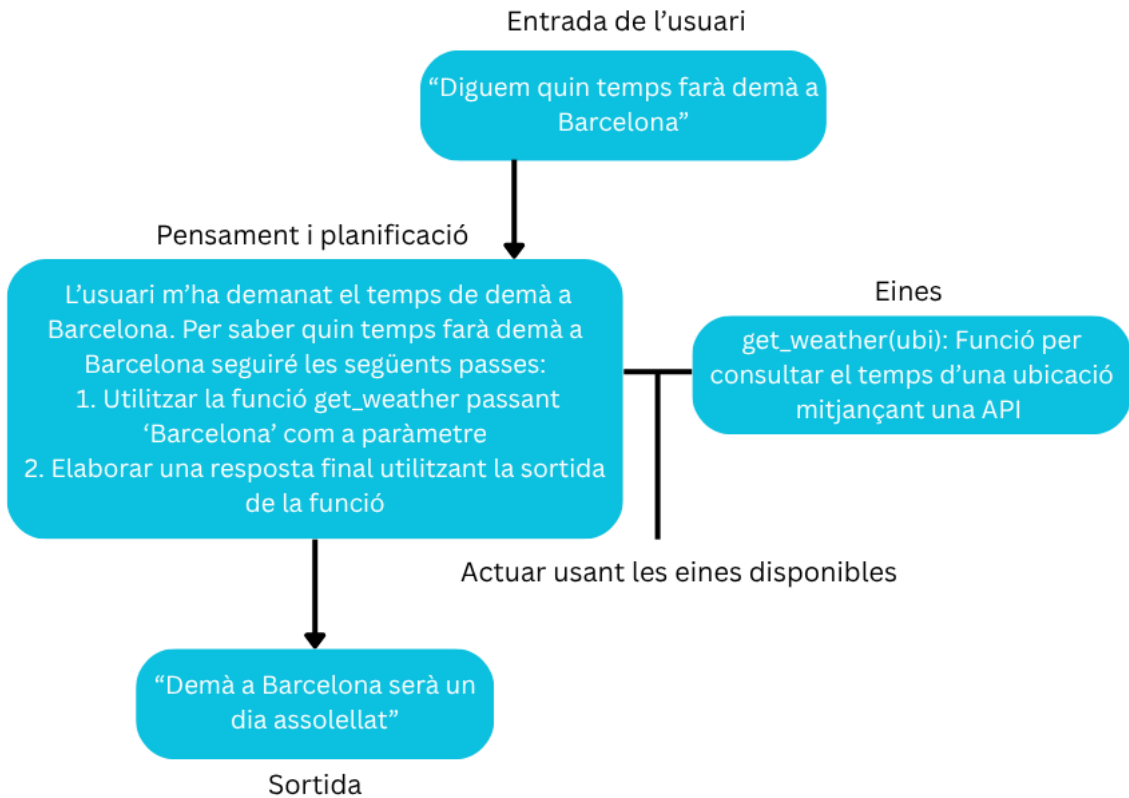


Figura 4: Exemple de funcionament simple d'un agent del temps. Font: pròpia

Per tant, vista aquesta analogia amb la definició de Russell i Norvig i incloent els LLMs com a mitjà d'interacció amb l'entorn, una definició adaptada d'aquest tipus d'agents podria ser la següent:

'Un agent és un sistema que, mitjançant un LLM, és capaç d'interactuar amb el seu entorn i, combinant raonament, planificació i execució d'accions —amb el suport d'eines externes si cal—, assolir un objectiu definit per l'usuari.' [10]

2.2.2 Integració amb els LLMs

Una vegada definit aquest concepte, el següent pas es veure el funcionament de la integració d'un LLM com a 'cervell' de l'agent, ja que, com es pot veure en l'exemple previ, el seu flux d'execució està controlat per la sortida d'aquest model.

Aquesta capacitat de controlar el flux d'execució per part del model és el que s'anomena 'agència' o *agency*. D'aquesta forma, contra més influència tingui la sortida del model en aquest flux, més agència tindrà. Una conseqüència directa d'aquest fet és la pèrdua de predictibilitat en el comportament del sistema, que pot provocar resultats no desitjats. Per tant, a l'hora de dissenyar un agent, s'ha de tenir en compte quanta capacitat de control sobre aquest flux es delega al LLM.

Es pot, llavors, crear una classificació d'aquests agents segons el seu nivell d'agència (ordenada de menor a major): [11]

- **Processador simple:** no controla el flux d'execució.
- **Encaminador:** controla una estructura 'if/else' per decidir el camí.
- **Executor de funcions:** controla l'execució de diferents funcions.
- **Multi-step Agent:** controla la iteració i la continuació del programa.
- **Multi-Agent:** un flux d'execució pot iniciar un altre flux d'execució d'un altre agent.
- **Code Agent:** pot generar codi, definir les seves pròpies funcions i iniciar altres agents.

Aquesta classificació és acumulativa, és a dir, cadascun d'aquests nivells d'agència inclou les capacitats dels nivells anteriors.



Figura 5: Flux d'execució d'un encaminador i d'un multi-step agent. Font: [12]

Per explicar el funcionament d'aquests agents, s'analitzaran en detall els *Code Agents*, la representació del nivell màxim d'agència, que, com s'ha esmentat prèviament, inclou totes les capacitats dels nivells previs.

Abans d'això, però, cal definir un concepte molt important que permet als LLMs comunicar-se amb el 'món real' i expandir les seves capacitats. Reprenent l'exemple previ de l'agent del temps, quan l'usuari demana informació sobre el clima en un lloc determinat, s'ha definit l'agent de tal forma que el LLM disposa d'una funció específica per recuperar aquesta informació. Però, què passaria si no tingués aquesta funció? Com s'ha vist a la introducció als LLMs, aquests generen text basant-se en els conjunts de dades amb què han estat entrenats i, per tant, no inclouen informació d'esdeveniments posteriors al seu entrenament. Així doncs, si se'ls pregunta per aquesta informació sense disposar d'aquesta funció, la resposta segurament sigui incorrecta.

Aquestes funcions s'anomenen eines o *tools* i serveixen per complementar i ampliar les capacitats d'un LLM. Alguns exemples d'eines que es poden incorporar en un agent poden ser un cercador a internet, una calculadora, una connexió amb una API, etc.

A l'hora de crear aquestes eines, és necessari tenir present que han de ser identificades correctament per part del LLM, així doncs, s'ha de tenir molta cura per descriure-les. Per exemple, si tenim una eina que suma dos nombres, una possible descripció seria la següent: **Tool name:** `calculator`, **Description:** `Sum two numbers.`, **Arguments:** `a: float, b: float`, **Output:** `float`.

Per informar el LLM que disposa d'aquesta eina, cal afegir aquesta descripció al que s'anomena *system prompt* o *system message*. Aquest missatge s'envia al model al principi de cada interacció i també li proporciona informació sobre el seu comportament (vegeu l'apartat 2.2.3). D'aquesta forma, si LLM considera convenient utilitzar aquesta eina, generarà text representant aquesta crida (p.ex. `calculator(3,4)`). Posteriorment, l'agent processarà la resposta del model, identificarà aquesta crida i executarà aquesta funció. [13]

2.2.3 Funcionament

Al final de l'apartat previ, s'ha mencionat el concepte de *system prompt* i de com inclou informació relativa al comportament esperat de l'agent. De fet, aquest missatge, juntament amb les entrades de l'usuari, és l'única forma de què disposa el LLM per conèixer el seu entorn. Per tant, és mitjançant aquesta eina que es pot incloure informació sobre com hauria de processar els seus pensaments i prendre les seves accions, és a dir, establir el seu comportament. Per exemple, si es volgués crear un agent encaminador, es podria tenir el següent *system prompt*:

Exemple de system prompt per a un agent encaminador

Ets un agent classificador de correus electrònics encarregat de determinar el CAMÍ que s'ha de seguir segons si el correu és SPAM o no.
Hi ha dues opcions possibles:
- "spam": quan detectes que el correu és SPAM
- "no spam": quan detectes que el correu no és SPAM
RESPON només amb el nom del camí que s'ha de seguir: "spam" o "no spam".

Es pot observar que és en aquest missatge on se li donen les 'capacitats' a l'agent i el control que té sobre el flux d'execució, és a dir, el seu nivell d'agència.

Per tasques simples com les d'aquest agent encaminador, aquest *system prompt* és més que suficient. Tanmateix, quan les tasques són més complexes, és necessari, per tal de millorar la precisió i la qualitat de la resposta, adoptar altres formes de comportament.

Actualment, la forma més popular de crear agents es basa en el mètode *ReAct* [14], una estratègia de *prompting* introduïda en un article d'investigació l'any 2022 [15]. Aquesta estratègia consisteix en dividir la tasca inicial en subproblemes més petits, permetent així al LLM un processament del problema amb més detall i precisió.

A continuació, es pot observar una figura de l'article esmentat que realitza una comparació del funcionament de varis mètodes de *prompting* amb dues tasques que inclouen aritmètica, sentit comú i raonament simbòlic:

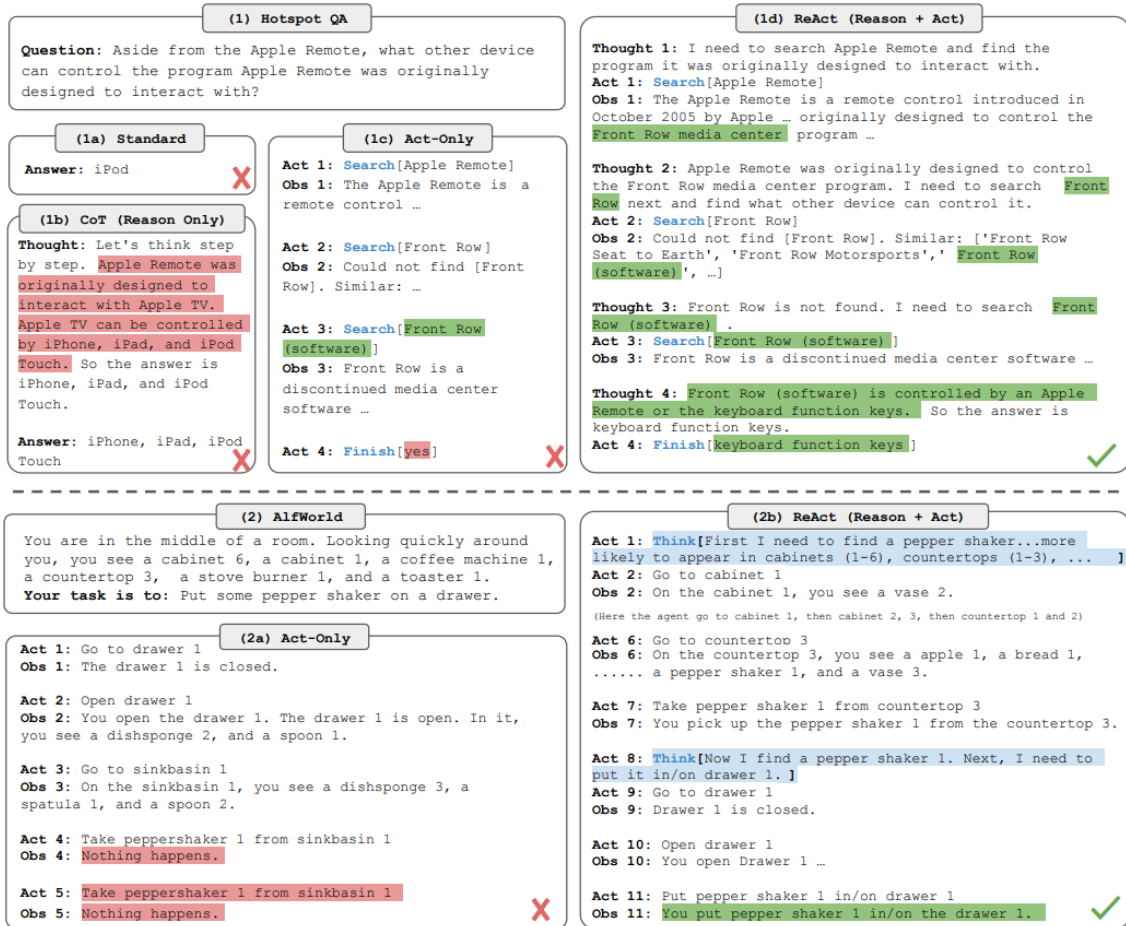


Figura 6: Comparació de 4 mètodes de *prompting*. Font: [15]

Amb aquest mètode, els agents funcionen de forma iterativa, solucionant el problema per passes considerant aquestes 'subtasques'. Cadascuna d'aquestes iteracions conté una primera etapa de raonament (*reasoning step*) seguida d'una etapa d'acció (*action step*) (d'aquí surt el nom *ReAct*) on s'executen les diferents eines de-

finides a l'agent que, idealment, l'aproparan a la solució final. Abans de la següent etapa de raonament, es proporciona el resultat de l'etapa d'acció al LLM per tal de que pugui reflectir sobre els resultats o, dit d'una altra forma, veure la resposta que produeix l'entorn. Aquest procediment és especialment rellevant ja que permet a l'agent veure les conseqüències de les seves accions (informació d'una API, missatges d'error, etc.) i, en funció del nou context, readaptar la seva estratègia. Finalment, quan l'agent té la resposta definitiva, finalitza aquest cicle proporcionant-la.

Si es fa una abstracció d'aquest funcionament, es tenen, en realitat, tres etapes diferents funcionant en un bucle continu de '*Thought-Action-Observation*' [16].

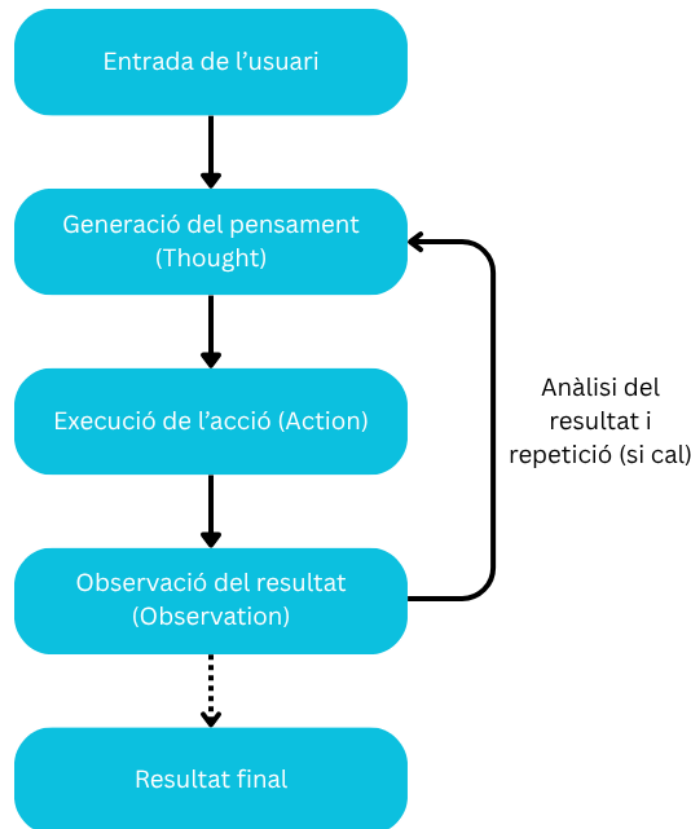


Figura 7: Cicle *Thought-Action-Observation* (TAO). Font: pròpia

En l'etapa d'acció, com podria ser evident, un cop el LLM genera l'acció que s'ha d'executar, cal que interrompi la generació de text i esperi el resultat abans de reprendre-la. Aquesta generació de l'acció, pot estar implementada de diverses formes.

Tradicionalment, l'enfocament ha sigut formatar aquestes accions en format JSON que, posteriorment, mitjançant un analitzador sintàctic o *parser*, permeten extreure la funció i els seus arguments. Recentment, però, s'ha publicat l'article '*Executable Code Actions Elicit Better LLM Agents*' [17] que proposa que aquestes siguin descrites en blocs de codi executables (en el cas de l'article s'utilitza Python). Aquest enfocament, ofereix multitud d'avantatges que s'aprofiten de l'expressivitat dels llenguatges de programació per definir accions complexes i de l'entrenament dels LLMs més moderns amb dades que inclouen codi.

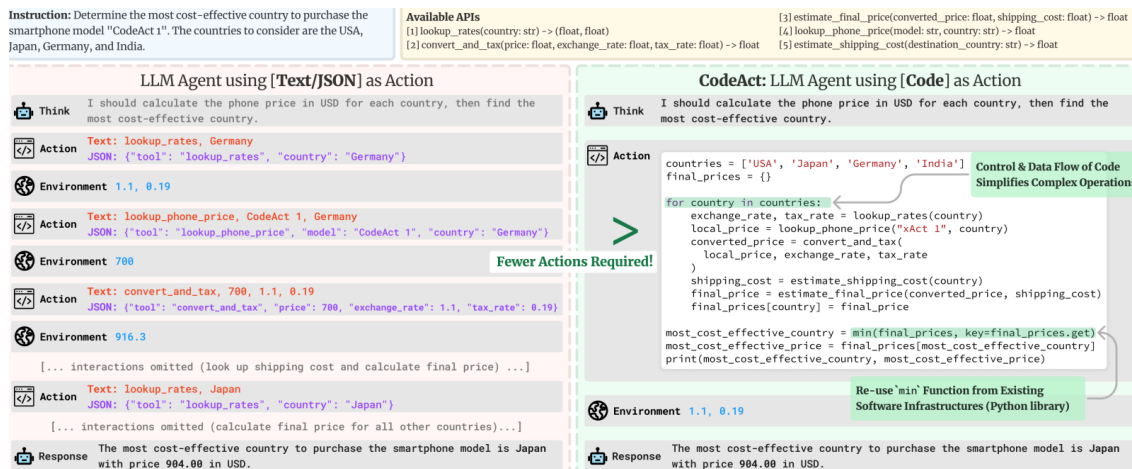


Figura 8: Comparació entre accions formatades amb JSON i Python. Font: [17]

En aquesta figura s'observa com, donada una tasca que implica múltiples crides a funcions, en el cas de l'agent que formata les accions en JSON, això implica un increment considerable d'iteracions del cicle *TAO*. D'altra banda, gràcies a aquesta expressivitat per representar lògica complexa, el *Code Agent* pot realitzar aquesta mateixa tasca amb moltes menys accions. També, pot reutilitzar funcions de diferents accions o tasques i integrar directament funcionalitats de llibreries externes o APIs.

Per concloure aquest apartat, es veurà un exemple simple de funcionament integrant tots els conceptes explicats fins ara (*system prompts*, eines, mètode *ReAct*, cicle *TAO* i *Code Agents*) en el context de l'agent del temps explicat prèviament:

Exemple de system prompt per a un agent del temps

Ets un agent intel·ligent que resol tasques relacionades amb el clima de forma eficient i precisa. Per resoldre una tasca, ho has de fer de forma iterativa, procedint pas a pas i utilitzant un bucle de Pensament/Acció/Observació. A cada pas, en el 'Pensament' has d'explicar el teu raonament i les eines que vols utilitzar. En l'Acció' has d'escriure el codi corresponent a la crida de les funcions. En l'Observació' has d'incloure el resultat de l'execució d'aquest codi que estarà disponible com a entrada de la següent iteració.

