Flujos de trabajo inteligentes con LLM, MCP, A2A y mucho más

Agentes de IA EN ACCIÓN

Segunda edición



Micheal Lanham

ANAYA MULTIMEDIA

Contenidos

Agradecimientos *5*
Sobre el autor *7*
Sobre la imagen de cubierta *7*

Prefacio ***13***

Sobre el libro ***15***

Quién debería leer este libro 15
Novedades de la segunda edición 16
Cómo está organizado este libro: una hoja de ruta 16
Acerca del código 18

1 El auge de los agentes de IA ***19***

1.1. Definición de agente y pensamiento agéntico 20
1.2. Presentación del protocolo MCP 28
1.3. Comprender las cinco capas funcionales de un agente 30
1.4. Avanzando hacia los sistemas multiagente 38
1.5. Próximos pasos 42
Resumen 43

2 Componentes esenciales: LLM, prompting y agentes ***45***

2.1. Comprender los grandes modelos de lenguaje 46
2.2. Controlar los LLM con ingeniería de *prompts* (persona del agente) 51
2.3. Crear un agente con el SDK de agentes de OpenAI 57
2.4. Mejorar los agentes mediante integración de herramientas 65
2.5. Ejercicios 71
Resumen 73

3 Acciones con MCP para agentes de IA ***75***

3.1. Comprender los fundamentos de MCP para el desarrollo de agentes 76
3.2. Primeros pasos con los servidores MCP 86
3.3. Utilizar servidores MCP con agentes 95
3.4. Crear servidores MCP para agentes 101
3.5. Ejercicios 107
Resumen 109

4 Diseño y desarrollo de sistemas multiagente ***111***

4.1. Diseñar sistemas multiagente 112
4.2. Equilibrar agentes con flujos agénticos 127
4.3. Comprender los traspasos en los flujos de agentes 135
4.4. Validar los flujos de agentes mediante barreras de seguridad 140
4.5. Ejercicios 150
Resumen 151

5 Razonamiento y planificación agénticos ***153***

5.1. Entender el razonamiento y la planificación de los LLM 154
5.2. Instruir a los agentes para que razonen y planifiquen 162
5.3. Razonamiento avanzado con agentes 164
5.4. Utilizar el servidor MCP de pensamiento secuencial 170
5.5. Ejercicios 177
Resumen 179

6 Trabajo con RAG de memoria y conocimiento para agentes 181

6.1. Entender la recuperación en las aplicaciones de inteligencia artificial ... 182
6.2. Bases de datos vectoriales y búsquedas por similitud ... 191
6.3. Construir agentes de conocimiento RAG prácticos ... 196
6.4. Añadir memoria a los agentes con MCP ... 205
6.5. Ejercicios ... 217
Resumen ... 219

7 Creación de agentes robustos mediante evaluación y retroalimentación 221

7.1. Presentación de la evaluación y retroalimentación agénticas ... 222
7.2. Implementación del desarrollo de agentes dirigido por pruebas ... 225
7.3. Recurrir a agentes de anclaje, agentes críticos y de evaluación ... 235
7.4. Phoenix para evaluación y retroalimentación ... 246
7.5. Ejercicios ... 254
Resumen ... 256

8 Despliegue de agentes y sistemas agénticos 259

8.1. Estrategias para consumir agentes ... 260
8.2. Desplegar sistemas agénticos con Docker ... 269
8.3. Análisis de estrategias de despliegue avanzadas ... 276
8.4. Seguridad, protección y gobernanza en la producción ... 286
8.5. Ejercicios ... 294
Resumen ... 296

9 Comprender el bucle agéntico 299

9.1. Desentrañar las tres capas de los bucles de agentes ... 300
9.2. Capa 2: Bucles con un agente de investigación profunda ... 304
9.3. Capa 3: Bucles de orquestación multiagente ... 315
9.4. Creación de bucles agénticos colaborativos ... 321
9.5. Ejercicios ... 325
Resumen ... 327

10 Exploremos el agente cognitivo que piensa, supervisa y se adapta 329

10.1. Comprender la cognición y la metacognición de los agentes como conceptos de ingeniería ... 330
10.2. Representación de la mente en una arquitectura de agente cognitivo ... 338
10.3. Creación y ejecución del agente cognitivo ... 352
10.4. La evaluación de la capacidad cognitiva y la visión de futuro ... 360
10.5. Ejercicios ... 363
Resumen ... 365

11 Consejos para crear sistemas agénticos 367

11.1. Consejos probados en la práctica, organizados según las cinco capas agénticas ... 367
11.2. Consejos para crear un agente de atención al cliente ... 377
11.3. Consejos para crear un sistema de agentes RAG ... 380
11.4. Consejos para crear un sistema de agentes de investigación profunda ... 381
Resumen ... 385

Apéndice A. Configuración del repositorio de códigos de ejemplo 389

A.1. Clonar el repositorio ... 389
A.2. Creación de un entorno de Python ... 390
A.3. Instalación de dependencias y configuración del entorno ... 390
A.4. Ejecutar el código de ejemplo ... 391

Apéndice B. Configuración de Node.js para servidores MCP locales 393

B.1. Instalación de Node.js ... 393
B.2. Comprobación de la instalación de Node y npx ... 395
B.3. Ejecutar un servidor MCP con npx ... 396
B.4. Solución de problemas y mantenimiento del buen funcionamiento de Node ... 397

Índice alfabético 399

Prefacio

Mi andadura en el mundo de los sistemas inteligentes comenzó a principios de la década de 1980. Al igual que mucha gente en aquella época, yo pensaba que la inteligencia artificial (IA) estaba a la vuelta de la esquina. Siempre parecía que una innovación más, un salto tecnológico más, nos llevaría a la inteligencia que imaginábamos. Sin embargo, ese salto nunca llegó.

Quizá la promesa de HAL, de la película de Stanley Kubrick *2001: Una odisea del espacio*, me cautivó con la idea de tener un compañero informático verdaderamente inteligente. Tras años de esfuerzo, pruebas e innumerables errores, empecé a comprender que crear IA era mucho más complejo de lo que los humanos habíamos imaginado. A principios de la década de 1990 cambié de enfoque, y apliqué mis habilidades a objetivos más tangibles en otros sectores.

No fue hasta finales de la década de los años 90, tras vivir una serie de acontecimientos difíciles y transformadores, cuando me di cuenta de mi pasión por la creación de sistemas inteligentes. Sabía que quizá estos sistemas nunca llegarían a alcanzar la superinteligencia de HAL, pero no me importaba. Encontré satisfacción trabajando con el aprendizaje automático y la ciencia de datos, creando modelos capaces de aprender y adaptarse. Durante más de 20 años me desarrollé plenamente en este ámbito, abordando problemas que requerían creatividad, precisión y un sentido de las posibilidades.