Al final, has de retornar la resposta final amb el prefix 'Resposta final: '

Tens accés a les següents eines:

Nom de l'eina: get_weather, Descripció: Retorna el temps d'un lloc determinat, Arguments: ubicació: string, Sortida: string

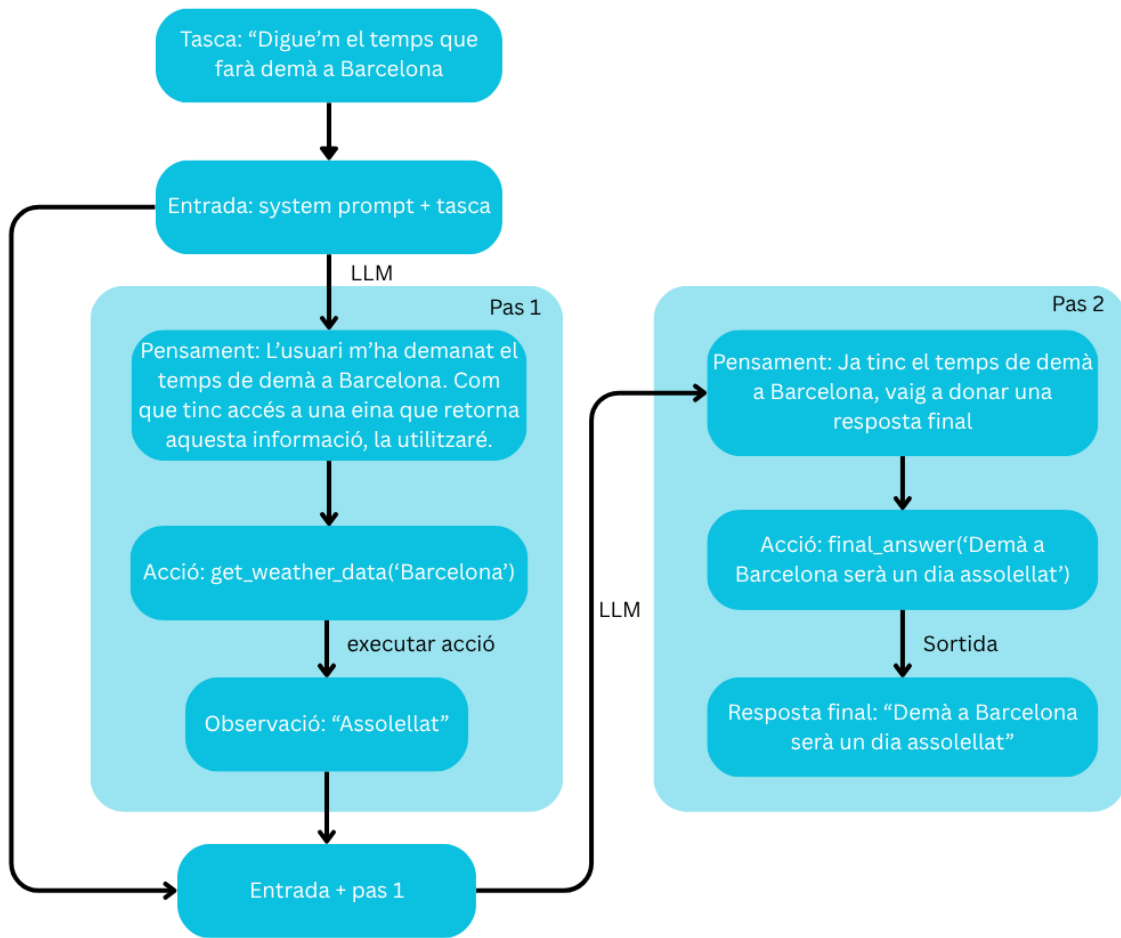


Figura 9: Diagrama d'execució de l'agent implementant el mètode ReAct. Font: pròpia

2.2.4 Com crear bons agents?

En aquest apartat es mostraran varies guies i bones pràctiques que seran d'utilitat en el procés de creació d'un bon agent. Primerament, però, s'ha d'establir el significat d'aquest concepte. Un 'bon' agent, en el context d'aquest projecte, és aquell capaç de produir els resultats esperats minimitzant els costos, la latència i els errors [18].

En el procés de creació d'un agent, primer cal establir el nivell d'agència que se li vol donar al LLM que, degut a la impredictibilitat de les seves sortides, contra menys se li doni, menys risc d'error s'estarà introduint en el sistema. Dit d'una altra forma, contra més determinista sigui l'agent, millor. És per això que, abans de crear un agent, és necessari analitzar si realment fa falta introduir un LLM per gestionar el flux de l'aplicació. En cas afirmatiu, l'objectiu serà minimitzar els errors que pugui cometre simplificant al màxim el flux d'execució del sistema i comunicant-li curiosament l'entorn en el qual treballa.

Una bona pràctica molt utilitzada és l'ús d'exemples d'execució en el *system prompt* per tal de mostrar al LLM com hauria d'actuar per completar una tasca. Segons el nombre d'exemples que tingui disponibles es poden diferenciar tres tipus de tècniques de *prompting*, *zero-shot* (no hi ha exemples), *one-shot* (el LLM dispo-

sa d'un únic exemple) o *few-shot* (varis exemples). Aquests exemples han de ser rellevants, diversos i estar ben escrits, ja que un petit error pot confondre al model i donar un resultat no desitjat [8].

Un altra forma de minimitzar aquests errors és dissenyar les eines (*tools*) perquè proporcionin el màxim d'informació possible al LLM, sobretot quan es produeixen errors en la seva execució. Així, si el LLM disposa d'informació detallada de l'error que s'ha produït, pot reformular la seva planificació i intentar solucionar la tasca d'una altra forma. Una altra avantatge de proporcionar informació detallada de les eines és la capacitat que dona d'integrar amb facilitat un concepte molt potent a l'hora d'interactuar amb agents.

Aquest concepte és el *human-in-the-loop*, que integra a l'usuari dins del flux d'execució de l'agent [19]. Això és especialment útil, per exemple, en el cas de que es doni una tasca que requereix la crida d'una eina i, aquesta, necessita d'informació addicional per tal d'executar-se correctament. Per exemple, si l'usuari pregunta per la temperatura actual sense especificar el lloc, l'agent, que disposa d'una eina per consultar aquesta temperatura donada una ubicació, abans d'executar aquesta funció, interromp el flux i pregunta a l'usuari per aquesta informació. D'aquesta forma, al afegir aquesta interacció amb l'usuari, es poden minimitzar els errors i readaptar de forma dinàmica la tasca segons les indicacions que es donin a l'agent.

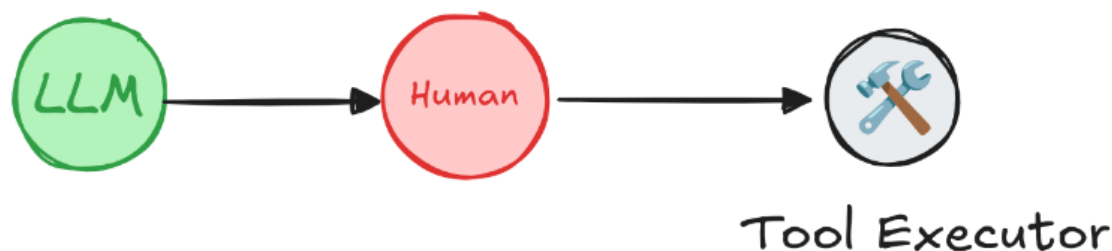


Figura 10: Esquema de funcionament del *Human-in-the-loop* . Font: [19]

Hi ha vegades, però, que tota aquesta informació que se li dona mitjançant el *system prompt* i la guia de l'usuari no és suficient i no és capaç d'organitzar el flux d'execució de forma correcta. En aquest cas, cal tenir en compte quin model s'està utilitzant com a 'cervell' i valorar si és suficient per al context de l'agent. Generalment, agents que requereixen més planificació i més capacitat de resolució de problemes complexos es beneficien de models més potents, així, es redueixen aquests errors a canvi d'un increment dels costos.

Finalment, un aspecte important a valorar en el cas dels *Code Agents* és el tema de la seguretat. En aquest tipus d'agents, com s'ha observat en el apartat anterior, s'executa directament codi generat per un LLM, això, si no es controla, pot comportar riscos importants a l'entorn d'execució de l'agent. Hi ha diverses formes de gestionar aquest problema, per exemple, es pot crear un entorn d'execució (*sandbox*) diferent al entorn local, mitjançant eines com *Docker* [28], on executar els blocs de codi generats per l'agent.

2.3 Sistemes multiagent

Per garantir un correcte funcionament, com a norma general, un agent hauria d'estar especialitzat únicament en un tipus de tasques determinades. Així, és molt més simple controlar el seu comportament esperat mitjançant el *system prompt* i evita possibles al·lucinacions i crides errònies a eines per part del LLM, especialment quan el nombre d'aquestes és elevat [20].

En cas de necessitar un agent que integri varies funcionalitats diferents, la manera més adequada de fer-ho és mitjançant un sistema multiagent.

2.3.1 Definició

Com es pot inferir del nom, un sistema multiagent (MAS) està format per diversos agents que col·laboren entre si per resoldre una tasca determinada. D'aquesta forma, gràcies a aquesta col·laboració, són capaços d'abordar problemes més complexos i distribuir les diferents sub-tasques segons les capacitats de cada agent.

Aquesta distribució de responsabilitats segons l'especialitat de cada agent, proporciona una sèrie d'avantatges tant en el seu desenvolupament com en el funcionament [21]:

- **Especialització:** Cada agent té el seu *system prompt* optimitzat per a la tasca que ha de realitzar, tenint així més control sobre el seu comportament.
- **Modularitat:** Un disseny modular facilita el desenvolupament al poder tractar cada agent com una entitat independent.
- **Escalabilitat:** En cas d'afegir noves funcionalitats, es pot crear un nou agent sense haver de modificar els altres.
- **Robustesa:** En cas d'error o d'un agent que no sigui capaç de solucionar la seva tasca, es pot redirigir el flux d'execució a un altre agent.

Dissenyar un sistema multiagent, però, també comporta una sèrie de desavantatges que cal considerar:

- **Coordinació:** Cal establir un mecanisme de control per tal de coordinar correctament els agents.
- **Comunicació:** L'intercanvi d'informació entre agents indueix un augment de la latència.
- **Traçabilitat:** Complexitat per entendre el flux d'execució del sistema per donar una resposta final.
- **Increment dels punts de fallada:** Al augmentar el nombre d'agents, s'augmenta el nombre de crides al LLM i, per tant, també el risc de fallada. També es poden produir errors en la comunicació.

Una vegada considerades totes aquestes virtuts i defectes dels sistemes multiagent, convé analitzar les diferents architectures possibles que es poden implementar.

2.3.2 Architectures

En el moment d'establir una arquitectura en un sistema multiagent, es poden considerar varies opcions, depenent de com es vulgui gestionar la comunicació entre els agents [20] [21]:

- **Descentralitzada:** Cada agent es pot comunicar directament amb altres agents formant així una xarxa *peer-to-peer*. Ofereix una alta robustesa però la manca d'un agent coordinador pot provocar pèrdua de coherència i possibles conflictes entre agents.
- **Centralitzada (manager):** Existeix un agent director que coordina els agents i delega les diferents tasques entre ells. Facilita molt la coordinació per resoldre tasques complexes i dona una gran escalabilitat. D'altra banda, els agents supervisats tenen menys autonomia i, degut a que tota la informació ha de passar per aquest supervisor, es pot produir un 'coll d'ampolla', augmentant considerablement la latència.
- **Seqüencial:** Cada agent s'encarrega de realitzar una tasca determinada i la seva sortida es passa com a entrada a un altre agent. És fàcil d'implementar i molt útil quan el flux d'execució és simple i les tasques es poden resoldre de forma seqüencial. Una desavantatge clara d'aquesta arquitectura és la poca flexibilitat que dona i la baixa capacitat d'adaptació davant errors.

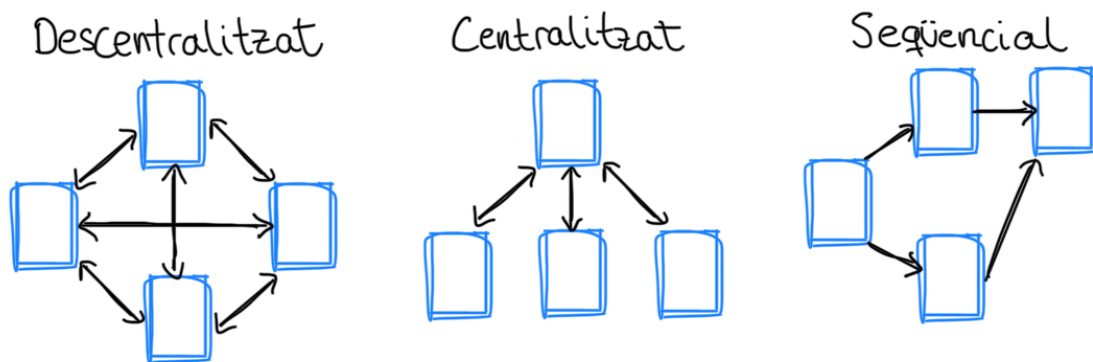


Figura 11: Esquemes d'arquitectures multiagent. Font: pròpia

Aquí es presenten tres tipus d'arquitectures bastant comunes en el disseny de sistemes multiagent. Això no implica, però, que siguin les més eficients. Es poden considerar altres tipus d'arquitectures diferents que, depenent del context d'ús de cada sistema, poden funcionar millor que aquestes.

2.4 Integració amb sistemes RAG

Com s'ha explicat en els apartats previs, els LLMs estan entrenats amb quantitats massives de dades amb l'objectiu d'aprendre coneixement general del món en el que vivim. Una vegada el LLM ha estat entrenat, qualsevol informació posterior a aquest entrenament no formarà part del model i, per exemple, si se li demana el preu de l'energia el dia d'ahir, aquest no disposarà d'aquesta informació i s'inventarà el resultat (al·lucinació).

Nota: Prèviament, s'ha vist que això es pot solucionar dotant a aquest LLM d'unes eines que li permeten interactuar amb el món real i recopilar aquesta informació.

Com que els LLMs generen text basant-se en aquest coneixement general, que pot no ser gaire rellevant o pot estar desactualitzat, és necessari implementar d'alguna forma una base de coneixement controlada de la qual el LLM pugui buscar informació quan sigui necessari.

Aquí és on entren els sistemes RAG (*Retrieval Augmented Generation*), que es basen en utilitzar un LLM per respondre a una consulta d'un usuari (com es feia fins ara), però utilitzant una base de coneixement on fonamentar la seva resposta [22]. D'aquesta forma, es poden controlar molt més les respostes d'aquest LLM, evitant al·lucinacions i proveint al model amb informació específica del domini en el qual està treballant.

2.4.1 Funcionament

Abans de veure el flux d'execució d'un sistema RAG, és necessari veure com es prepara aquesta base de coneixement per tal de que el LLM pugui respondre consultes basant-se en ella.

Primer de tot, s'ha d'obtenir tota la informació que es vulgui incloure. Aquesta, per exemple, pot venir d'una base de dades estructurada, una sèrie de documents de text (p.ex. PDFs) o d'una API. Posteriorment, es divideixen tots aquests documents en fragments més petits ja que, d'aquesta forma, és més fàcil trobar informació rellevant a la consulta. L'objectiu principal és que aquests fragments continguin informació semànticament relacionada i, per tant, és important anar amb precaució a l'hora de separar-los. Si són massa petits, es dividiran les idees i, si són massa grans, es diluiran.

Per realitzar aquesta separació hi ha diversos mètodes, per exemple, la fragmentació recursiva estableix una mida de fragment orientativa i va dividint el text en parts cada cop més petites (paràgrafs, línies, final d'una frase, etc.). Aquesta separació continua fins que retorna fragments que s'apropen a aquesta mida orientativa [24].

Per millorar aquest procés de separació es podrien considerar altres formes de fragmentació més potents (amb l'inconvenient que són més difícils d'implementar). Ja que l'objectiu principal és obtenir fragments semànticament relacionats, una de les millors formes de fer-ho és mitjançant *embeddings* [25].

Una vegada obtinguts tots els fragments, es calculen els diferents *embeddings* mitjançant un *encoder-based Transformer* i es guarden en una base de dades vectorial. Quan ja es té preparada la base de coneixement, es pot observar el seu funcionament en la següent figura:

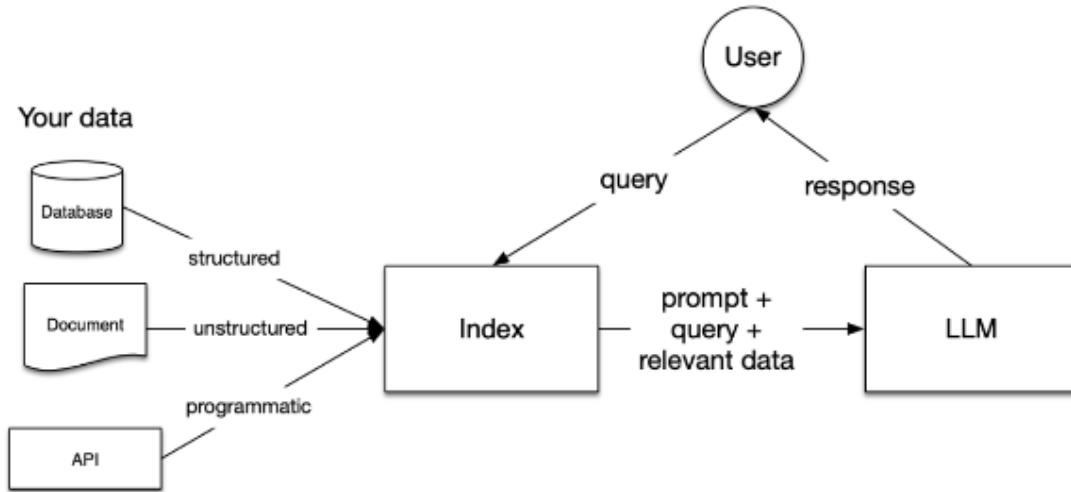


Figura 12: Esquema de funcionament d'un sistema RAG. Font: [23]

Primerament, per tal d'obtenir els fragments d'informació rellevants donada una consulta d'un usuari, aquesta ha de seguir el mateix procés d'*embedding* amb el mateix model. Seguidament, es realitza una cerca en la base de dades vectorial on es calcula la distància d'aquest vector d'*embeddings* de la consulta amb els *embeddings* prèviament calculats dels diferents fragments. Computades aquestes distàncies, es retornen els k documents més rellevants (més pròxims) on k , generalment, és un valor entre 5 i 10 que depen de la quantitat d'informació que es vulgui donar al LLM. Finalment, tota aquesta informació és enviada al LLM perquè pugui generar una resposta basada en aquest coneixement.