Durante ese tiempo trabajé en todo tipo de proyectos, desde algoritmos genéticos para predecir entradas desconocidas hasta el desarrollo de modelos de aprendizaje generativo para la perforación horizontal en el sector del petróleo y el gas. Estas experiencias me llevaron a escribir, compartiendo mis conocimientos a través de libros sobre diversos temas, entre ellos la ingeniería inversa de *Pokémon Go*, la creación de experiencias de realidad aumentada y virtual, el diseño de audio para videojuegos y la aplicación del aprendizaje

por refuerzo para crear agentes inteligentes. Pasé años sumergido de lleno en el código, desarrollando agentes con el kit de código abierto Unity ML-Agents y con aprendizaje profundo por refuerzo.

Ni siquiera entonces pensé que algún día podría limitarme a describir lo que quería a un modelo de IA y que este lo hiciera realidad. Nunca se me pasó por la mente que llegaría a ser capaz de colaborar con una IA con tanta naturalidad como lo hago hoy y, desde luego, nunca imaginé lo rápido (y, al mismo tiempo, lo lento) que se me haría este viaje.

En noviembre de 2022, el lanzamiento de ChatGPT lo cambió todo. Transformó la percepción que el mundo tenía de la IA y la forma en que creamos sistemas inteligentes. En mi caso, también modificó mi perspectiva sobre las capacidades de estos sistemas. De repente, la idea de agentes capaces de realizar tareas complejas de forma autónoma ya no era un sueño remoto, sino una realidad tangible y alcanzable. En algunos de mis libros anteriores, había descrito sistemas basados en agentes capaces de llevar a cabo tareas específicas, pero ahora esas ideas, que antes eran puramente teóricas, estaban al alcance de la mano.

Este libro es la culminación de mis décadas de experiencia en el desarrollo de sistemas inteligentes, pero también es la materialización de los sueños que una vez tuve sobre lo que la IA podría llegar a ser. Los agentes de IA ya están aquí, listos para transformar el modo en que interactuamos con la tecnología, cómo trabajamos y, en última instancia, cómo vivimos.

Fue una agradable sorpresa que la primera edición de este libro tuviera tan buena acogida. Creo que tuve mi momento revelador (en el que comprendí el impacto que tendrían los agentes) a finales de 2023, aunque ahora me parece que fue hace décadas. Ello impulsó mi pasión por escribir la primera edición y, sobre todo, cambió mi forma de trabajar y mi visión del futuro de cómo trabajaremos (especialmente en disciplinas técnicas como el desarrollo de software). En aquellos primeros tiempos, estoy convencido de que la gente pensaba que yo estaba loco por explicar cómo los agentes cambiarían nuestra forma de percibir el trabajo. Por supuesto, aún eran los albores de la revolución de la IA, y la mayoría de la gente estaba aprendiendo a utilizar ChatGPT de una forma eficaz. Intentar explicarles el cambio de chatbots a agentes era una tarea abrumadora.

En tan poco tiempo, los agentes se han convertido en la columna vertebral de la IA, y la época de los chatbots ya ha quedado muy atrás. La IA ha ido más allá de los modelos fundamentales para adentrarse en los marcos de trabajo de los agentes y las herramientas que los potencian. Aunque el revuelo sigue enfocado en los lanzamientos de nuevos modelos, en realidad, la mayor parte del trabajo y la influencia se centra en la capa oculta de los agentes, que ahora determina la forma en que interactuamos con la IA.

Sin embargo, incluso en la actualidad, observo que las organizaciones se muestran reticentes a la hora de adoptar sistemas basados en agentes. Creo que esta desconfianza no se debe al miedo a la IA, sino a la falta de conocimientos y experiencia a la hora de desarrollar estos sistemas. Espero que este libro ayude a salvar esa brecha. Quiero presentar los agentes de IA como herramientas accesibles para todos, a las que no debemos temer, sino que, por el contrario, debemos respetar, gestionar de forma responsable y aprender a trabajar con ellas en armonía.

Sobre el libro

Este libro trata sobre la creación y el uso de sistemas de agentes inteligentes. No solo habla de crear entidades autónomas, sino también de desarrollar agentes capaces de abordar y resolver de forma eficaz problemas del mundo real. El libro comienza con los fundamentos del trabajo con grandes modelos de lenguaje (LLM) para crear asistentes, sistemas multiagente y agentes con comportamiento autónomo. A partir de ahí, explora los componentes clave de los sistemas agénticos: persona, herramientas y acciones, conocimiento y memoria, recuperación y aumento, razonamiento y planificación, y evaluación y retroalimentación. A través de ejemplos prácticos, el libro muestra cómo estos componentes permiten a los agentes realizar una amplia gama de tareas complejas.

Este viaje no se limita a la tecnología; se trata de replantearnos cómo abordamos la resolución de problemas. Espero que este libro inspire al lector a ver a los agentes inteligentes como socios en la innovación, capaces de transformar ideas en acciones de formas que antes se consideraban imposibles. Juntos exploraremos cómo la IA puede impulsar el potencial humano, permitiéndonos lograr mucho más de lo que podríamos conseguir por nuestra cuenta.

Quién debería leer este libro

Este libro está dirigido a cualquier persona que sienta curiosidad por los agentes inteligentes y por cómo desarrollar sistemas basados en ellos, tanto si la intención es crear un primer asistente como si se desea profundizar en complejos sistemas multiagente. No se requiere experiencia previa con agentes, sistemas agénticos o ingeniería de *prompts*, ni en el trabajo con grandes modelos de lenguaje (LLM). Lo único que se necesita es un conocimiento básico de Python y estar familiarizado con los repositorios de GitHub. Mi objetivo es hacer que estos conceptos resulten accesibles y atractivos, para que cualquiera que quiera explorar el mundo de los agentes de IA pueda hacerlo con confianza.

Tanto si eres desarrollador, investigador, aficionado o simplemente te intrigan las posibilidades de la IA, este libro es para ti. Espero que en estas páginas encuentres inspiración, orientación práctica y una nueva forma de valorar el extraordinario potencial de los agentes inteligentes. Deja que este libro te guíe para comprender, crear y dar rienda suelta al poder de los agentes de IA en acción.