Aquests sistemes RAG només realitzen una iteració d'aquest cicle i, per tant, només fan una consulta a aquesta base de dades. Si la informació obtinguda no és rellevant, la resposta del LLM tampoc ho serà. També, a vegades, és necessari reformular la consulta de l'usuari perquè s'adapti millor a l'estructura de la informació que s'està buscant. Si es realitza una consulta mitjançant la pregunta "Qui va ser el president d'Estats Units l'any 2022?", és convenient, degut al to afirmatiu que habitualment s'utilitza en la informació guardada en aquestes bases de dades, reformular aquesta pregunta a "Nom del president d'Estats Units l'any 2022". D'aquesta forma, es minimitza el risc de perdre informació rellevant [22].

2.4.2 Agentic RAG

Per solucionar aquestes problemes esmentats en aquest darrer paràgraf, es poden aprofitar les característiques dels agents per integrar amb facilitat aquests sistemes RAG i potenciar el seu rendiment.

Això es pot implementar dotant l'agent d'una eina per realitzar cerques en aquesta base de dades. Juntament amb les altres eines de què disposa, decideix si és convenient recuperar aquesta informació o executar altres funcions. A més, si s'utilitzen *multi-step agents*, l'agent pot valorar la informació recuperada en cada pas i, en cas necessari, tornar a realitzar una altra cerca reformulant la consulta inicial. Així, es realitzen varies iteracions d'aquest cicle de recuperació d'informació i s'elimina la rigidesa del flux d'execució del RAG tradicional.

Les avantatges que aporta aquesta integració són més que evidents. Al incloure raonament i observació en aquest procés de recuperació d'informació les respostes que s'obtenen són molt més rellevants permetent a l'agent contrastar diverses fonts.

3 Anàlisi dels recursos actuals

Un cop introduïts aquests conceptes, que han estat fonamentals a l'hora de dissenyar l'agent d'intel·ligència artificial, és necessari escollir les eines que permetran la seva implementació. Abans, però, s'ha d'analitzar l'estat actual d'aquestes i observar les seves avantatges i inconvenients per tal de fer una tria informada. Aquest anàlisi es divideix en dues parts: *frameworks* per a la creació d'agents i obtenció de dades de Fórmula 1.

3.1 Requeriments

Abans de realitzar aquest anàlisi, cal establir uns requeriments per poder comparar adequadament les diferents opcions. Aquests, es poden dividir en dos tipus, requeriments de disseny i requeriments de funcionalitat.

Els requeriments de disseny són aquells relatius al funcionament intern de l'agent i estan molt relacionats amb el contingut vist al marc teòric:

- **REQ1:** El flux d'execució ha de seguir el mètode *ReAct* i implementar el cicle *Thought-Action-Observation*.
- **REQ2:** En l'etapa d'acció, aquesta ha de ser generada en blocs de codi de Python.
- **REQ3:** El disseny de l'agent ha de ser escalable, modular i robust.
- **REQ4:** El disseny de l'agent ha d'incloure una base de coneixement d'on es pugui obtenir informació relativa a la F1.
- **REQ5:** El disseny de l'agent ha de permetre la generació de gràfiques a partir de les dades obtingudes.
- **REQ6:** El disseny de les eines de l'agent ha de permetre el *human-in-the-loop* per, en cas necessari, poder obtenir informació addicional i adaptar-se dinàmicament a les consultes.

D'altra banda, els requeriments funcionals són aquells relatius a les capacitats de l'agent i estan relacionats amb la consulta i la visualització de les dades:

- **REQ7:** Únicament es poden respondre consultes relatives a la Fórmula 1.
- **REQ8:** L'agent ha de recordar interaccions prèvies dins d'una mateixa execució.
- **REQ9:** L'agent ha de tenir la capacitat per recuperar dades tals com els resultats d'una sessió (posició, volta ràpida, etc.), les voltes de cada pilot en una sessió determinada (incloent el neumàtic utilitzat) i la telemetria associada a la seva volta ràpida.

- **REQ10:** L'agent ha de tenir la capacitat per recuperar dades que puguin donar context als resultats d'una carrera, com les penalitzacions rebudes o les millores introduïdes per cada equip en un gran premi.
- **REQ11:** L'agent ha de tenir la capacitat de donar informació d'àmbit general, com la biografia d'un pilot o les dades d'un circuit.
- **REQ12:** L'agent ha de tenir la capacitat de generar gràfiques a partir de les dades obtingudes i realitzar anàlisis detallats d'aquestes.

Aquests requeriments formen una base sòlida per guiar el disseny de l'agent i estableixen les funcionalitats que ha d'incorporar.

3.2 Frameworks per a la creació d'agents

Degut a l'aparició dels LLMs, actualment existeixen un gran nombre de *frameworks* orientats al desenvolupament d'agents d'intel·ligència artificial. Seleccionar-ne un que permeti implementar amb facilitat el disseny realitzat és clau per entendre amb claredat el seu funcionament.

La corba d'aprenentatge d'aquests *frameworks*, sol ser, en general, bastant elevada i, degut al temps limitat d'aquest projecte, s'ha volgut escollir un que sigui fàcil d'aprendre.

En aquest context, entra *smolagents*[29], un *framework* introduït a principis de 2025, que, degut a la seva simplicitat i a la bona documentació recent, és ideal per introduir-se en el món del desenvolupament d'agents d'intel·ligència artificial.

Aquest *framework*, està orientat a aquest nou enfocament d'agents que s'han introduït en el marc teòric que escriuen les seves accions en codi (*Code Agents*) (que no és el mateix que els agents utilitzats per escriure codi). Gràcies a això i que integren un executor de Python, fan que es simplifiqui molt la implementació d'agents. Aquesta simplicitat, però, té un cost. Només és útil per implementar agents senzills i que requereixen pocs recursos. Quan la lògica del sistema és més complexa, altres *frameworks* com *LangGraph*[30] poden funcionar millor.

Després d'un estudi de les característiques de *smolagents* i dels *Code Agents* es pot analitzar si els requeriments de disseny es poden assolir amb aquestes tecnologies:

Requeriment	Es pot assolir?
REQ1	Forma part del funcionament dels <i>Code Agents</i>
REQ2	Forma part del funcionament dels <i>Code Agents</i>
REQ3	Pot ser modular, robust, però no escalable (o fins a cert punt)
REQ4	Permet la integració de RAG mitjançant les eines (<i>tools</i>)
REQ5	Mitjançant les eines dissenyades o generant el codi directament
REQ6	No es pot implementar directament però es pot recrear

Taula 1: Anàlisi dels requeriments de disseny. Font: pròpia

Observem que, com s’ha vist al marc teòric, els dos primers requeriments formen part de la naturalesa dels *Code Agents*. El tercer requeriment es compleix per la part de ser modular, gràcies a les eines i el suport per sistemes multiagent, i robust, degut a la implementació del cicle *TAO*, però falla en l’escalabilitat. Això es degut a la falta de control sobre el flux de l’aplicació degut al gran nivell d’agència d’aquests tipus d’agents.

3.3 Obtenció de dades de la Fórmula 1

Actualment, per accedir a les dades generades pels cotxes durant una sessió de Fórmula 1, hi ha diverses APIs que les proporcionen. Per tal d’escollir una adequada als requeriments del projecte, aquesta ha de poder retornar per cada sessió d’un gran premi (carrera, classificació o entrenaments lliures) les següents dades:

- Els resultats finals incloent el nom del pilot, la seva posició i els punts aconseguits
- Totes les voltes d’un pilot determinat incloent informació com els temps per sector o el neumàtic utilitzat.
- La telemetria associada a una volta qualsevol incloent la velocitat, les revolucions del motor, el percentatge d’acceleració, etc.

Entre totes les APIs que existeixen, s’han valorat dues que compleixen aquests requisits (*OpenF1* [31] i *FastF1* [32]) però, finalment, s’ha decidit utilitzar *FastF1* per la seva integració en una llibreria de Python, facilitant les consultes, i pel seu sistema de *cache* que permet guardar les dades consultades de forma local, minimitzant així el temps de resposta en cas de consultes futures.

En el cas de les dades utilitzades per donar context a les respostes, s’han volgut adoptar dues estratègies diferents.

D’una banda, per obtenir la informació relativa a les penalitzacions dels pilots i les millores introduïdes per cada equip en un gran premi, s’aprofitarà el fet de que els equips estan obligats a donar aquesta informació i es publica, juntament amb aquestes penalitzacions, a la pàgina web de la FIA (Federació Internacional de l’Automòbil) [27].

D'altra banda, per obtenir informació d'àmbit general, com la biografia d'un pilot o la informació d'un circuit, la millor opció és dotar a l'agent d'una eina per poder realitzar cerques en internet.

Amb aquestes tres fonts d'informació, es pot establir una base de coneixement sòlida per tal d'implementar un sistema de RAG agent (*Agentic RAG*) on, depenent de la consulta, es recuperi informació d'una font o d'una altra. Utilitzant aquestes dades, en cas de que sigui convenient, l'agent serà capaç de generar gràfiques utilitzant la generació de codi del *Code Agent*.

4 Disseny i implementació del sistema multiagent

En aquest apartat es detalla el procés de disseny de l'arquitectura de l'agent. Posteriorment, s'analitzen les diferents eines i llibreries utilitzades per la gestió i la visualització de les dades, així com també es mostra la integració del *Code Agent* de *smolagents*. Una vegada vist això, es procedeix amb els dissenys de les eines dels agents que permetran la comunicació d'aquests amb els sistemes d'obtenció i visualització de les dades. Seguidament, es justifica la tria del LLM i del *system prompt* utilitzat i es presenta un canvi d'arquitectura degut a problemes trobats durant la implementació. Finalment, es mostra la connexió del sistema amb una interfície gràfica.

4.1 Disseny de l'arquitectura

Una vegada establerts els requeriments de l'agent i analitzades les capacitats de creació d'agents de *smolagents*, es considera la següent arquitectura:

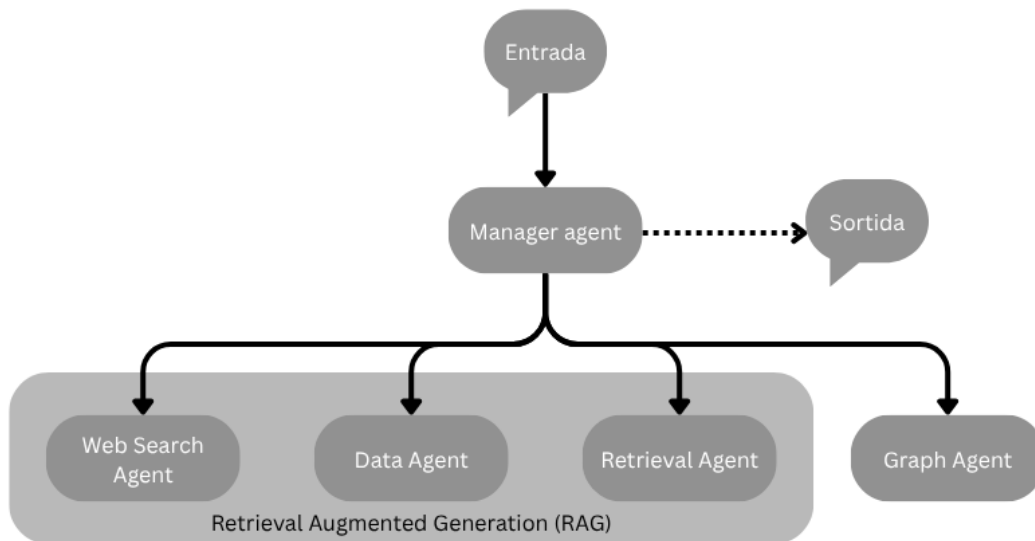


Figura 13: Arquitectura inicial del sistema multiagent. Font: pròpia

Es pot observar com aquesta segueix una arquitectura multiagent centralitzada, de tal forma que existeix un agent supervisor que coordina tots els altres agents.

S'ha considerat aquesta arquitectura davant d'altres ja que la naturalesa del agent que es vol implementar implica la coordinació entre diferents parts d'aquest. A més, aquesta comunicació entre els agents, és necessari fer-la de forma seqüencial ja que, per generar una gràfica, primer s'han de recuperar les dades. És per això que, per controlar aquest flux, l'existència d'un intermediari entre els agents és essencial pel garantir un bon funcionament.

Ara, s'analitzarà amb més detall cada part d'aquesta arquitectura:

- **Web Search Agent:** És l'encarregat de realitzar consultes de Fórmula 1 d'àmbit general a internet, això ho fa mitjançant una connexió a la API de *Serper* [33] que fa consultes a Google.
- **Data Agent:** És l'encarregat d'obtenir les dades generades pels cotxes en les diferents sessions d'un gran premi. Això ho fa integrant una connexió amb la API *FastF1* que s'ha mencionat prèviament.
- **Retrieval Agent:** És l'encarregat de recuperar la informació dels documents de la FIA relatius a les penalitzacions dels pilots i les millores dels cotxes. Aquests documents es guarden en una base de dades vectorial.
- **Graph Agent:** És l'encarregat de generar les gràfiques a partir de dades de la Fórmula 1. Això ho fa utilitzant algunes eines per generar gràfiques predefinides o aprofitant la seva capacitat per generar codi.
- **Manager Agent:** És l'agent que interactua amb l'usuari. S'encarrega de coordinar l'execució i la comunicació dels diferents agents per poder respondre adequadament una consulta qualsevol.

Seguint aquesta arquitectura, es pot veure la separació de responsabilitats entre els diferents agents, complint així el requisit de la modularitat. També, dona facilitat a l'hora d'integrar nous agents amb altres funcionalitats addicionals, aportant escalabilitat. Finalment, degut a les varies fonts d'informació existents, en cas d'una cerca fallida, es poden complementar entre ells, donant robustesa.

Una vegada fet aquest anàlisi, es pot veure si es compleixen els requeriments de disseny:

Requeriments	REQ1	REQ2	REQ3	REQ4	REQ5	REQ6
Compliment	Sí	Sí	Sí	Sí	Sí	-

Taula 2: Comprovació del compliment dels requeriments de disseny. Font: pròpia

El requeriment 6, que establia que el disseny de les eines havia de permetre el *human-in-the-loop*, es veurà en el apartat 4.6.

4.2 Anàlisi de les llibreries utilitzades

En aquest apartat, es veurà el funcionament i les capacitats de les llibreries utilitzades en la implementació dels diferents agents que formen el sistema. Mitjançant aquestes llibreries, es podran dissenyar les eines de cadascun dels agents de forma eficient i establir l'arquitectura vista.

4.2.1 Smolagents

Com s'ha explicat a l'apartat 3.2, s'ha utilitzat aquest *framework* per a la creació del sistema multiagent. Cada agent d'aquest sistema és un *Code Agent* que està

implementat en aquesta llibreria. Aquest tipus d'agent està dissenyat de tal forma que integra totes les característiques de funcionament vistes en el apartat 2.2.3 (*ReAct*, accions en codi, etc.). Per entendre com ho fa, es pot observar el seu flux d'execució en la següent figura:

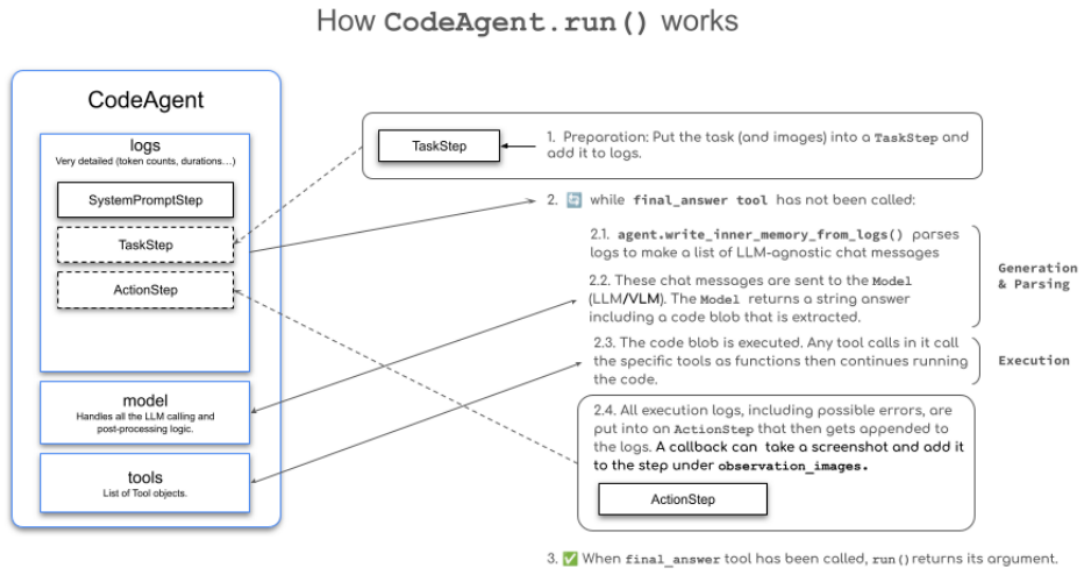


Figura 14: Funcionament del *Code Agent* de *smolagents*. Font: [26]

Primerament, es guarda el *system prompt* en el `SystemPromptStep` i la consulta inicial en el `TaskStep` i s'afegeix tot als *logs* de l'agent, és a dir, a la seva memòria. Posteriorment, s'executa un bucle que no termina fins que es retorna una resposta final mitjançant la `final_answer` tool (es pot definir un nombre màxim d'iteracions per tal d'evitar bucles infinits).

Dins d'aquest bucle, es comproven els *logs* i es transformen a una llista de missatges que el LLM pot entendre. S'envien aquests missatges a aquest LLM i retorna una resposta contenen un string on s'inclou un bloc de codi de Python que s'extreu per ser executat.

Cada agent disposa del seu propi entorn d'execució de Python contenint les mínimes llibreries possibles per garantir la seguretat i evitar l'execució de codi perillós generat pel LLM. Segons les funcionalitats de cada agent, es poden importar altres llibreries que es creuin convenientes (p.ex. *pandas* o *matplotlib.pyplot* per l'agent generador de gràfiques).

Durant l'execució del codi, en cas d'incloure crides a eines, aquestes s'executen com a funcions normals. Tots els *logs* generats durant aquesta execució, s'inclouen en una `ActionStep` que, posteriorment, s'incorpora a la memòria de l'agent per poder iniciar una altra iteració del bucle.