Novedades de la segunda edición

Esta edición es prácticamente una actualización completa, por lo que incluye todos los conceptos básicos nuevos que un desarrollador de sistemas agénticos necesita para ser competente. Además de un nuevo subtítulo ("Flujos de trabajo inteligentes con LLM, MCP, A2A y mucho más"), he añadido nueve capítulos nuevos sobre temas de actualidad, he eliminado el material obsoleto y he revisado y actualizado los capítulos restantes con las últimas herramientas y protocolos.

El libro cuenta ahora con un capítulo dedicado al protocolo MCP (*Model Context Protocol*) y otro al despliegue de orquestación distribuida mediante Docker. A lo largo del libro, también hay nuevas secciones o capítulos que abordan los flujos de trabajo multiagente, el pensamiento agéntico, la memoria, los sistemas de razonamiento, los sistemas de evaluación y retroalimentación, el desarrollo de agentes dirigido por pruebas, los agentes de atención al cliente, los agentes RAG, los agentes de investigación profunda, el bucle agéntico y la metacognición como automejora en los agentes.

La primera edición presentó los conceptos y patrones de las cinco capas funcionales de los agentes. En esta segunda edición, dichas capas se han formalizado en mayor medida, se presentan al inicio del libro y sirven como principio organizativo para la primera mitad del mismo.

Gran parte de la primera edición se dedicó a explicar los conceptos que hicieron posible la creación de agentes. Sin embargo, en esta ocasión no ha sido necesario explicar los entresijos de su funcionamiento, ya que ningún marco de trabajo incluye ahora esos detalles.

Al comprender estos fundamentos y patrones, los desarrolladores pueden crear agentes a nivel de producción para gestionar cualquier cosa en cualquier ámbito. Aunque los principios fundamentales de los agentes siguen evolucionando, creo que esta edición proporcionará la base para una sólida estructura de conocimientos sobre la que los lectores puedan seguir construyendo.

Cómo está organizado este libro: una hoja de ruta

Este libro consta de 11 capítulos. El capítulo 1 comienza sentando las bases con definiciones de agente y pensamiento agéntico, presentando el protocolo MCP y estableciendo las cinco capas funcionales que conforman un agente. A medida que avanza el libro, el análisis se centra en los componentes clave que dotan de potencia a los agentes y en cómo estos componentes interactúan para crear sistemas verdaderamente eficaces. A continuación, se ofrece un breve resumen de los capítulos 2 a 11:

- El capítulo 2 comienza analizando cómo funcionan los grandes modelos de lenguaje como máquinas de *tokens* probabilísticas, y cómo controlarlos mediante la ingeniería de *prompts*. A continuación, examinamos cómo crear agentes utilizando el SDK de agentes de OpenAI y cómo mejorarlos mediante la integración de herramientas, estableciendo así los fundamentos para todo lo que viene a continuación.
- El capítulo 3 se adentra en el protocolo MCP, a menudo descrito como el "USB-C" de los agentes. Exploramos su arquitectura (clientes, servidores y servicios) y mostramos cómo dar los primeros pasos con los servidores MCP, ponerlos en marcha con agentes y crear servidores MCP personalizados, que permitan a los agentes interactuar a la perfección con sistemas externos.
- El capítulo 4 explora las arquitecturas fundamentales para crear sistemas multiagente (flujos, orquestación y colaboración), además de cómo equilibrar los agentes con los flujos agénticos, gestionar los traspasos entre agentes y validar los flujos con barreras de seguridad.
- El capítulo 5 analiza el modo en que los grandes modelos de lenguaje (LLM) razonan y planifican mediante técnicas como la cadena de pensamiento (CoT) y el paradigma ReAct. Mostramos cómo dar instrucciones a los agentes para que razonen y planifiquen, exploramos estrategias avanzadas de planificación y utilizamos el servidor MCP de pensamiento secuencial para mejorar la capacidad de un agente de resolver problemas de forma autónoma.
- El capítulo 6 habla de cómo los mecanismos de recuperación pueden servir tanto como fuente de conocimiento (mediante el procesamiento de los documentos cargados mediante bases de datos vectoriales y búsquedas por similitud) como de memoria, al permitir a los agentes recordar interacciones anteriores mediante MCP.
- El capítulo 7 presenta el desarrollo de agentes dirigido por pruebas (TDAD) como una forma práctica de crear agentes fiables, muestra cómo los agentes de anclaje y los agentes críticos aportan niveles adicionales de evaluación, y explica cómo se pueden utilizar herramientas como Phoenix para llevar a cabo evaluaciones exhaustivas y obtener retroalimentación.
- El capítulo 8 presenta estrategias para el uso y despliegue de agentes en la práctica. Analizamos cómo integrar agentes en aplicaciones, exponerlos como microservicios y adaptar los sistemas agénticos a Docker para su uso en producción. El capítulo concluye con estrategias avanzadas de despliegue y consideraciones esenciales en materia de seguridad, protección y gobierno.
- El capítulo 9 presenta las tres capas del bucle agéntico (el bucle SPAL interno, el bucle de tareas externo y el metabucle) y las pone en práctica mediante la creación de un agente de investigación profunda, la implementación de bucles de orquestación multiagente y la construcción de bucles agénticos de colaboración.
- El capítulo 10 va más allá de las bases del razonamiento y explora la cognición y la metacognición como conceptos prácticos para la ingeniería de agentes. Trasladamos los fundamentos teóricos de la ciencia cognitiva a una arquitectura concreta de

agentes cognitivos, e implementamos procesos metacognitivos que permiten a los agentes reflexionar sobre su propio pensamiento, supervisar su rendimiento y adaptar sus estrategias.

- El capítulo 11 ofrece consejos probados en la práctica, organizados según las cinco capas de los agentes, junto con orientaciones específicas para cada función y destinadas a la creación de un agente de atención al cliente, patrones de diseño para un sistema de agentes RAG y un modelo para un agente de investigación profunda. El capítulo concluye integrando todos los componentes clave (acciones, memoria y conocimiento, razonamiento, evaluación, planificación y retroalimentación) en ejemplos prácticos de sistemas agénticos que resuelven problemas del mundo real.

Acerca del código

El código de este libro está distribuido en varios proyectos de código abierto, muchos de los cuales están alojados por mí o por otras organizaciones en repositorios de GitHub. A lo largo del libro, me he esforzado por asegurar que el contenido sea totalmente accesible, adoptando un enfoque con muy poco código para que el lector pueda centrarse en los conceptos fundamentales. Muchos capítulos muestran cómo unos sencillos *prompts* pueden generar código significativo, lo que pone de manifiesto el potencial del desarrollo asistido por IA.