Per tal d'aprofitar al màxim aquest disseny, s'ha de complementar amb un *system prompt* molt detallat indicant el comportament que ha de tenir el LLM. A continuació és mostra un petit fragment d'aquest:

System prompt del Code Agent de smolagents [29]

You are an expert assistant who can solve any task using code blobs. You will be given a task to solve as best you can. To do so, you have been given access to a list of tools: these tools are basically Python functions which you can call with code. To solve the task, you must plan forward to proceed in a series of steps, in a cycle of 'Thought:', 'Code:', and 'Observation:' sequences. At each step, in the 'Thought:' sequence, you should first explain your reasoning towards solving the task and the tools that you want to use. Then in the 'Code:' sequence, you should write the code in simple Python. The code sequence must end with '<end_code>' sequence. During each intermediate step, you can use 'print()' to save whatever important information you will then need. These print outputs will then appear in the 'Observation:' field, which will be available as input for the next step. In the end you have to return a final answer using the 'final_answer' tool.

Posteriorment, seguint la estratègia de *few-shot prompting* s'inclouen varis exemples de com hauria de ser una execució:

Exemple d'execució en el system prompt [29]

```
Task: "What is the result of the following operation: 5 + 3 + 1294.678?"
Thought: I will use python code to compute the result of the operation and
then return the final answer using the 'final_answer' tool
Code:
"py
result = 5 + 3 + 1294.678
final_answer(result)
"<end_code>
```

A part d'aquests exemples, també s'inclouen una sèrie de regles per intentar controlar el comportament del LLM i evitar errors d'execució. Per exemple, algunes d'aquestes regles són 'utilitza només variables que has definit' o 'no li donis el nom d'una eina a una variable'.

Finalment, per donar informació sobre les eines que pot utilitzar o els agents que pot cridar per delegar una tasca, s'inclou també una descripció detallada obtinguda a partir del seu *docstring*, en el cas de les eines, o de la seva descripció, en el cas de l'agent (obligatòria quan es crea). Donant aquesta 'visibilitat' dels agents que es poden cridar, es pot crear una arquitectura com la del disseny.

4.2.2 FastF1

Aquesta llibreria de Python dona accés a una gran quantitat de dades relatives a la Fórmula 1 tals com el cronometratge de les voltes, la telemetria dels cotxes, la posició al circuit, informació del neumàtics, informació del temps, resultats d'una sessió, etc. Tota aquesta informació la dona a través de *DataFrames* de la llibreria

Pandas [34] dotats de funcions especials per facilitar l'accés a les dades. Gràcies a això, es poden dissenyar les eines del Data Agent, fent modificacions, si cal, als *DataFrames* obtinguts mitjançant les consultes a la API.

També, disposa d'una integració amb la llibreria *matplotlib*[35] que facilita la visualització de les dades. Aquesta funcionalitat, però, només es fa servir en els gràfics predeterminats que inclou el Graph Agent com a eines.

Aquesta llibreria també disposa d'un sistema de *cache* que permet reduir considerablement el temps de les consultes a la API. Aquesta funcionalitat és clau per optimitzar el funcionament de l'agent i millorar els temps de resposta.

4.2.3 LangChain

Per tal de crear la base de dades vectorial del Retrieval Agent, s'ha fet servir una sèrie d'eines definides a la llibreria *LangChain Community* [36]. En aquesta, s'inclou una multitud d'integracions fetes per tercers que la permeten implementar amb facilitat. Les que es faran servir seran:

- **HuggingFaceEmbeddings:** Per a obtenir un model que generi els *embeddings* de cada fragment de text.
- **RecursiveCharacterTextSplitter:** Per poder fragmentar els documents seguint l'algorisme de fragmentació recursiva.
- **VectorStores:** Per poder guardar els *embeddings* generats pel model.
- **FAISS:** Per poder realitzar una cerca eficient per similitud en la base de dades vectorial.

4.2.4 Tractament i visualització de les dades

Com s'ha comentat anteriorment, en el disseny de les eines del Data Agent es processen les dades obtingudes utilitzant una de les llibreries de Python més potents actualment pel tractament de dades, *pandas*[34].

Pot succeir, però, que aquest agent, una vegada recuperades les dades, necessiti realitzar operacions addicionals per tal d'adaptar el seu resultat a la consulta. És per això, que també té la capacitat d'utilitzar aquesta llibreria de forma independent.

El mateix passa en el cas del Graph Agent. Aquest, disposa d'unes eines que generen les gràfiques utilitzant la llibreria *matplotlib* [35] però, en el cas de que no disposi d'una eina per generar una gràfica determinada, pot utilitzar aquesta llibreria per generar ell mateix el codi corresponent.

4.3 Disseny de les eines

En aquest apartat, s'estableix el disseny de les eines de cada agent, utilitzant les llibreries prèviament esmentades, en base als requeriments funcionals que s'han

definit a l'apartat 3.1. Així, es fa la següent assignació de responsabilitats:

Data Agent	Retrieval Agent	Web Search Agent	Graph Agent
REQ9	REQ10	REQ11	REQ12

Taula 3: Assignació dels requeriments funcionals a cada agent. Font: pròpia

Els requeriments 7 i 8, que fan referència al funcionament general del sistema, es veuran en els apartats 4.6 i 4.7, respectivament.

4.3.1 Data Agent

Recordant la definició del requeriment 9, aquest establia que l'agent havia de tenir la capacitat d'obtenir dades relatives als resultats d'una sessió (incloent informació com la posició o la volta més ràpida de cada pilot), a totes les seves voltes en una sessió determinada (incloent el tipus de neumàtic utilitzat) i a la telemetria associada a la volta ràpida.

Mitjançant una única crida a la API, es pot retornar un objecte de tipus *Session* que conté tota aquesta informació a la qual es pot accedir posteriorment mitjançant els mètodes integrats a la llibreria.

En base a aquests requeriments, s'han definit les següents eines:

- **get_session_results:** Donada una sessió d'un gran premi determinat (p.ex. carrera del Gran Premi d'Espanya de 2024), retorna un *pandas.DataFrame* contenint informació com el nom del pilot, l'equip al qual pertany, la posició final de carrera i classificació, el temps de la volta més ràpida o els punts obtinguts.
- **get_all_laps:** Donada una sessió d'un gran premi determinat i el nom d'un pilot, retorna un *pandas.DataFrame* contenint informació de totes les seves voltes. La informació més rellevant que es dona per cada volta és, el seu temps, així com també els temps per sector, el número de volta, el neumàtic utilitzat o la posició al final de la volta (en el cas de la carrera).
- **get_fastest_lap:** És un cas especial de la funció anterior on, per simplificar el processament de les dades per part de l'agent, que pot cometre errors, es retorna directament la volta més ràpida d'un pilot en una sessió determinada.
- **get_telemetry_fastest_lap:** Donada una sessió d'un gran premi determinat i el nom d'un pilot, retorna un *pandas.DataFrame* contenint la telemetria de la volta més ràpida. Aquesta telemetria, inclou camps com, la velocitat, les revolucions del motor (RPM), la marxa, el percentatge d'acceleració, l'ús del fre o la posició del pilot al circuit.

Es pot observar com, amb aquestes eines, definides de la forma més modular possible, separant-les perquè cadascuna tingui únicament un objectiu, es compleix el requeriment associat al Data Agent.

4.3.2 Retrieval Agent

En el cas del Retrieval Agent, és necessari preparar prèviament la base de dades contenint els documents de les penalitzacions i les millores de cada equip per cada gran premi. És fàcil veure, però, que aquest nombre de documents augmenta considerablement per cada any del que volem incloure aquesta informació. Amb una mitja d'uns 10 documents de 2 pàgines cadascun per gran premi (estimat a la baixa) i uns 24 gran premis per temporada, obtenim que, per cada any, es tenen 480 pàgines d'informació en format PDF.

Per tant, degut a la dificultat de recopilar tots aquests documents i per garantir un correcte funcionament d'aquest agent, només s'han considerat els de la temporada actual.

Per poder cercar dins d'aquesta base de dades, s'ha dividit aquesta en parts més petites que contenen els documents de cada gran premi per separat. D'aquesta forma, es pot dissenyar una eina de recuperació diferent per cada circuit, augmentant així considerablement la precisió de les respostes. Aquestes eines segueixen la següent estructura:

- **f1_{race}_grand_prix_info_retriever:** Cada eina té una base de dades vectorial associada amb els *embeddings* dels documents del gran premi corresponent. Quan s'utilitza, es transforma la consulta de l'usuari a *embeddings* i, mitjançant una cerca per similitud, retorna els 10 documents més semblants a aquesta.
- **f1_sporting_regulations_info_retriever:** Aquesta eina segueix el mateix funcionament però la seva base de dades vectorial conté el document del reglament esportiu de la Fórmula 1. D'aquesta forma, es pot donar més context a les penalitzacions aplicades a cada pilot.

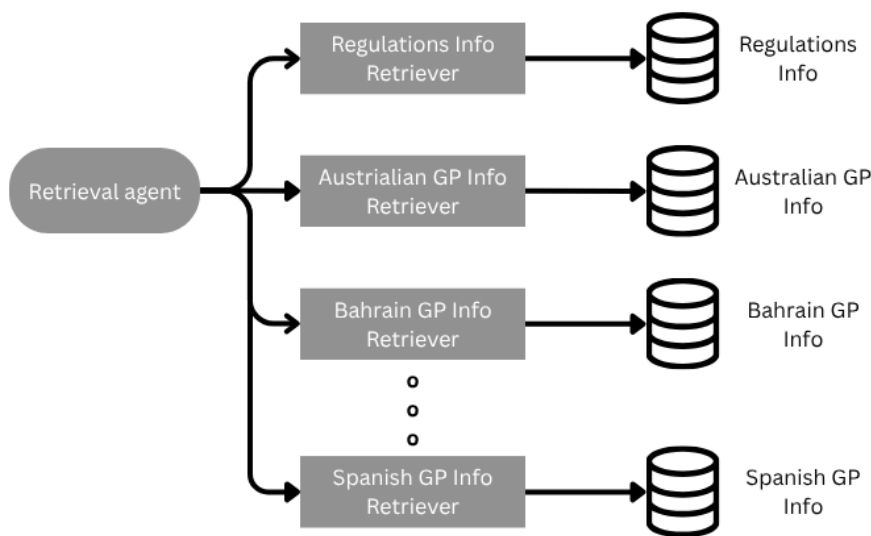


Figura 15: Esquema del disseny del Retrieval Agent. Font: pròpia

4.3.3 Web Search Agent

En aquest agent, es fa ús directament de les eines proporcionades per *smolagents* per a realitzar cerques en internet. Aquestes són les següents:

- **GoogleSearchTool:** Donada una consulta, utilitza la API de *Serper* per realitzar una cerca a Google mitjançant un **GET** a la direcció *https://google.serper.dev/search*. Per cada consulta, obté un string dels resultats més rellevants amb la seva URL i una petita descripció.
- **VisitWebpageTool:** Aquesta eina, donada una URL d'una pàgina web, la visita i retorna els seus continguts en un string.

Gràcies a aquest agent, el sistema és capaç de recuperar informació a la que no té accés mitjançant el Data Agent o el Retrieval Agent. Aquesta, generalment, hauria de ser d'àmbit general i accessible a través d'una cerca d'aquest tipus.

Per exemple, es pot utilitzar per obtenir la biografia d'un pilot, informació d'un circuit o la classificació de punts general d'una temporada per tal de donar context als resultats d'una carrera. En cas de buscar informació més específica, com la telemetria dels cotxes o quin temps va fer Fernando Alonso a la volta 33 del Gran Premi d'Espanya de 2024, no la trobarà.

4.3.4 Graph Agent

Teòricament, en aquest agent no s'hauria de dissenyar cap eina generadora de gràfiques ja que l'objectiu definitiu d'aquest sistema es que, a partir de les dades recuperades dels altres agents i en base a la consulta de l'usuari, es creïn les gràfiques automàticament gràcies a la generació i execució de codi dels *Code Agents*. L'única 'eina' que se li hauria de donar és l'accés a les llibreries per poder tractar i visualitzar aquestes dades.

Això, però, no funciona molt bé si es busquen visualitzacions molt concretes d'aquestes dades o que requereixen d'un tractament avançat d'aquestes. És per això, que s'han afegit algunes eines per poder generar gràfiques interessants:

- **plot_fastest_lap_telemetry:** Donada una sessió qualsevol d'un gran premi, genera una gràfica comparant la telemetria de la volta ràpida dels pilots passats per paràmetre. Aquesta funció admet múltiples entrades de pilots (des de 1 fins a 20) i també permet modificar els canals de telemetria visualitzats (velocitat, revolucions, marxa, percentatge d'acceleració, ús del fre i ús del DRS (*Drag Reduction System*) segons la consulta de l'usuari. La seva sortida és un string contenenent el nom de l'arxiu on s'ha guardat el gràfic.

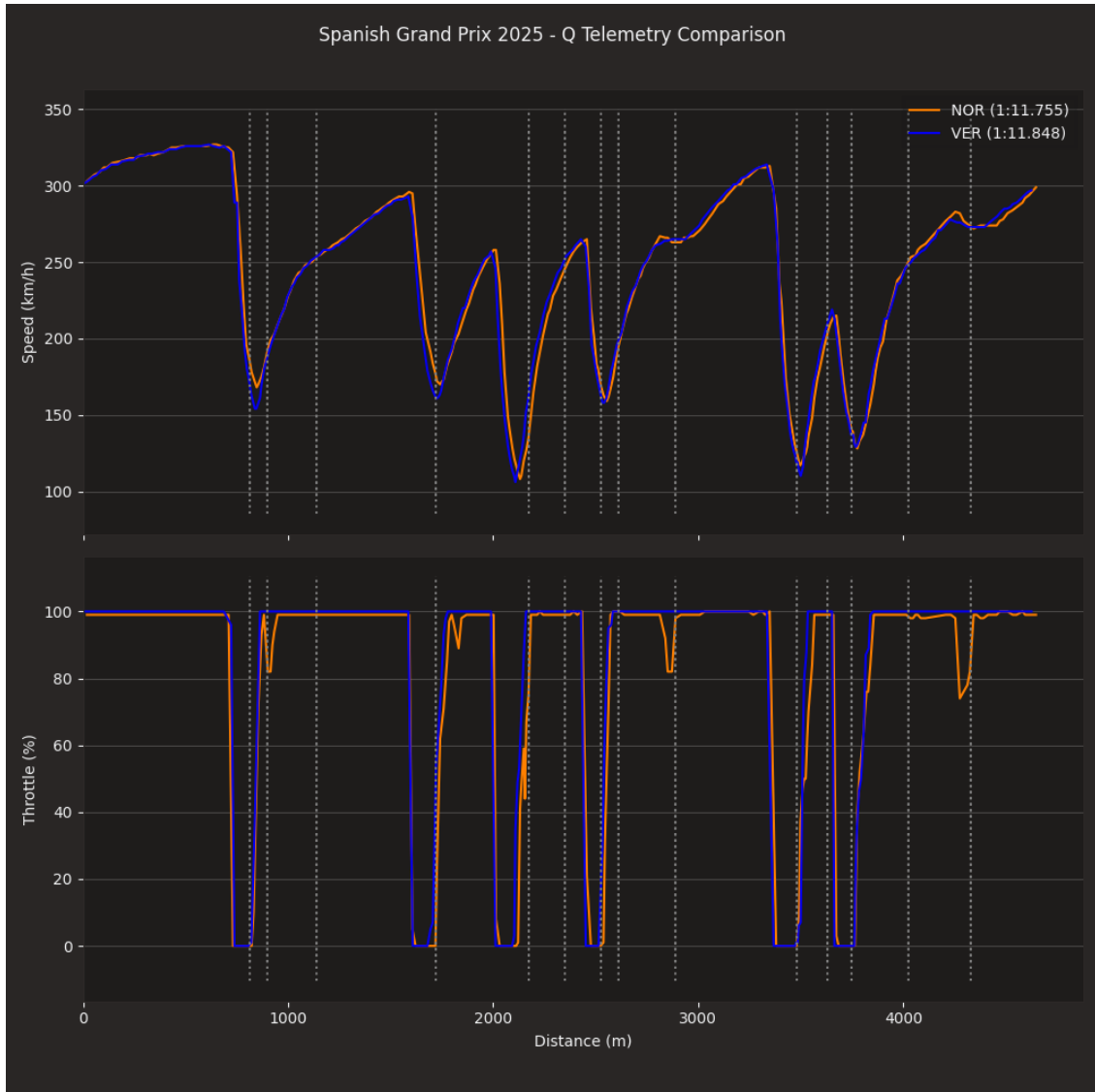


Figura 16: Comparació de telemetria entre dos pilots incloent velocitat i percentatge d'acceleració. Font: pròpia

- **plot_race_strategy:** Donat un gran premi, genera una gràfica de barres mostrant l'estratègia de cada pilot en la carrera. Aquesta barra mostra els neumàtics utilitzats en cada volta pintant-los amb el color associat. També mostra, mitjançant una línia vertical a la barra, quan es produeix un canvi de neumàtics. Igual que en les eines prèvies, es pot seleccionar els pilots que apareixen a la gràfica. Retorna un string contenent el nom de l'arxiu on s'ha guardat el gràfic.

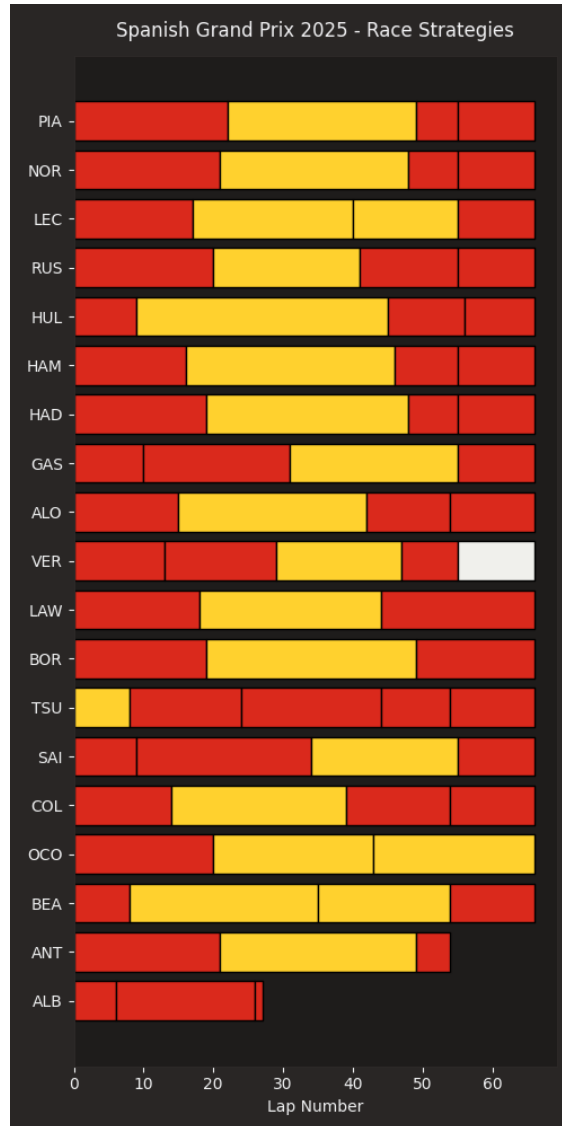


Figura 17: Comparació de la estratègia de carrera entre tots els pilots. Font: pròpia

- **plot_race_pace:** Donat un gran premi, genera una gràfica tipus *boxplot* analitzant el ritme de carrera de cada pilot ordenant-los segons el temps mig per volta. Es poden variar els pilots que apareixen a la gràfica segons la consulta de l'usuari. Si no es passa cap pilot per paràmetre, es mostren tots. De la mateixa forma, la sortida és un string amb el nom del arxiu on s'ha guardat el gràfic.