El lector encontrará además diferentes perfiles de asistentes y sistemas multiagente, que muestran cómo resolver problemas del mundo real utilizando código generado. Estos ejemplos tienen como objetivo inspirar, guiar y proporcionar las herramientas necesarias para explorar las posibilidades que ofrecen los agentes de IA. Estoy profundamente agradecido a los numerosos colaboradores y miembros de la comunidad que han participado en estos proyectos, y animo al lector a que explore los repositorios, experimente con el código y lo adapte a sus necesidades. Este libro es un testimonio del poder de la colaboración y de las cosas increíbles que podemos lograr juntos.

El libro contiene muchos ejemplos de código fuente, tanto en listados numerados como integrados en el texto normal. En ambos casos, el código fuente se presenta con una `fuente monoespacial como esta` para diferenciarlo del texto general. Muchos de los listados van acompañados de anotaciones que destacan conceptos importantes.

Además, en muchos casos se ha reformateado el código fuente original; hemos añadido saltos de línea y hemos reajustado la sangría para adaptarnos al espacio disponible en las páginas del libro. En algunos casos, ni siquiera esto ha sido suficiente, por lo que los listados incluyen marcadores de interrupción de línea (➥). Además, con frecuencia se han eliminado los comentarios incluidos en el código fuente cuando este se describe en el texto.

Todos los ejemplos de código fuente de este libro están disponibles para su descarga en la página web de Anaya Multimedia en `https://anayamultimedia.es`, en la opción Selecciona complemento que encontrará en la ficha correspondiente a este libro. También puede descargarlos de la página web del libro original en `www.manning.com/books/ai-agents-in-action-second-edition`, y en este repositorio de Git: `https://github.com/cxbxmxcx/AI-Agent-Workflows`.

1 El auge de los agentes de IA

En este capítulo veremos cómo:

- Definir el agente y el pensamiento agéntico.
- Presentar el protocolo MCP.
- Comprender las capas funcionales del agente.
- Avanzar hacia los sistemas multiagente.

Por sí solas, las aplicaciones basadas en grandes modelos de lenguaje o LLM (*Large Language Models*), como por ejemplo los sencillos chatbots, pueden generar respuestas y contestar a preguntas. El siguiente paso es que sean capaces de elaborar planes y llevarlos a cabo. Queremos que reserven un vuelo (no solo que listen los vuelos disponibles), o que actualicen un rastreador de proyectos, no solo que enumeren los cambios que hay que realizar. Para dotar a las aplicaciones de IA de capacidades activas, añadimos al sistema agentes de IA. El agente no es un concepto nuevo en el aprendizaje automático y la inteligencia artificial, y el término puede resultar ambiguo. Cuando hablamos de agentes inteligentes o de agentes de IA, generalmente nos referimos a software que percibe su entorno, decide qué hacer y actúa para alcanzar un objetivo, utilizando los recursos que proporciona un LLM.

Este libro trata sobre la creación y el uso de agentes de IA para ampliar las capacidades de los sistemas impulsados por grandes modelos de lenguaje. Analizamos cómo funcionan los agentes, qué los hace potentes y cómo podemos crearlos para automatizar tareas complejas. Comenzamos con un único agente simplificado y, a continuación, pasamos a sistemas agénticos (basados en agentes), como el flujo de agentes, la orquestación y los patrones

de colaboración. Estos sistemas han ampliado a gran velocidad lo que un agente sencillo puede razonar y lograr. Aprenderás a crear y conectar estos agentes utilizando estructuras modernas, como el SDK de agentes de OpenAI y el protocolo MCP, vinculándolos a bases de datos, interfaces API, repositorios de conocimiento y aplicaciones completas. El objetivo es guiarte desde el nivel de aficionado hasta el de arquitecto de agentes, equipándote con patrones, código y modelos mentales que permitan a tus agentes demostrar su utilidad en entornos de producción.

Aquí nos centramos en las habilidades prácticas necesarias para diseñar y desarrollar agentes y sistemas basados en agentes. Lo hacemos mediante ejemplos concisos e informativos, en lugar de utilizar largos fragmentos de código de producción, para que los mecanismos queden patentes en cada paso. Hacia el final del libro, analizamos las arquitecturas de producción que impulsan los sistemas de IA más potentes de la actualidad. Sin embargo, este libro no es un manual de operaciones de producción. Abordamos cuestiones relacionadas con la producción, como el cumplimiento normativo y la seguridad, pero no tratamos todos los ajustes de despliegue, la pila de observabilidad ni todas las cuestiones de cumplimiento que conlleva el lanzamiento de agentes a gran escala. El objetivo es enseñarte la arquitectura y los componentes básicos a un nivel suficiente como para que esas decisiones de producción resulten manejables. Los ejemplos se mantienen deliberadamente sencillos para que las ideas queden bien claras.

El recorrido que haremos no se limita a la tecnología, sino que se trata también de replantearnos cómo abordamos la resolución de problemas. Espero que este libro te inspire a ver a los agentes inteligentes como socios en la innovación, capaces de transformar las ideas en acciones de formas que antes se consideraban imposibles.

Empecemos por aclarar qué entendemos por agentes en este contexto y qué aportan los agentes de IA a la inteligencia artificial.

1.1. Definición de agente y pensamiento agéntico

El concepto de "agente" no es nuevo en el ámbito de la inteligencia artificial. En el aprendizaje por refuerzo, un agente es una entidad que toma decisiones y aprende a partir de sus interacciones con el entorno. En otros contextos, el término se utiliza de forma más amplia para describir un software que realiza tareas en nombre de un usuario. Sin embargo, en todas estas tradiciones, la idea central es la misma: un agente percibe su entorno, decide qué hacer y actúa para alcanzar un objetivo. Esta definición es sencilla, pero lo suficientemente amplia como para abarcar desde los sistemas clásicos de inteligencia artificial hasta los agentes basados en grandes modelos de lenguaje (LLM) que construiremos en este libro.

El término "agéntico" amplía esta idea. Describe sistemas o comportamientos que muestran capacidad de acción, es decir, la idoneidad para percibir, decidir y actuar con cierto grado de autonomía en la consecución de un objetivo. Cuando decimos que un sistema es "agéntico", nos referimos a que sigue estos patrones en su funcionamiento, no simplemente a que se parezca superficialmente a un agente. Aunque las definiciones filosóficas de la expresión "capacidad de acción" pueden ser más exigentes y requerir elementos como la intención o la adaptabilidad, este enfoque práctico es suficiente para fines de ingeniería.

Esta distinción se hace más evidente cuando se compara con los asistentes de IA tradicionales. Un asistente generativo responde a una solicitud y luego se detiene. Un agente, en cambio, trabaja para alcanzar un objetivo a lo largo de varios pasos, eligiendo qué hacer a continuación en función de lo que observa por el camino. La autonomía y la persistencia transforman un sistema de asistente reactivo en agente activo. A medida que los sistemas modernos incorporan cada vez más herramientas y flujos de trabajo de varios pasos (como se observa en funciones como la investigación profunda de proveedores como OpenAI, Google y Anthropic), la línea divisoria entre asistente y agente puede difuminarse. Por eso es importante comprender los patrones subyacentes.

Aunque aquí nos centremos en los agentes impulsados por grandes modelos de lenguaje, el ámbito del diseño tiene una larga trayectoria. Los sistemas basados en reglas, el razonamiento simbólico, las arquitecturas de creencia-deseo-intención, los agentes de aprendizaje por refuerzo y los enfoques neurosimbólicos ofrecen patrones útiles, cada uno adecuado para diferentes tipos de problemas. Lo que ha cambiado es el alcance: los LLM amplían enormemente lo que un solo agente puede razonar y lograr.

Desde hace tiempo existe cierta confusión sobre qué es lo que define a un agente. Para aclarar el término, repasemos la evolución desde las interacciones directas con modelos de lenguaje hasta las acciones de los LLM, los *proxies* de asistente, los asistentes, los agentes y los sistemas agénticos.

1.1.1. Comprender los patrones de agente, asistente y LLM

Cuando se lanzaron por primera vez ChatGPT, Claude y otros sistemas similares, interactuabas directamente con el LLM subyacente: escribías una pregunta, el modelo generaba texto y ahí terminaba todo el proceso. Si utilizaste estas herramientas en sus inicios, es probable que lo que recuerdes sea esa experiencia de conversación directa. Un paso más allá de estas interacciones consiste en pedirle a un LLM que utilice una herramienta para crear una imagen, buscar en la web o realizar otra tarea. Los actuales ChatGPT y Claude hacen esto de forma habitual, pero esta acción supone un cambio significativo con respecto al patrón original. La figura 1.1 muestra ejemplos en los que un usuario puede interactuar con un LLM directamente (arriba) o a través de un LLM que utiliza herramientas, denominado asistente (abajo). Fíjate en cómo el uso de ChatGPT como asistente genera una solicitud más adecuada para la generación de imágenes.

Si has utilizado Nano Banana, Image 2.0 u otras herramientas de generación de imágenes a través de plataformas como ChatGPT o Gemini, has experimentado una interacción con un agente intermediario. En este caso, un LLM intercepta tus solicitudes y las reformula en un formato más adecuado para la tarea.

A partir de estos dos patrones de comunicación directa, hemos evolucionado hacia asistentes y agentes que utilizan herramientas. La terminología en este ámbito no está aún consolidada, y verás que los términos "asistente" y "agente" se utilizan indistintamente en muchos contextos. En este libro, trazamos la línea divisoria en función del grado de autonomía. Aquí utilizamos el término "asistente" para referirnos a un LLM que puede utilizar herramientas en nombre del usuario, pero que requiere aprobación en cada paso, y carece de la capacidad para planificar y completar múltiples tareas por sí mismo. Por el contrario,

los "agentes" pueden trabajar de forma autónoma para razonar, planificar y llevar a cabo tareas como enviar correos electrónicos, crear presentaciones, reservar vuelos y realizar investigaciones.

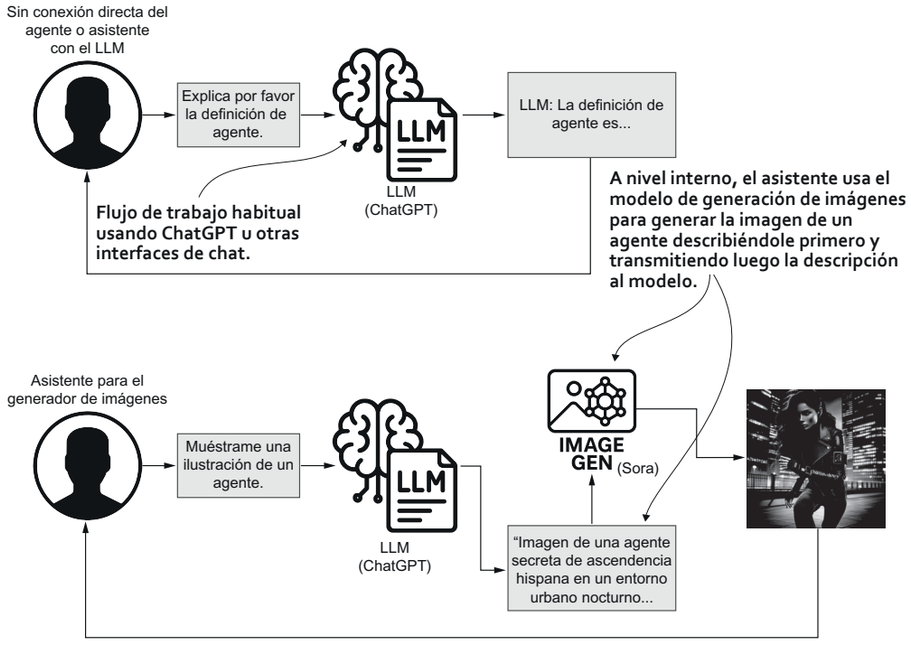


Figura 1.1. Patrones habituales para comunicarse directamente con un LLM, ya sea por sí solo o con herramientas. Si has utilizado versiones anteriores de ChatGPT, habrás experimentado la interacción directa con el LLM. Ningún agente intermediario ni otro asistente actuaba en tu nombre. Hoy en día, ChatGPT cuenta con numerosas herramientas que le ayudan a responder, entre ellas la búsqueda enla web y la programación, lo que hace que la versión actual funcione más bien como un asistente.

En la práctica, la línea que separa al asistente del agente es más bien un continuo que una frontera rígida. Los agentes de producción casi siempre aplican controles graduales con intervención humana: los pasos de bajo riesgo se ejecutan de forma autónoma; los de riesgo medio pueden requerir confirmación; y las acciones de alto riesgo, como enviar correos electrónicos, realizar compras o eliminar registros, suelen estar sujetas a una aprobación explícita, independientemente del nivel general de autonomía del agente. Cuando en este libro nos referimos a los agentes como autónomos, queremos decir que pueden planificar y ejecutar por sí mismos tareas de varios pasos. No queremos decir que deban funcionar sin supervisión en todas las situaciones. El diseño de los mecanismos adecuados de aprobación y supervisión forma parte de la creación de un agente, tema al que volveremos a lo largo del libro.