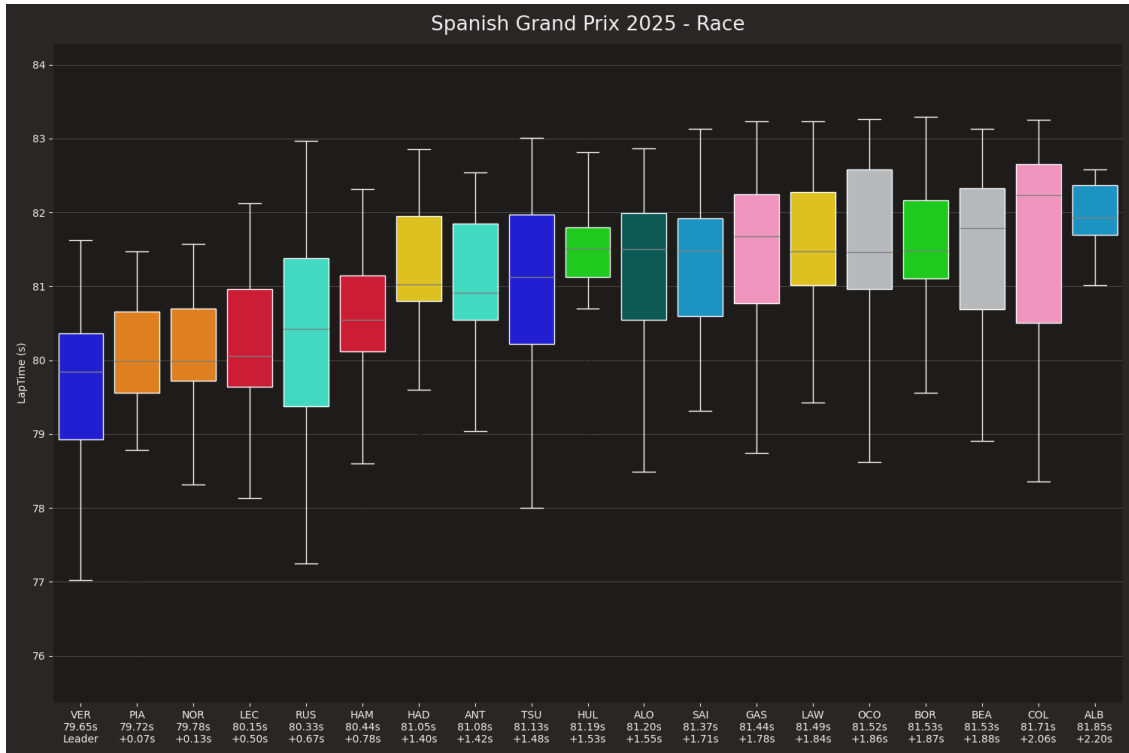


Figura 18: Comparació del ritme de carrera entre tots els pilots. Font: pròpia

Aquestes eines són una 'ajuda' a l'agent per tal de que, quan hagi de generar gràfiques relacionades, les utilitzi directament. S'han realitzat aquestes visualitzacions perquè aporten una gran quantitat d'informació per entendre el desenvolupament d'una carrera i poder comparar directament el rendiment dels cotxes i dels pilots. Aquestes, com s'ha esmentat, requereixen d'un tractament complex de les dades que l'agent no és capaç de reproduir amb consistència i, per tant, s'ha volgut implementar-les manualment.

Nota: Això no significa, però, que l'agent no sigui capaç de generar gràfiques interessants; aquestes es veuran en el apartat 5 i en l'annex.

Com s'ha explicat en la seva definició, aquestes gràfiques admeten canvis segons la consulta de l'usuari. Així, es poden adaptar millor a aquestes mantenint l'estil de visualització.

Finalment, per complir el requeriment associat, ha de ser capaç d'analitzar aquestes gràfiques generades i poder descriure el seu contingut. Per fer-ho, s'ha dissenyat una eina addicional:

- **analise_graph:** Donat un gràfic, aquest es passa al LLM juntament amb la consulta "Analitza aquesta imatge que conté un gràfic relacionat amb la F1", per a que generi una resposta basant-se en aquest.

Amb la definició d'aquestes eines, es compleixen els requeriments 9, 10, 11 i 12, associats a aquests agents. El requeriment 10, però, es compleix amb la limitació de que els documents només estan disponibles per la temporada actual.

4.4 Tria del LLM

Definida l'arquitectura del sistema multiagent i de les eines de cadascun d'ells, el següent pas es escollir un LLM amb la capacitat de gestionar tot aquest sistema. Aquest LLM ha de complir una sèrie de requeriments per poder garantir un bon funcionament:

- Ha de tenir una gran capacitat de raonament per tal de coordinar-se adequadament amb els altres agents.
- Ha de tenir una gran capacitat per a la generació de codi per poder integrar-se de forma eficient amb els *Code Agents* i crear gràfiques amb facilitat.
- Ha de permetre imatges com a entrada per poder utilitzar la eina per analitzar gràfiques.
- Ha de ser barat (preferiblement per sota del dòlar per milió de tokens).
- Ha de poder processar una gran quantitat de tokens a la vegada (context) degut a les múltiples passes que segueixen els agents per respondre una consulta.

Després de considerar i provar alguns models de *MetaAI*[37], *DeepSeek*[38], *Mistral*[39] i *OpenAI*[40], finalment, es va triar un model d'aquest últim, GPT-4o-mini. Aquest model, durant l'etapa de desenvolupament, ha sigut el que ha funcionat de forma més consistent, complint a la perfecció amb tots els requisits i sent relativament econòmic (0,15\$/milió de tokens d'entrada, 0,6\$/milió de tokens de sortida)[41].

4.5 Problemes trobats i canvi d'arquitectura

Durant la fase de desenvolupament, es va trobar un problema relatiu a la comunicació entre els agents que comprometia el funcionament del sistema. Aquests agents, són cridats com a 'funcions' en els blocs de codi generats pel Manager Agent i, per tant, la seva sortida, que serà un string contenint tota la informació donada amb la *final_answer tool* es guarda en una variable de Python.

El problema es que, aquesta informació, en el cas del Data Agent, eren *DataFrames* contenint una gran quantitat de dades que, al generar una resposta final per enviar-li al Manager Agent, el LLM no les incloïa totes i es produïa una pèrdua de dades.

Pel mateix motiu, aquesta pèrdua també es produïa en el moment de passar les dades del Manager Agent al Graph Agent. És per això, que es va plantejar un canvi en l'arquitectura del sistema i en el disseny de les eines del Data Agent.

Aquest nou enfocament, es va basar en la conversió d'aquests *DataFrames* en arxius *parquet*, que serveixen per emmagatzemar de forma molt eficient aquestes dades. D'aquesta forma, retornant com a sortida de les eines del Data Agent el

nom d'aquests arxius, el Graph Agent pot tenir accés a aquesta informació llegint aquest fitxers. Així, es pot crear un sistema de 'memòria compartida' entre els agents simplificant la seva comunicació i evitant una sobrecàrrega de tokens al LLM.

Seguint aquesta línia, per tal de minimitzar l'intercanvi d'aquesta informació entre els agents (que es on es produïen la majoria dels errors), es va reconsiderar l'arquitectura establerta.

Com que l'objectiu principal és recuperar dades d'un sistema RAG per extreure'n conclusions d'aquestes mitjançant visualitzacions, es va establir el Graph Agent com a agent coordinador. Una alternativa equivalent, hagués sigut dotar al Manager Agent de la capacitat de processar aquestes dades i generar les gràfiques. Utilitzant aquesta nova arquitectura, es va notar un augment significatiu en la precisió i es va aconseguir reduir considerablement el nombre de tokens utilitzat.

Analitzades totes les eines de cada agent i incorporant aquest canvi d'arquitectura, obtenim el següent diagrama:

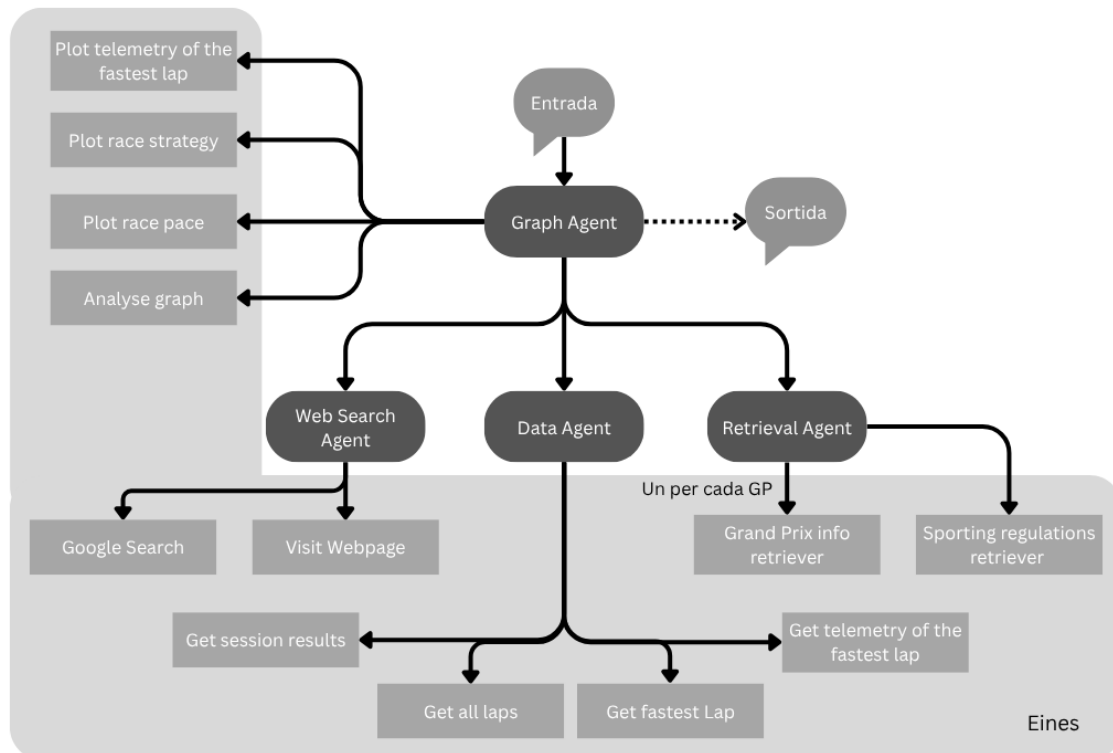


Figura 19: Arquitectura final del sistema multiagent. Font: pròpia

4.6 Elecció del system prompt

Com s'ha vist en l'apartat 4.2.1, on s'introdueix el *framework* de *smolagents*, aquest utilitza un *system prompt* molt detallat per complementar les funcionalitats dels *Code Agents*. En cas de modificar-ho, s'ha d'anar molt en compte perquè podria comprometre el seu funcionament.

Per aquest motiu, per tal d'adaptar el comportament del LLM a l'entorn dels agents amb els quals treballa, s'han afegit una sèrie de regles a continuació del *system prompt* predeterminat. Aquestes serveixen per 'guiar' o donar recomanacions al LLM del que hauria de fer en un cas determinat. Així, es poden evitar errors comuns produïts per aquests. Per exemple, en el Graph Agent s'ha afegit la frase "En cas de generar una gràfica, guarda-la mitjançant *plt.savefig()* en el mateix directori on estàs sent executat", d'aquesta forma, el LLM evita l'ús de la instrucció *plt.show()* per mostrar el gràfic i permet accedir posteriorment a aquest per analitzar-ho.

Gràcies a aquestes regles, afegint la frase "No contestis cap consulta que no estigui relacionada amb la Fórmula 1", es pot fer que el sistema "s'especialitzi" en aquest tipus de tasques, complint així amb el requeriment 7.

El mateix es pot fer pel requeriment 6 (*human-in-the-loop*), afegint la frase "Si necessites informació addicional de l'usuari per poder resoldre de forma precisa la tasca, utilitza la *final_answer tool* per preguntar-li". Si es recorda el funcionament dels *Code Agents* de *smolagents*, aquests detenien el bucle d'execució al utilitzar aquesta funció. D'aquesta manera es compleix també aquest requeriment.

No hi ha una forma establerta per seleccionar aquestes regles, depenent del sistema algunes funcionaran millor que altres. Això si, s'ha de ser sempre el més descriptiu possible i no introduir regles contradictòries que podrien confondre al LLM. Després de molta prova i error, s'han seleccionat aquelles que donen la millor precisió del sistema i que fan que el seu comportament sigui bastant previsible.

4.7 Connexió de l'agent amb una UI

Finalment, per poder interactuar amb l'agent, s'ha aprofitat la integració que té la llibreria de *smolagents* amb *Gradio*[42]. Aquesta llibreria de Python permet crear de forma molt senzilla una interfície web on fer consultes de manera interactiva a l'agent, seguint l'estil *chatbot*.

D'aquesta forma, l'agent pot utilitzar la memòria que té integrada per basar les seves respostes en consultes prèvies de l'usuari. Això es degut a que, cada cop que se li dona una tasca, aquest crida al seu mètode per iniciar la seva execució establint, mitjançant un paràmetre, que no vol reiniciar la seva memòria. Així, es compleix el requeriment 8, l'últim que quedava pendent.

Aquesta memòria, però, s'ha de controlar ja que, per cada interacció, es passa la conversa completa al LLM augmentant així considerablement el nombre de tokens per consulta i, conseqüentment, el seu preu. Per això, s'ha establert un límit de 10000 tokens d'entrada i 2000 de sortida. Aquests valors s'han fixat després de varies proves on es buscava minimitzar el cost de les consultes al mateix temps que es mantenia un context suficientment bo com per poder respondre amb precisió.

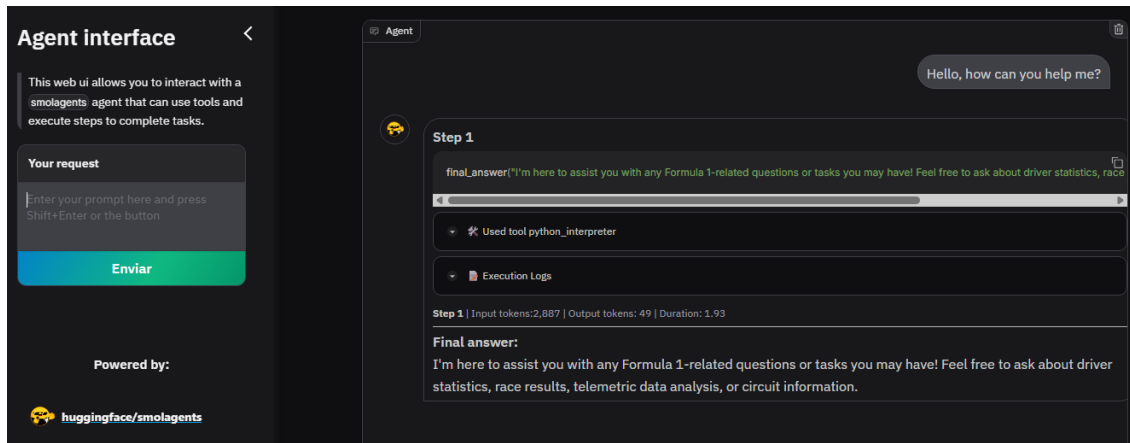


Figura 20: Gradio UI connectada al Graph Agent. Font: pròpia

Aquesta UI es connecta a l'agent principal del sistema, en aquest cas, el Graph Agent. A més de la funcionalitat bàsica d'un *chatbot*, consulta-reposta, dona la capacitat de veure els passos que s'han realitzat per arribar a la resposta final. Aprofitant això, es pot *debuggar* el comportament del sistema i avaluar el seu rendiment.

5 Resultats

En aquesta secció, es mostraran els resultats finals obtinguts després d'implementar el disseny vist a l'apartat 4.5. Per fer-ho, s'analitzaran alguns casos d'ús on es demostra el compliment dels requeriments funcionals i es veuran alguns exemples de visualitzacions que es poden aconseguir.

També, per tal d'avaluar d'una forma més detallada el funcionament del sistema, es veuran algunes mètriques interessants que permetran extreure conclusions sobre el seu rendiment.