La tabla 1.1 resume los cuatro patrones de interacción con los que trabajaremos, desde el chat directo hasta los agentes autónomos.

Tabla 1.1. Los cuatro patrones de interacción con LLM que se tratan en este libro.

Patrón	Modelo de aprobación	Nivel de autonomía	Casos de uso típicos	Ejemplos de plataformas
Chat directo con LLM	No se invocan herramientas; el modelo solo genera texto	Ninguno	Preguntas y respuestas, redacción, lluvia de ideas, aprendizaje	ChatGPT original (2022), primeras versiones de Claude (2023)
LLM potenciado con herramientas	Implícita por llamada; una solicitud del usuario activa una única herramienta	Bajo	Generación de imágenes, búsqueda en la web, consultas sencillas	ChatGPT con DALL·E, Gemini con búsqueda de Google
Asistente	Aprobación por tarea; el usuario confirma cada acción	Medio	Programación en pareja, investigación citada, edición de documentos	GitHub Copilot Chat, Cursor, Claude en Excel
Agente	Aprobación a nivel de objetivo con acciones de alto riesgo controladas; el usuario establece el objetivo y el agente decide los pasos	Alto	Investigación de varios pasos, refactorizaciones en todo el repositorio, automatización del navegador	Claude Code, OpenAI Operator, Devin

La figura 1.2 muestra una comparación en paralelo de los patrones de asistente y agente. La diferencia clave radica en el papel que desempeña el humano en el proceso: el asistente requiere una aprobación en cada paso, mientras que el agente actúa para lograr el objetivo principal.

En la figura, el asistente (arriba) utiliza una única herramienta para llamar al servicio meteorológico y responder a una consulta sobre el tiempo, pero primero solicita la aprobación del usuario. Por el contrario, el usuario otorga al agente (abajo) la capacidad de tomar decisiones, razonar y planificar el uso de herramientas para alcanzar un objetivo principal. En esta situación, el usuario autoriza al agente a seleccionar las herramientas adecuadas para gestionar los correos electrónicos y las notificaciones, de modo que pueda identificar y mostrar los cinco correos electrónicos más importantes.

Si alguna vez has utilizado una herramienta de ChatGPT, como la búsqueda web, Canvas o GPT Assistant, entonces has usado un asistente. En este caso, el LLM conoce las funciones del asistente o las herramientas disponibles y puede prepararse para invocarlas. Antes de invocar una nueva herramienta externa, el LLM se detiene y solicita la aprobación del usuario. Si se aprueba, se ejecutan las herramientas y los resultados se devuelven al LLM. A continuación, el LLM reformula la respuesta en lenguaje natural y la devuelve al usuario.

Por el contrario, el agente interpreta la solicitud u objetivo del usuario y, a continuación, puede razonar y elaborar un plan para identificar nuevos puntos de decisión. A partir de ahí, ejecuta los pasos del plan y toma las decisiones necesarias de forma independiente. Puede solicitar la opinión del usuario tras completar tareas clave específicas, pero este agente tiene la capacidad y autoridad para crear y ejecutar un plan con el fin de alcanzar un objetivo. La autonomía permite a los agentes alcanzar objetivos complejos, pero hay que tener cuidado para evitar que realicen acciones desconocidas.

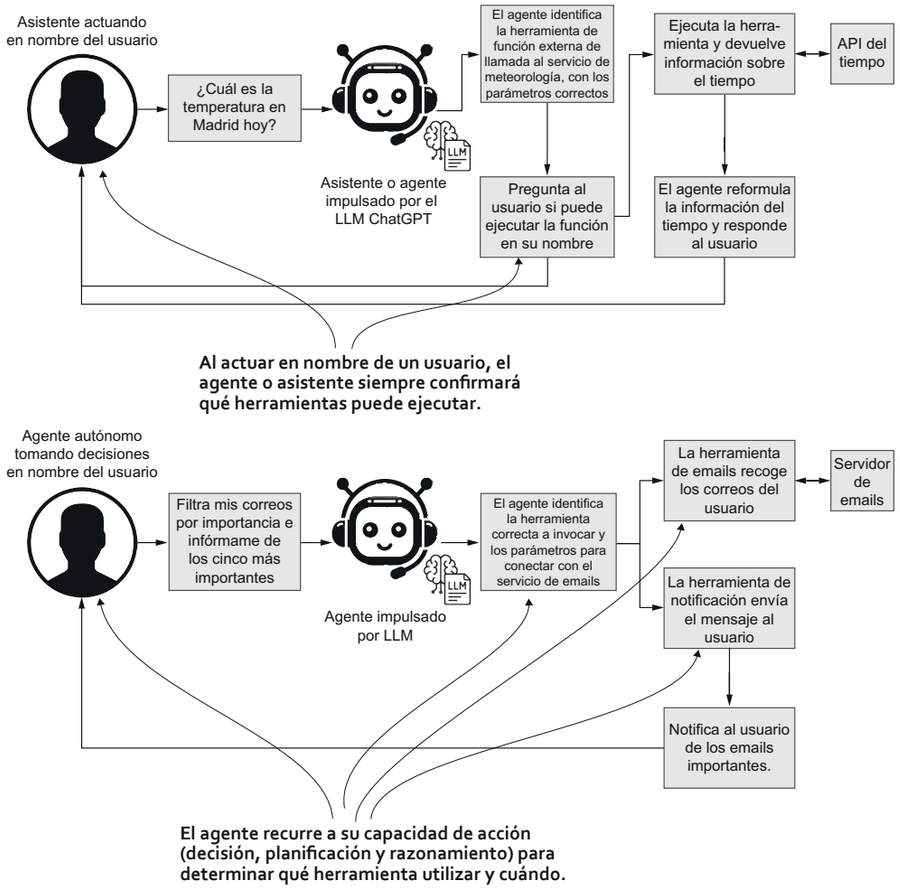


Figura 1.2. Arriba: un asistente realiza una o varias tareas en nombre de un usuario, y cada tarea requiere la aprobación de este. Abajo: un agente puede utilizar varias herramientas de forma autónoma, sin necesidad de aprobación humana, para cumplir un objetivo.