5.1 Verificació dels requeriments

Primerament, es comprova que l'agent només respon consultes relacionades amb la Fórmula 1 (REQ7):

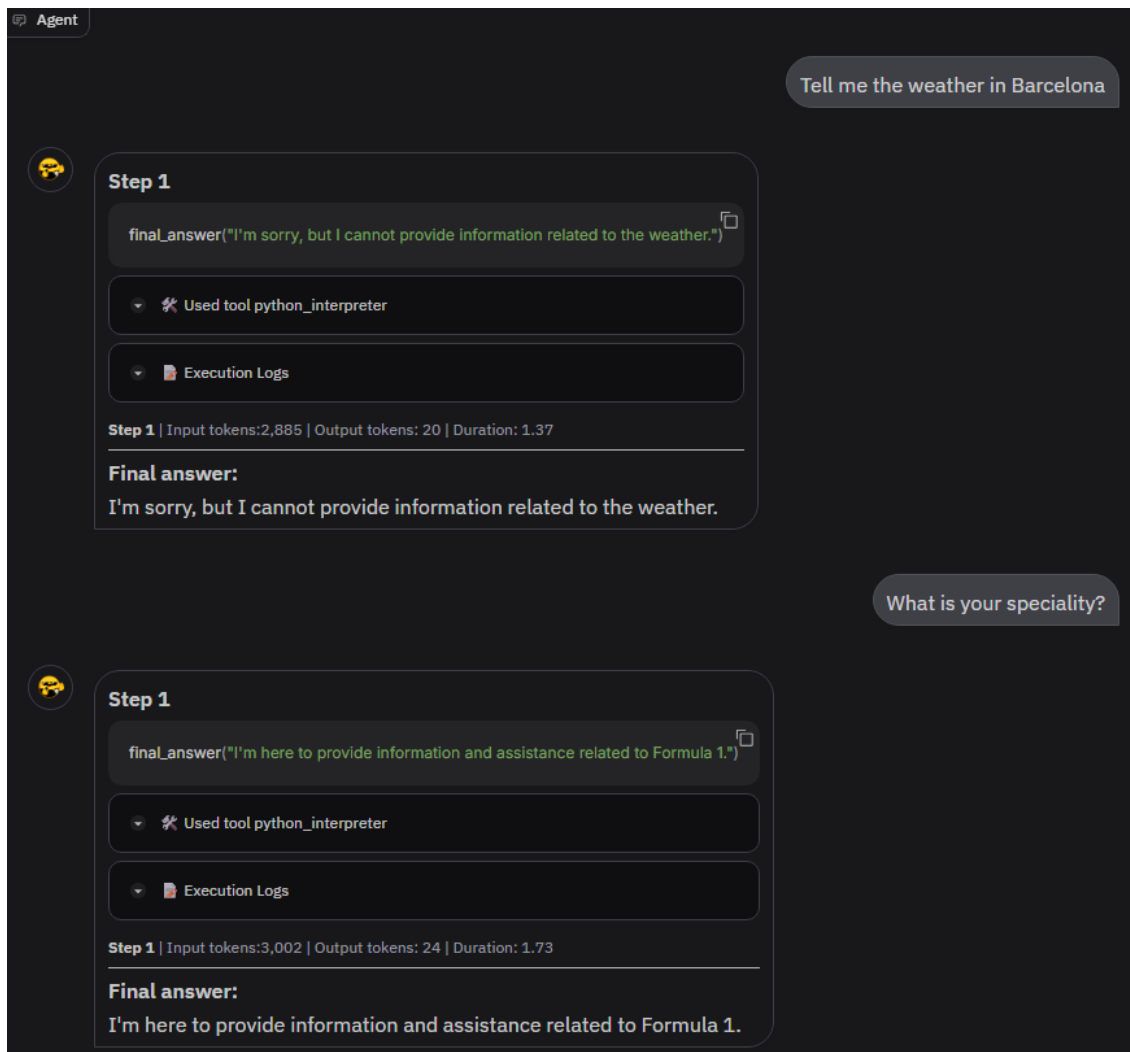


Figura 21: Verificació del requeriment 7. Font: pròpia

També, es comprova que l'agent pot recordar interaccions prèvies amb l'usuari, és a dir, té memòria (REQ8):

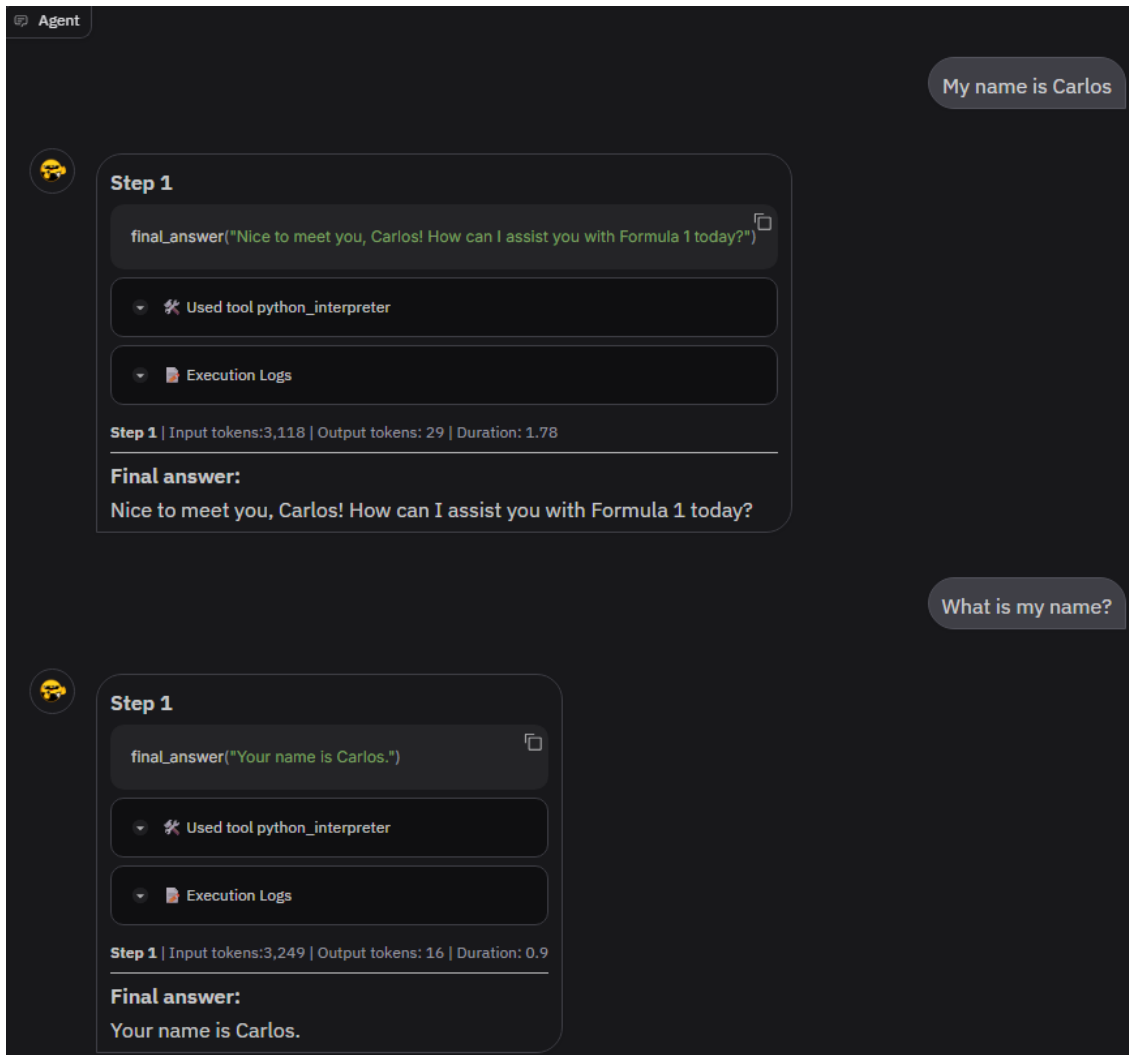


Figura 22: Verificació del requeriment 8. Font: pròpia

Com s'ha comentat a l'apartat anterior, per tal de minimitzar els costos del LLM, aquesta memòria té un límit de 10000 tokens de context. Durant les proves realitzades, aquest valor a funcionat a la perfecció ja que en cap moment s'ha notat una pèrdua de memòria per part de l'agent.

Ara, es verifica que té la capacitat de retornar informació relativa als resultats d'una sessió i que pot accedir a les voltes de cada pilot per determinar la volta més ràpida (REQ9). També, es podria demanar que inclogués la velocitat màxima d'aquesta volta, fent ús així de la telemetria.

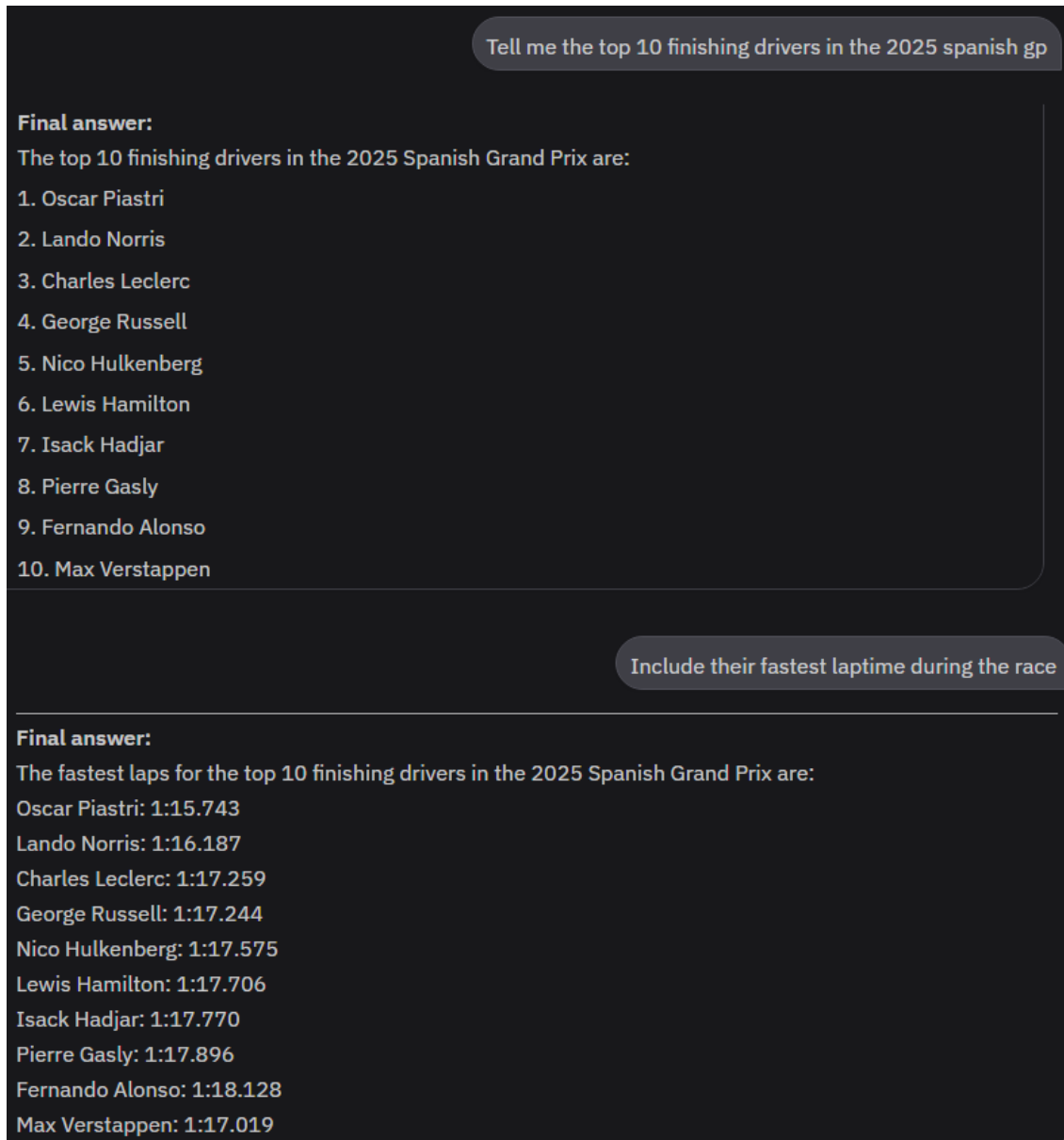


Figura 23: Verificació del requeriment 9. Font: pròpia

Tot i que aquesta funcionalitat pot ser útil per determinades consultes, el seu objectiu és ser utilitzada internament per generar les corresponents gràfiques.

Es comprova que la base de dades vectorial juntament amb el Retriever Agent funcionen adequadament realitzant algunes consultes (REQ10). Aquesta hauria de contenir informació sobre les penalitzacions de cada pilot en cada carrera de la temporada actual i les millores introduïdes per cada equip.

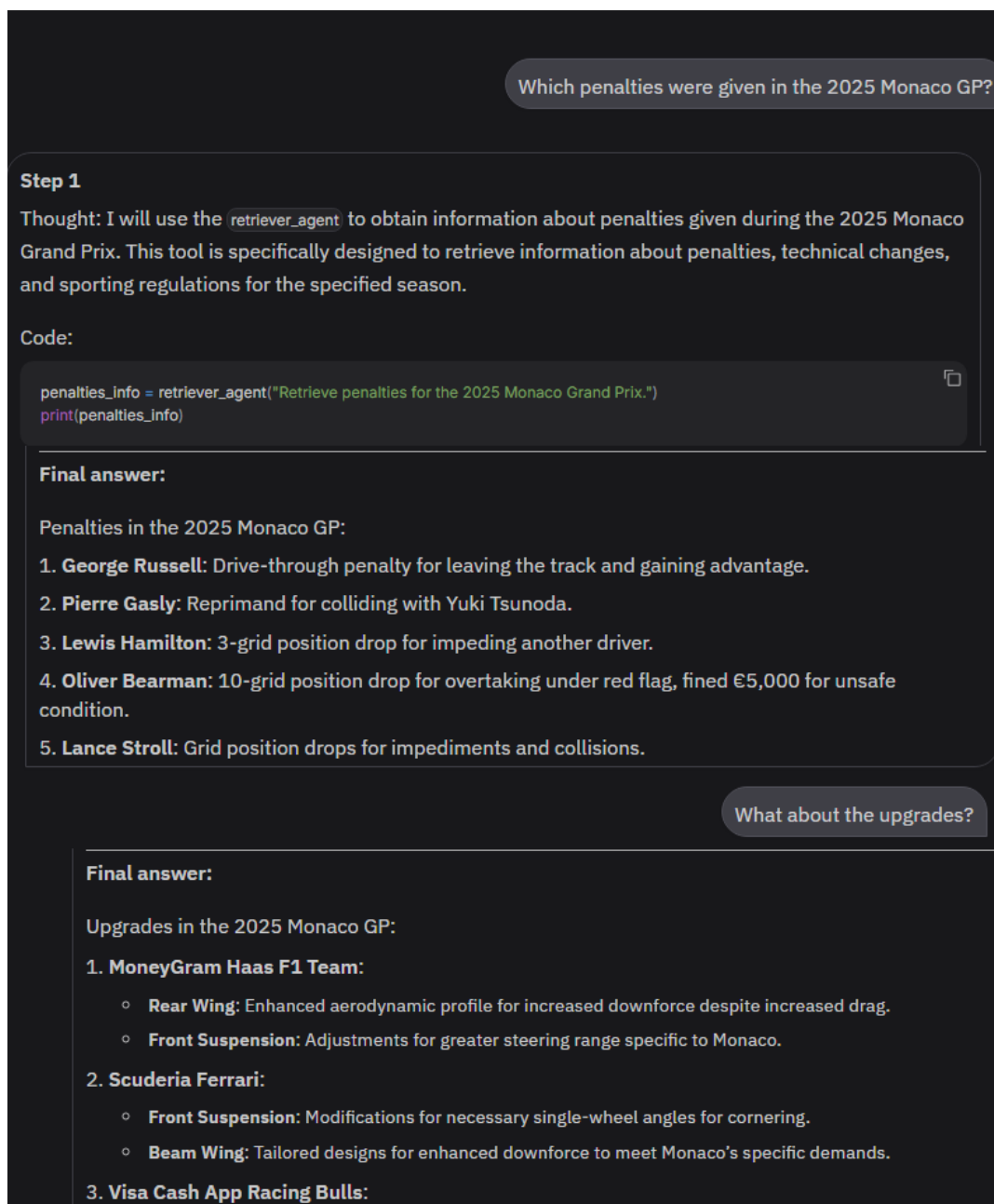


Figura 24: Verificació del requeriment 10. Font: pròpia

Tot i que pugui semblar que l'objectiu d'aquest agent és purament informatiu, també es poden generar gràfiques interessants amb aquest. Per exemple, es podria fer una comparativa del nombre de penalitzacions rebudes per cada pilot en una temporada. Així, es podria determinar quins pilots són més 'nets' que altres.

El mateix passa amb el Web Search Agent, amb aquest agent s'obre una finestra de possibilitats pràcticament il·limitada. Primer, es comprovarà que realitza correctament les cerques a internet preguntant informació sobre el Circuit de Barcelona (REQ11) i, posteriorment, es mostrarà un exemple on s'ha utilitzat per consultar la classificació general de la temporada i demanar al Graph Agent que generi una gràfica.

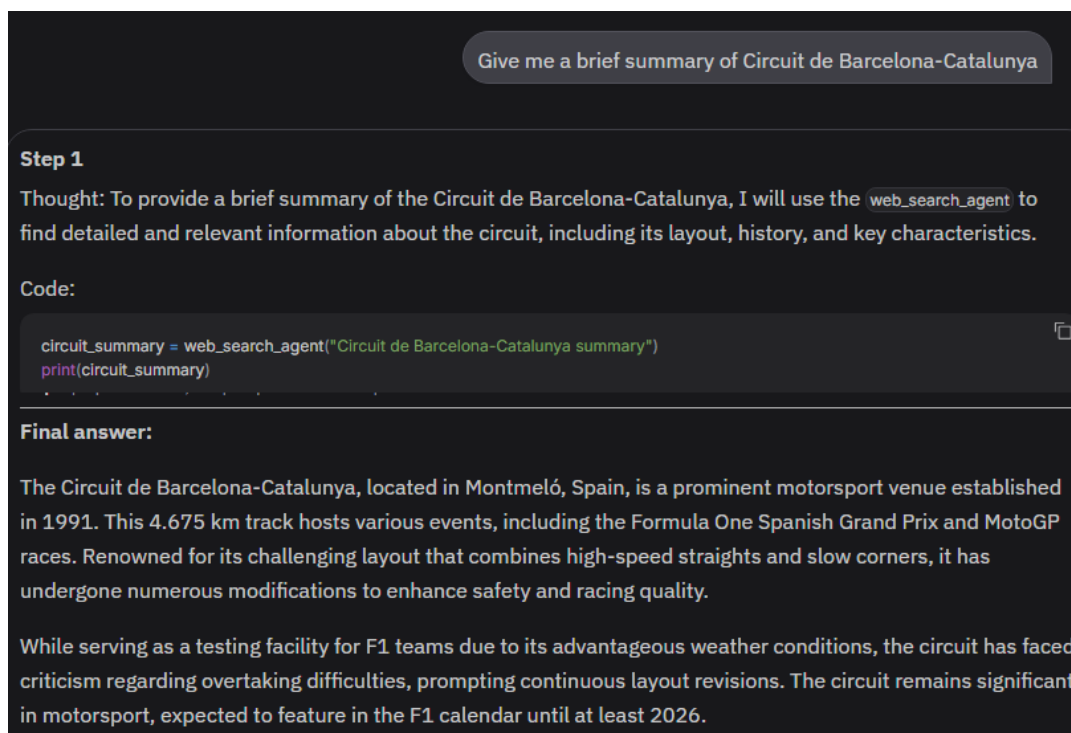


Figura 25: Verificació del requeriment 11. Font: pròpia

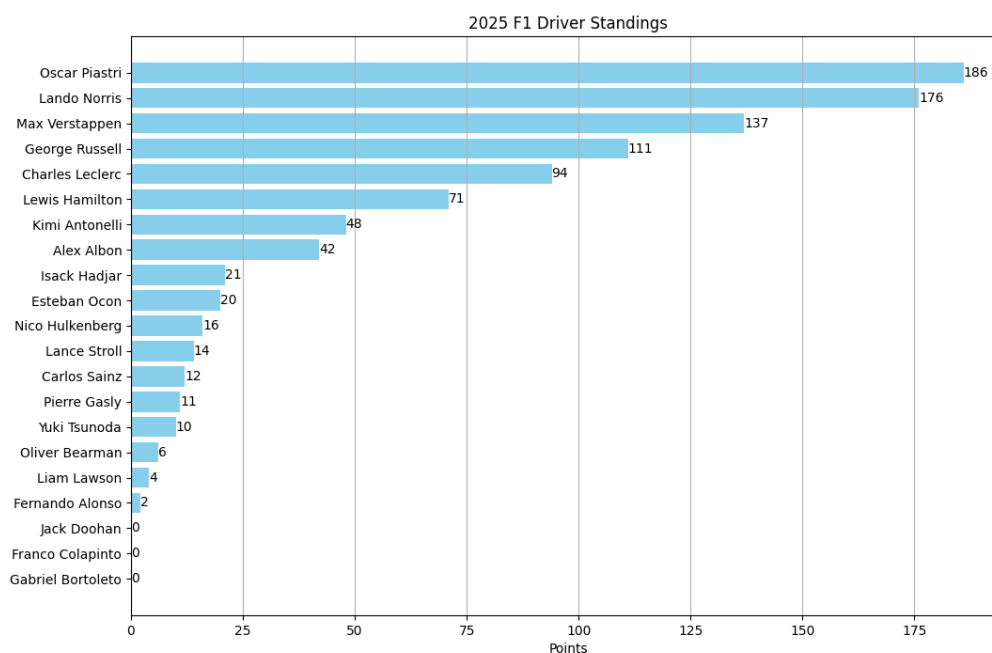


Figura 26: Exemple de gràfica generada amb dades obtingudes del Web Search Agent. Font: pròpia

Amb l'exemple previ, ja es compleix la part del requeriment 12 que estableix que s'han de poder generar gràfiques amb les dades obtingudes amb els altres agents. A continuació, es mostra un altre exemple on se li ha demanat que mostri la posició mitja dels pilots en la temporada 2025 i, posteriorment, realitzi una anàlisi detallada, complint així amb la totalitat del requeriment.

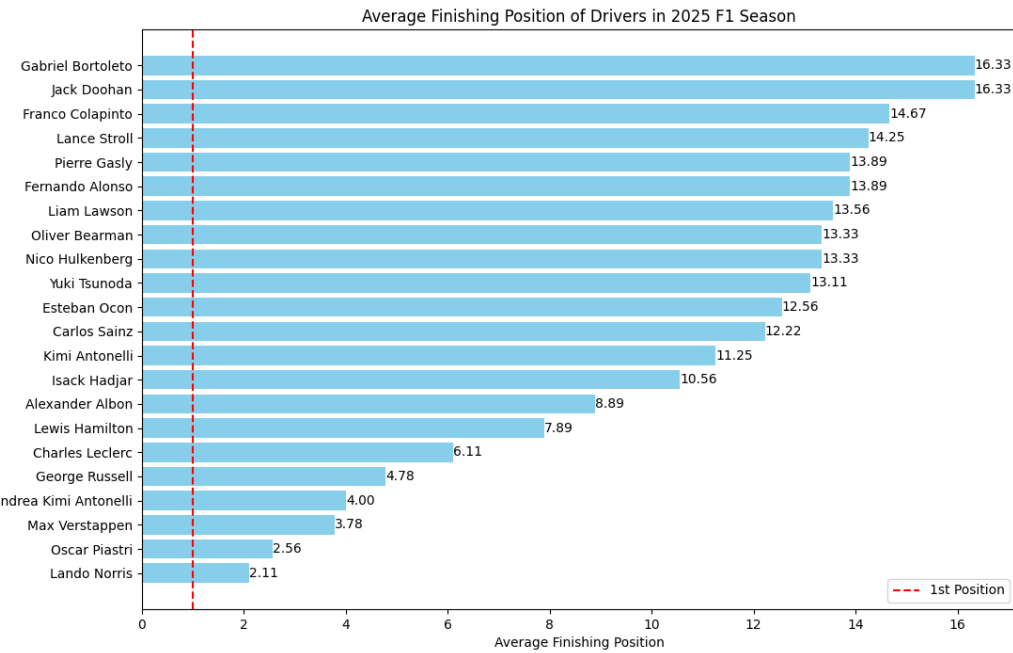


Figura 27: Exemple de gràfica generada amb dades obtingudes del Data Agent. Font: pròpia

Final answer: The graph presents the average finishing position of Formula 1 drivers for the 2025 season. Here are the key observations:

- Performance Range:** The average finishing positions vary significantly among the drivers, ranging from 2.11 to 16.33.
- Top Performers:**
 - Lando Norris has the best average finishing position at 2.11, indicating he consistently finishes near the front of the grid.
 - Oscar Piastri follows closely with an average finishing position of 2.56.
- Middle Pack:**
 - Drivers like George Russell (4.78), Charles Leclerc (6.11), and Lewis Hamilton (7.89) show competitive performances, typically finishing in the top tier but not consistently in the top three.
- Bottom Performers:**
 - The bottom five positions feature drivers like Gabriel Bortoleto and Jack Doohan, who both have an average of 16.33, indicating struggles throughout the season.
- Visual Representation:** The horizontal bar chart effectively illustrates these average positions, allowing for a quick comparison across drivers. The red dashed line signifies the 1st position.
- General Insights:** The data highlights discrepancies in team performance and driver skill among the grid, suggesting areas for improvement for lower-ranked drivers and teams.

Overall, this graph provides a clear snapshot of driver performance for the 2025 F1 season, allowing for easy identification of top and lower-performing drivers.

Figura 28: Anàlisi del gràfic previ generat pel Graph Agent. Font: pròpia

5.2 Avaluació del funcionament i exemples d'ús

Vists els resultats que pot produir aquest sistema, en aquest apartat es farà una anàlisi del seu rendiment mitjançant algunes mètriques rellevants com el temps de

resposta, els costos del model i els passos realitzats.

Per poder posar en context els resultats obtinguts, idealment s'hauria de realitzar una comparativa amb alguna eina similar. Desafortunadament, no s'ha trobat cap *chatbot* especialitzat en Fórmula 1 que sigui capaç de crear gràfiques utilitzant les dades generades pels cotxes. El més proper que existeix, són eines com *ChatGPT* d'*OpenAI*[40] que realitzen cerques a internet per obtenir les dades, per tant, seria com el nostre Graph Agent connectat únicament amb el Web Search Agent.

Tot i això, s'ha volgut fer una petita prova demanant a *ChatGPT* que generi una gràfica de barres representant els punts dels pilots a la classificació general de la temporada actual i, utilitzant el model GPT-4o, s'ha obtingut el resultat desitjat en un temps de 30.19 segons. En comparativa, el millor temps que s'ha obtingut amb el nostre agent és de 33.92 segons, utilitzant el model GPT-4o-mini. Aquesta comparativa, però, deixa de tenir sentit quan es volen un altre tipus de gràfiques que *ChatGPT* no pot generar.

5.2.1 Anàlisi del rendiment

Per avaluar les mètriques comentades, es mostraran un parell d'exemples d'execució que busquen replicar una interacció esperada amb el sistema.

Primera consulta: *'Plot the average speed of every drivers' fastest lap time in the 2025 Spanish GP Qualifying'*

Segona consulta: *'Plot it as points and include the exact number'*

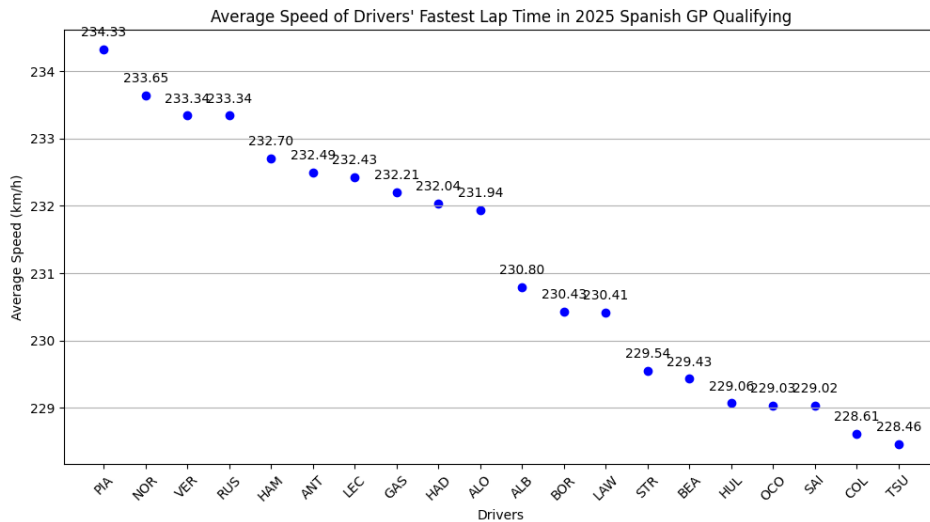


Figura 29: Gràfic mostrant la velocitat mitjana de cada pilot durant la seva volta ràpida en la classificació del Gran Premi d'Espanya de 2025. Font: pròpia

Mètriques	Consulta 1	Consulta 2	Total
Tokens d'entrada	37544	12805	50349
Tokens de sortida	2249	545	2794
Temps total (s)	73,43	10,37	83,8
Passos Graph Agent	4	2	6
Passos Data Agent	5	0	5
Cost (\$)	0,007	0,002	0,009

Taula 4: Anàlisi de les mètriques. Font: pròpia

Es pot observar com, degut a totes les crides que s'han hagut de fer a la API (com a mínim 20, una per cada pilot), el temps d'execució de la primera consulta és bastant elevat comparat amb el de la segona. Pel que fa al nombre de passos, aquests indiquen que, si hi ha hagut errors, s'han solucionat ràpidament ja que el nombre no és gaire elevat. Gràcies a això, el preu total, calculat amb la fórmula $Cost(\$) = (Input \cdot 0,15 + Output \cdot 0,6) / 1.000.000$, és relativament baix, menys d'un cèntim.

De fet, el preu real d'aquest exemple ha sigut de 0,005\$, això és degut a que els tokens que ja s'han passat al LLM es guarden en una *cache* i el seu preu és inferior.

Analitzem ara un altre exemple:

Primera consulta: *'Build a graph comparing the race position and qualifying position of every driver in the 2025 spanish gp'*

Segona consulta: *'Include the name of each driver representing the point'*

Tercera consulta: *'Change the scale of the axis to integers numbers and in intervals of 2. Draw a diagonal line in the axis representing the race performance of every driver (line should pass between points in which the qualifying and race position is the same)'*

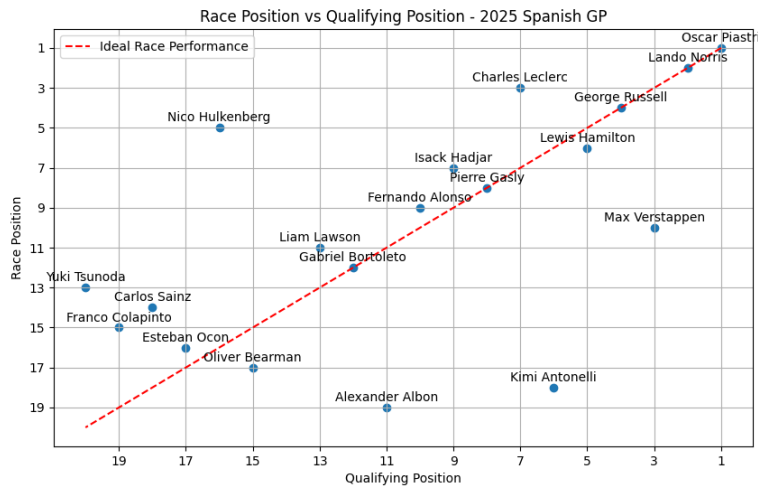


Figura 30: Gràfic comparant la posició de carrera amb la de classificació al Gran Premi d'Espanya de 2025. Font: pròpia

Mètriques	Consulta 1	Consulta 2	Consulta 3	Total
Tokens d'entrada	29053	10471	20556	60080
Tokens de sortida	1566	418	1008	2982
Temps total (s)	39,7	8,64	23,51	71,85
Passos Graph Agent	5	2	3	10
Passos Data Agent	3	0	0	3
Cost (\$)	0,005	0,002	0,004	0,011

Taula 5: Anàlisi de les mètriques. Font: pròpia

En aquest exemple, es pot observar com el temps de la primera consulta s'ha reduït considerablement degut a que s'han pogut obtenir les dades amb menys crides a la API. Es pot veure també que la mida d'aquestes dades és inferior, resultant en una reducció del cost. Finalment, a l'última consulta, on s'han introduït varies tasques a la vegada, l'agent necessita un pas extra per modificar el gràfic (en comparació amb la consulta prèvia).

Després de realitzar múltiples proves similars a aquests exemples, en les consultes que impliquen crides als agents de recuperació d'informació, s'han obtingut temps de resposta d'entre 20 a 90 segons, depenent de les crides necessàries i dels errors interns que es produeixen que impliquen passos addicionals. El cost d'aquestes consultes, ha oscil·lat entre 0,005\$ i 0,01\$, un preu relativament baix.

Per les consultes que únicament modificaven la visualització del gràfic, s'han obtingut temps de resposta d'entre 5 a 30 segons, depenent de la complexitat de la consulta, i costos entre 0,002\$ i 0,005\$.

5.2.2 Dificultats trobades

Una de les mètriques que es volia avaluar també era la precisió, és a dir, el percentatge de vegades que el sistema produïa la resposta esperada. S'han pensat algunes formes de poder mesurar-la com, per exemple, donat un *prompt*, repetir la seva execució varies vegades i avaluar si la sortida compleix amb la tasca.

El problema es que, al canviar mínimament aquest *prompt*, aquesta precisió calculada deixa de tenir sentit. És a dir, s'estaria calculant la precisió del *prompt* en el sistema i no la precisió del sistema en si. Per poder calcular-la de forma acurada, s'hauria de dissenyar un *benchmark* establint una gran quantitat de *prompts* relacionats amb les capacitats de l'agent i la seva corresponent sortida.

El disseny d'aquests *benchmarks*, però, és una tasca que requereix de coneixements més avançats que queden fora dels objectius del projecte.

Com a observació, és de coneixement general que, en aquests tipus d'agents, contra més específica sigui la descripció de la tasca, millor precisió s'obtindrà. És per això que conèixer l'estructura interna del sistema aporta una gran avantatge a l'hora d'escriure aquests *prompts*.

6 Conclusions i treball futur

Malgrat la falta d'avaluació de la precisió del sistema, s'han aconseguit resultats molt positius. S'han complert tots els requeriments establerts en la fase prèvia al disseny i la implementació, permetent a un usuari consultar dades avançades i generar gràfiques de Fórmula 1 mitjançant llenguatge natural. També, s'ha donat la capacitat al sistema de poder analitzar aquestes gràfiques i donar context a les seves respostes.

Pel que fa als objectius definits a l'inici del projecte, aquests es van dividir en dos, objectius relatius a l'aprenentatge i objectius relatius al desenvolupament.

La fase d'aprenentatge s'ha considerat un èxit, ja que ha permès l'adquisició de coneixements nous i ha reforçat els estudiats durant la carrera. Durant aquesta, s'ha investigat sobre els *Large Language Models* (LLMs) i els agents d'intel·ligència artificial, adquirint una base de coneixement necessària per poder dissenyar i implementar un bon sistema multiagent. També, s'ha pogut observar que, a mesura que s'aprofundia en la matèria, sorgien cada vegada més opcions diferents per afrontar el desenvolupament del projecte. D'entre aquestes, s'ha volgut explorar la branca dels *Code Agents* ja que és un enfocament relativament nou amb unes capacitats molt prometedores.

Pel que fa al desenvolupament, s'ha complert l'objectiu principal de dissenyar i implementar un agent d'intel·ligència artificial capaç de consultar i visualitzar dades de Fórmula 1. No obstant, aquest té certes limitacions relatives a la recuperació de la informació de la API de F1. Quan es realitza una consulta que implica múltiples crides a aquesta i que relaciona dades de taules diferents, es perd bastanta precisió. Una opció que milloraria bastant el rendiment del sistema, seria implementar una base de dades pròpia amb aquesta informació. D'aquesta forma, es reduiria considerablement el temps de resposta general i augmentaria la precisió en aquest tipus de consultes.

També, la informació guardada en la base de dades vectorial és bastant limitada, contenint únicament documents de la temporada 2025. Per tant, una millora òbvia del sistema seria augmentar la quantitat d'informació disponible per expandir el coneixement de l'agent.

Per poder establir una conversació amb un usuari, era necessari implementar una interfície gràfica. Com que el seu disseny quedava fora dels objectius del projecte, s'ha optat per una opció molt simple amb les funcionalitats bàsiques. Un disseny més elaborat hauria pogut millorar l'experiència de l'usuari i hagués facilitat la implementació de funcionalitats addicionals com afegir imatges de gràfiques a la consulta.

Finalment, gràcies a l'arquitectura del sistema, es podrien afegir fàcilment altres tipus d'agents amb funcionalitats diferents com, per exemple, un agent amb la capacitat de fer prediccions dels resultats d'una carrera.

En conclusió, gràcies a la realització d'aquest projecte s'ha pogut investigar sobre aquestes tecnologies tan populars avui dia i s'ha desenvolupat un cas d'ús que

demostra el seu potencial, consolidant i ampliant així els coneixements adquirits durant la carrera.