1.1.2. Pensar como los agentes: detectar, planificar, actuar, aprender

Utilizamos la expresión "capacidad de acción" para describir la capacidad de un agente para tomar decisiones y utilizar herramientas para cumplir un objetivo. A nivel interno, esto significa que los agentes (LLM) deben ser capaces de "razonar" usando las tareas necesarias para alcanzar dicho objetivo. La tabla 1.2 ofrece tres ejemplos de objetivos: crear una imagen, viajar a Calgary y comprar un ordenador. También enumera las tareas realizadas para lograr cada objetivo y las herramientas utilizadas para llevarlas a cabo.

Tabla 1.2. Ejemplos de objetivos, así como las tareas y herramientas necesarias para llevarlos a cabo.

Objetivos	Tareas	Herramientas
Crear una imagen	Crear una imagen	`create_image`
Viajar a Calgary	Buscar vuelos	`search_flights`
	Reservar vuelos	`book_flights`
	Reservar hoteles	`book_hotels`
	Reservar transporte	`book_transportation`
Comprar un ordenador	Buscar ordenadores	`web_search`
	Comparar características y precios	`web_search`
	Pedir un ordenador	`order`

Los objetivos pueden ser tan sencillos como crear una imagen o tan complejos como reservar un viaje o comprar algo. Las tareas necesarias para alcanzar un objetivo se desglosan hasta el nivel de las herramientas, de modo que cada tarea implica la ejecución de una herramienta. Las herramientas pueden estar diseñadas para tareas específicas, como crear imágenes, o para usos más generales, como la búsqueda en la web, que puede servir de apoyo para múltiples tareas. Así, el resultado ofrecido por una herramienta puede convertirse en la entrada para la siguiente herramienta, un proceso denominado "encadenamiento de herramientas".

Los agentes reciben un objetivo del usuario o del sistema y lo desglosan en tareas. A partir de ahí, el agente lleva a cabo cada una de las tareas necesarias para lograr el objetivo. La figura 1.3 ilustra el proceso habitual que realiza un agente cuando recibe la consulta de un usuario y ha de completar las tareas utilizando herramientas. Este proceso consta de cuatro pasos básicos: detectar, planificar, actuar y aprender.

El ciclo SPAL (S*ense-Plan-Act-Learn*: detectar-planificar-actuar-aprender) es un mecanismo interno que utilizan los agentes para completar objetivos que requieren distintas acciones. La capacidad de acción, incluida la idoneidad para decidir y planificar, impulsa este bucle para los agentes. SPAL, sin embargo, no es algo nuevo; se hace eco de ideas bien establecidas de la IA clásica y la robótica, incluido el ciclo OODA (observar, orientar, decidir, actuar) de la estrategia militar; las arquitecturas BDI (*Belief-Desire-Intention*: creencia, deseo, intención) de la investigación en IA basada en agentes; y los paradigmas de la planificación reactiva y "detectar-planificar-actuar" de la robótica que les precedieron. Adaptamos aquí este patrón a los agentes impulsados por LLM, donde "detectar" se corresponde con el procesamiento de entradas y contexto, "planificar" con el razonamiento del LLM, "actuar" con la ejecución de herramientas y "aprender" con la evaluación de resultados y las actualizaciones de la memoria.

Al configurar un agente, normalmente le proporcionamos las herramientas que necesita para actuar y completar tareas. A continuación, cuando se le asigna un objetivo, el agente carga sus indicaciones (*prompt*) y genera un plan internamente. Este plan especifica qué herramientas debe ejecutar y cómo debe hacerlo, así como qué resultados de las herramientas pueden tener que encadenarse. A continuación, el agente analiza los resultados de la tarea y de las herramientas para determinar si el objetivo se ha cumplido o si el proceso debe continuar. Si el objetivo se ha cumplido, el agente devuelve los resultados. Esta es una descripción general del proceso; algunos o todos estos pasos pueden resultar más complicados en la práctica.

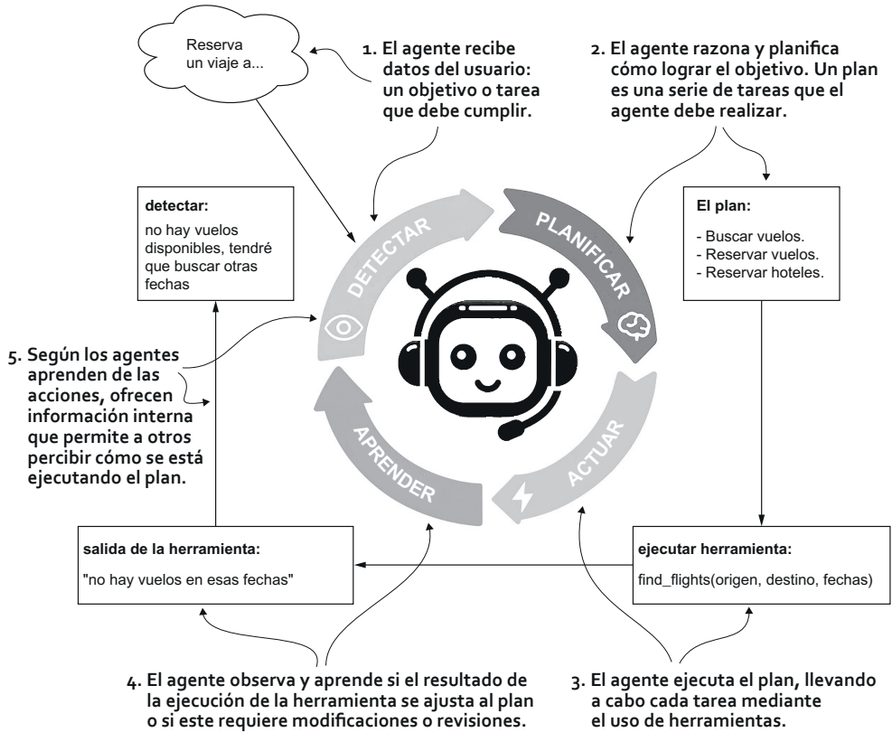


Figura 1.3. El proceso de cuatro pasos que siguen los agentes para completar sus objetivos: detectar (recibir información, como un objetivo o una retroalimentación) > planificar (definir las tareas necesarias para lograr el objetivo) > actuar (ejecutar la herramienta definida por la tarea) > aprender (observar el resultado de la tarea y determinar si el objetivo se ha alcanzado o si el proceso debe continuar).