Referències

- [1] Kevin Bartley (2025). *Big data statistics: How much data is there in the world?* Recuperat de <https://rivery.io/blog/big-data-statistics-how-much-data-is-there-in-the-world/> [Últim accés: Maig 2025]
- [2] Rich Miller (2021). *How Cloud Data-Crunching Power Accelerates the F1 Racing Experience*. Recuperat de <https://www.datacenterfrontier.com/cloud/article/11427867/how-cloud-data-crunching-power-accelerates-the-f1-racing-experience> [Últim accés: Maig 2025]
- [3] 3Blue1Brown (2024). *Brief explanation of extensive language models (LLM)*. YouTube. <https://www.youtube.com/watch?v=LPZh9B0jkQs> [Últim accés: Maig 2025]
- [4] Wikipedia (2025). *Large language model*. https://en.wikipedia.org/wiki/Large_language_model [Últim accés: Maig 2025]
- [5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention is all you need*. arXiv preprint arXiv:1706.03762. Recuperat de <https://arxiv.org/abs/1706.03762> [Últim accés: Maig 2025]
- [6] Hugging Face (2025). *What is a Large Language Model?* Recuperat de <https://huggingface.co/learn/agents-course/unit1/what-are-llms#what-is-a-large-language-model> [Últim accés: Maig 2025]
- [7] Hugging Face (2025). *Understanding Next Token Prediction*. Recuperat de <https://huggingface.co/learn/agents-course/unit1/what-are-llms#understanding-next-token-prediction> [Últim accés: Maig 2025]
- [8] Lee Boonstra. Google (2024). *Prompt Engineering Whitepaper (v4)*. Recuperat de https://ia800803.us.archive.org/22/items/whitepaper-prompt-engineering-v-4/whitepaper_Prompt%20Engineering_v4.pdf [Últim accés: Maig 2025]
- [9] Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4a ed.). Pearson.
- [10] Hugging Face (2025). *What Are Agents?* Recuperat de <https://huggingface.co/learn/agents-course/unit1/what-are-agents>. [Últim accés: Maig 2025]
- [11] Hugging Face (2025). *Introduction to Agents*. Recuperat de https://huggingface.co/docs/smolagents/conceptual_guides/intro_agents. [Últim accés: Maig 2025]
- [12] LangGraph (s.f.). *Agent architectures*. Recuperat de https://langchain-ai.github.io/langgraph/concepts/agentic_concepts/. [Últim accés: Maig 2025]

- [13] Hugging Face (2025). *What are Tools?* Recuperat de <https://huggingface.co/learn/agents-course/unit1/tools>. [Últim accés: Maig 2025]
- [14] Hugging Face (2025). *How do multi-step agents work?* Recuperat de https://huggingface.co/docs/smolagents/conceptual_guides/react. [Últim accés: Maig 2025]
- [15] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022) *ReAct: Synergizing Reasoning and Acting in Language Models*. arXiv preprint arXiv:2210.03629. Recuperat de <https://arxiv.org/pdf/2210.03629>. [Últim accés: Maig 2025]
- [16] Hugging Face (2025). *Understanding AI Agents through the Thought-Action-Observation Cycle*. Recuperat de <https://huggingface.co/learn/agents-course/unit1/agent-steps-and-structure>. [Últim accés: Maig 2025]
- [17] Wang, X., Chen, Y., Yuan, L., Zhang, Y., Li, Y., Peng, H., i Ji, H. (2024). *Executable Code Actions Elicit Better LLM Agents*. arXiv preprint arXiv:2402.01030. Recuperat de <https://arxiv.org/pdf/2402.01030>. [Últim accés: Maig 2025]
- [18] Hugging Face (2025). *Building Good Agents*. Recuperat de https://huggingface.co/docs/smolagents/tutorials/building_good_agents. [Últim accés: Maig 2025]
- [19] LangGraph (s.f.). *Human-in-the-loop*. Recuperat de <https://langchain-ai.github.io/langgraph/agents/human-in-the-loop>. [Últim accés: Maig 2025]
- [20] LangGraph (s.f.). *Multi-agent*. Recuperat de <https://langchain-ai.github.io/langgraph/agents/multi-agent/>. [Últim accés: Maig 2025]
- [21] Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., i Zhang, X. (2024). *Large Language Model based Multi-Agents: A Survey of Progress and Challenges*. arXiv preprint arXiv:2402.01680. Recuperat de <https://arxiv.org/pdf/2402.01680>. [Últim accés: Maig 2025]
- [22] Hugging Face (s.f.). *Agentic RAG: turbocharge your RAG with query reformulation and self-query*. Recuperat de https://huggingface.co/learn/cookbook/agent_rag. [Últim accés: Maig 2025]
- [23] Hugging Face (2025). *Agentic Retrieval Augmented Generation (RAG)*. Recuperat de: <https://huggingface.co/learn/agents-course/unit3/agent-c-rag/agent-c-rag>. [Últim accés: Maig 2025]
- [24] Hugging Face (s.f.). *Advanced RAG on Hugging Face documentation using LangChain*. Recuperat de https://huggingface.co/learn/cookbook/advanced_rag. [Últim accés: Maig 2025]

- [25] FullStackRetrieval-com. GitHub (2024). *5 Levels Of Text Splitting*. Recuperat de https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb. [Últim accés: Maig 2025]
- [26] Hugging Face (2025). *Building Agents That Use Code*. Recuperat de https://huggingface.co/learn/agents-course/unit2/smolagents/code_agents. [Últim accés: Maig 2025]
- [27] Fédération Internationale de l'Automobile (2025). *FIA Formula One World Championship - Documents*. Recuperat de: <https://www.fia.com/documents/championships/fia-formula-one-world-championship-14/>. [Últim accés: Maig 2025]
- [28] Docker Inc. *Docker*. <https://www.docker.com/>
- [29] Hugging Face. *smolagents*. <https://github.com/huggingface/smolagents>
- [30] LangChain. *LangGraph*. <https://www.langchain.com/langgraph>
- [31] Bruno Godefroy. *OpenF1*. <https://openf1.org/>.
- [32] Philipp Schaefer. *Fast-F1*. <https://github.com/the0ehrly/Fast-F1>
- [33] SerpApi, LLC. *SerpApi*. <https://serpapi.com/>
- [34] NumFOCUS, Inc. *Pandas*. <https://pandas.pydata.org/>
- [35] NumFOCUS, Inc. *Matplotlib*. <https://matplotlib.org/>
- [36] LangChain. *LangChain Community*. <https://pypi.org/project/langchain-community/>
- [37] Meta Platforms, Inc. *MetaAI*. <https://www.meta.ai/>
- [38] DeepSeek. *DeepSeek*. <https://www.deepseek.com/>
- [39] Mistral AI. *Mistral*. <https://mistral.ai/>
- [40] OpenAI. *OpenAI*. <https://openai.com/>
- [41] OpenRouter, Inc. *GPT-4o-mini via OpenRouter*. <https://openrouter.ai/openai/gpt-4o-mini>
- [42] Gradio Team. *Gradio*. <https://www.gradio.app/>

A Annex

En aquesta secció s'han volgut incloure alguns exemples de gràfiques generades amb el sistema multiagent, al repositori de GitHub (https://github.com/HyKei04/F1_Agent) es poden trobar més exemples.

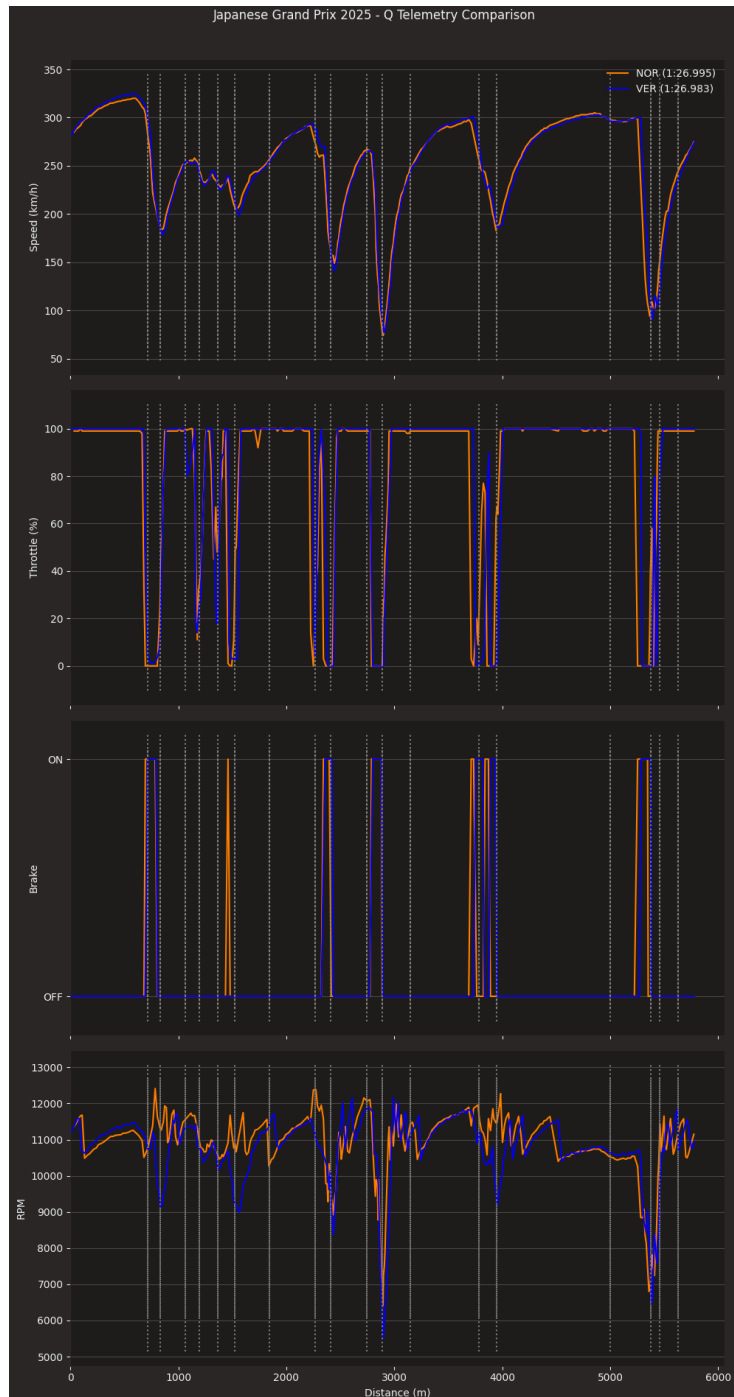


Figura 31: Comparació de la telemetria de les voltes ràpides de Lando Norris i Max Verstappen al Gran Premi del Japó de 2025 . Font: pròpia

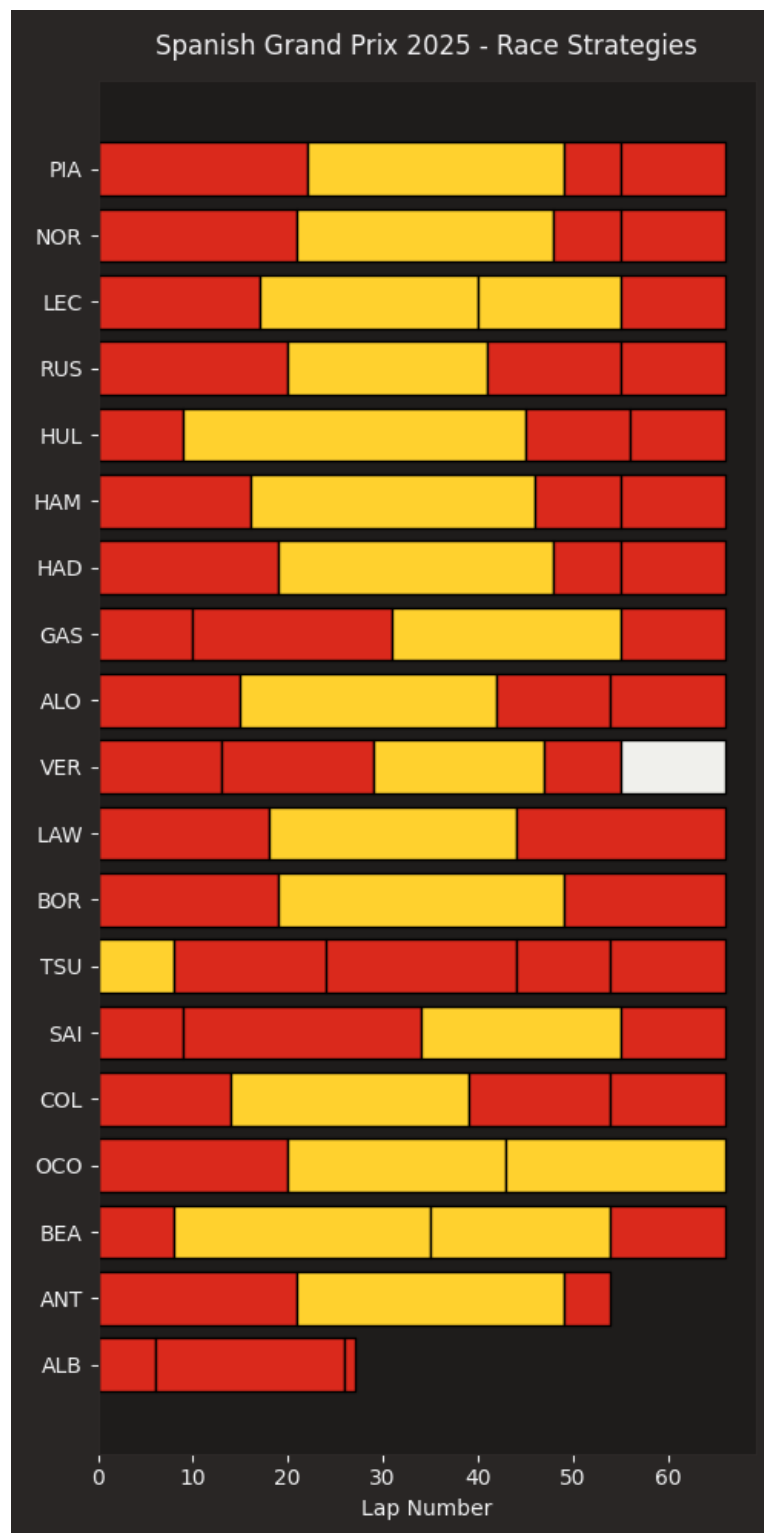


Figura 32: Estratègies de carrera dels diferents pilots en el Gran Premi d'Espanya de 2025. Font: pròpia

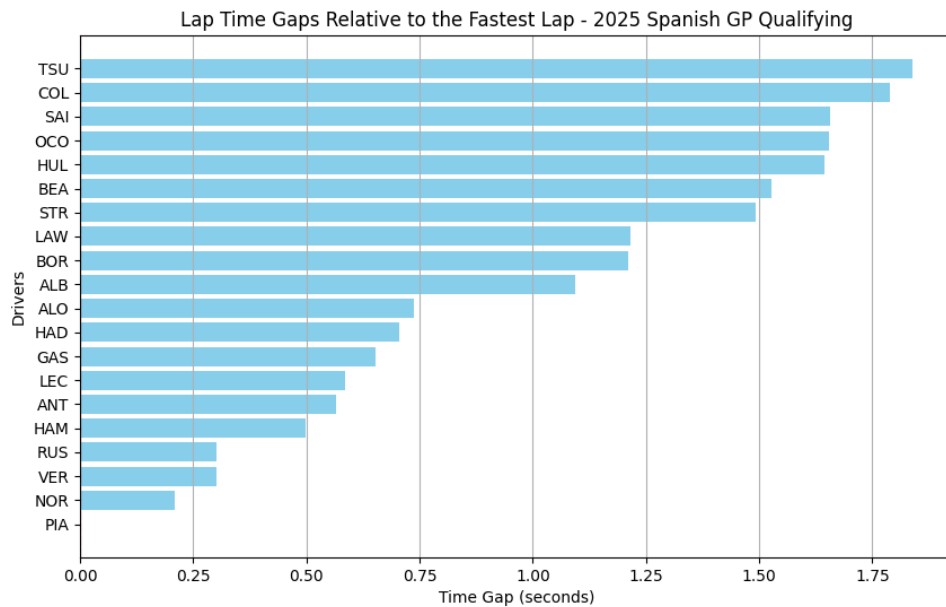


Figura 33: Distància a la volta més ràpida en la classificació del Gran Premi d'Espanya de 2025. Font: pròpia

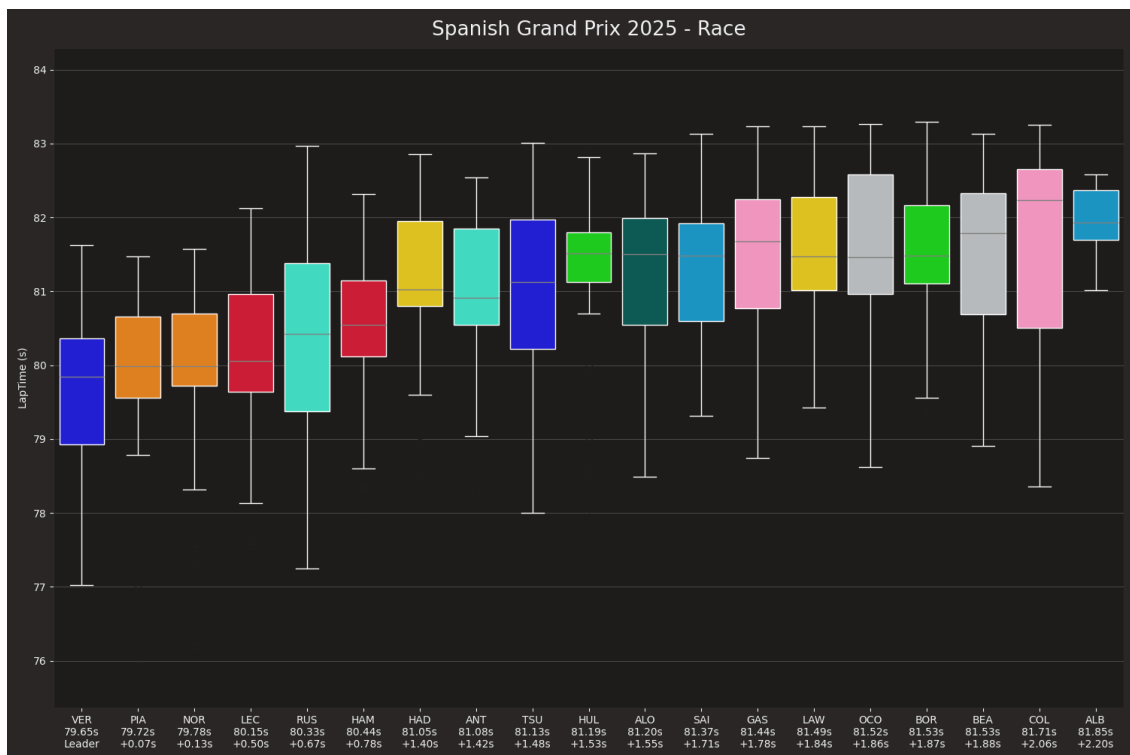


Figura 34: Boxplot mostrant el ritme de carrera de cada pilot en el Gran Premi d'Espanya de 2025. Font: pròpia

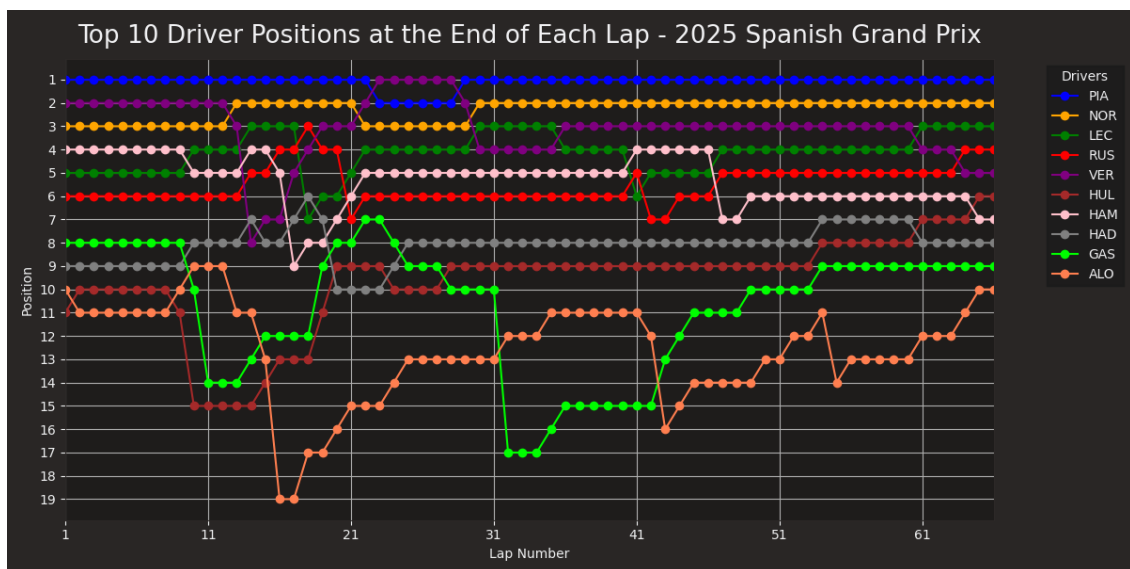


Figura 35: Posició per volta en la carrera del Gran Premi d'Espanya de 2025. Font: pròpia