En el caso de los primeros LLM (como GPT-4), teníamos que recurrir a técnicas de ingeniería de *prompts* para conseguir que el modelo razonara paso a paso, creara un plan y ejecutara las tareas asociadas. Hoy en día, la mayoría de los modelos de vanguardia que se utilizan para crear agentes incluyen cierta capacidad de razonamiento y planificación, que a menudo se manifiesta de forma explícita a través de un modo de pensamiento o razonamiento. Sin embargo, la calidad de ese razonamiento integrado varía considerablemente. Los modelos más pequeños, los más antiguos y aquellos que se utilizan fuera de su distribución de entrenamiento pueden razonar de forma superficial u omitir pasos en tareas complejas, incluso cuando, en teoría, admiten un modo de razonamiento.

El razonamiento integrado reduce en parte la necesidad de introducirlo a través de las instrucciones proporcionadas al agente, pero no elimina dicha necesidad. Como veremos en el capítulo 5, quizá aún te interese utilizar patrones de razonamiento para mejorar, estructurar y controlar el pensamiento del agente, especialmente cuando la tarea tiene un horizonte a largo plazo o el modelo subyacente no es un razonador de frontera.

1.1.3. Los agentes actúan con herramientas

OpenAI y ChatGPT establecieron un primer modelo para el uso de herramientas LLM, modelo que ha sido adaptado a los agentes. Es necesario indicar a un LLM o a un agente cómo utilizar las herramientas, incluyendo cómo definir las entradas de una herramienta y gestionar sus salidas. Las herramientas en sí mismas son extensiones de funciones, y a menudo nos referimos a ellas como "funciones de herramienta". La figura 1.4 ilustra cómo se puede configurar un agente para que utilice y, a continuación, invoque una función de herramienta.

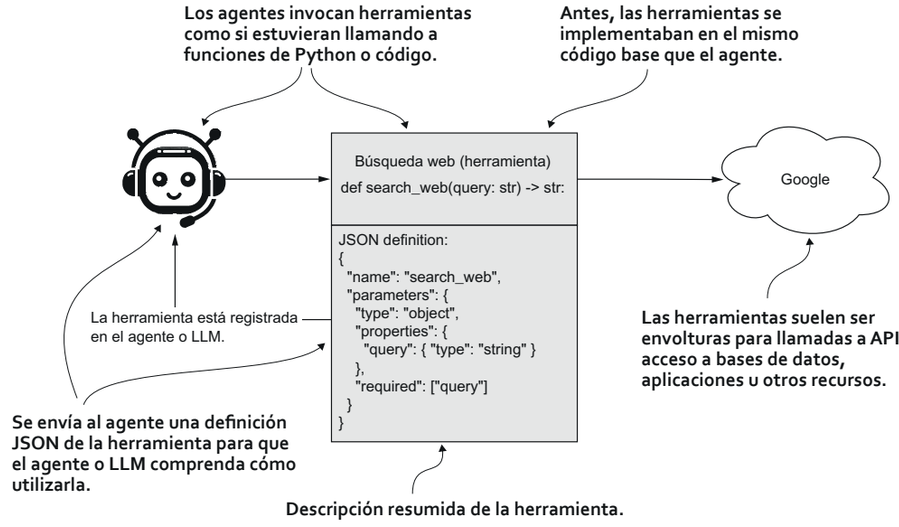


Figura 1.4. Para utilizar una herramienta, un agente debe registrarla primero como una descripción o definición en formato JSON. Una vez registrada, el agente utiliza la herramienta mediante un proceso muy similar a la llamada a una función en Python.

Las herramientas suelen encapsular llamadas a API, bases de datos, aplicaciones externas u otros recursos. Permiten al agente actuar más allá de su propio código e interactuar con sistemas externos. Hay diversos marcos de trabajo de agentes que permiten registrar una herramienta mediante la anotación de la función subyacente, por ejemplo, con un decorador de Python, un decorador de TypeScript o una entrada de configuración, dependiendo del marco, de modo que el agente sepa cómo y cuándo llamarla.

Las herramientas también pueden fallar, y la forma en que un agente gestiona esos fallos es parte de lo que distingue una demostración de un sistema de producción. Las API agotan el tiempo de espera, las bases de datos devuelven esquemas inesperados, se alcanzan los límites de frecuencia, los servicios se caen y los argumentos llegan con errores de formato. Un agente bien diseñado trata los fallos de las herramientas como parte del flujo de control normal, en lugar de como fallos del sistema. Transmite el error al modelo, permite que el

7 Creación de agentes robustos mediante evaluación y retroalimentación

En este capítulo veremos cómo:

- Presentar la evaluación y retroalimentación agénticas.
- Implementar el desarrollo de agentes dirigido por pruebas.
- Utilizar agentes de anclaje, agentes críticos y de evaluación.
- Emplear Phoenix para evaluación y retroalimentación.

La evaluación y retroalimentación aportan la disciplina que permite medir y mejorar la robustez de los agentes. Por sí solas, no generan robustez; un agente mal diseñado fallará de formas que ningún conjunto de pruebas de evaluación podrá solucionar. La evaluación y retroalimentación ofrecen un mecanismo para progresar hacia un mejor comportamiento con el tiempo, además de la posibilidad de ser testigos de cómo se comporta realmente el sistema.

La evaluación de los agentes adopta muchas formas, desde pruebas comparativas y de equipo rojo hasta comprobaciones de anclaje y agentes que evalúan a otros agentes. Los sistemas de retroalimentación desarrollados para los agentes provienen de revisores humanos, agentes evaluadores, resultados de pruebas y autoevaluación. Cada uno responde a una pregunta diferente sobre el agente y ocupa un papel distinto en el ciclo de vida del desarrollo.

Aunque la implementación de la evaluación y retroalimentación suele ser un requisito en cualquier sistema de agentes en producción, este no debería ser el único momento en el que te plantees reforzar la seguridad de tus sistemas agénticos. Casi siempre es recomendable incorporar esta última capa (la quinta capa de la figura 3.7, es decir, evaluación y retroalimentación), tema que trataremos en este capítulo.

7.1. Presentación de la evaluación y retroalimentación agénticas

En esta sección ampliamos la perspectiva y describimos las diferentes evaluaciones y ciclos de retroalimentación que pueden existir en torno a un agente (pruebas, humanos o evaluadores automatizados). Los agentes se basan en un mecanismo interno de evaluación y retroalimentación que se activa en la fase de aprendizaje del ciclo SPAL (detectar-planificar-actuar-aprender). Además, podemos proporcionar mecanismos externos que evalúen y proporcionen retroalimentación al agente o a nosotros mismos. La figura 7.1 destaca los distintos pasos y sistemas que pueden utilizarse para la evaluación y retroalimentación del agente: pruebas comparativas y de equipo rojo, evaluadores de anclaje o de agentes críticos, y evaluación humana.

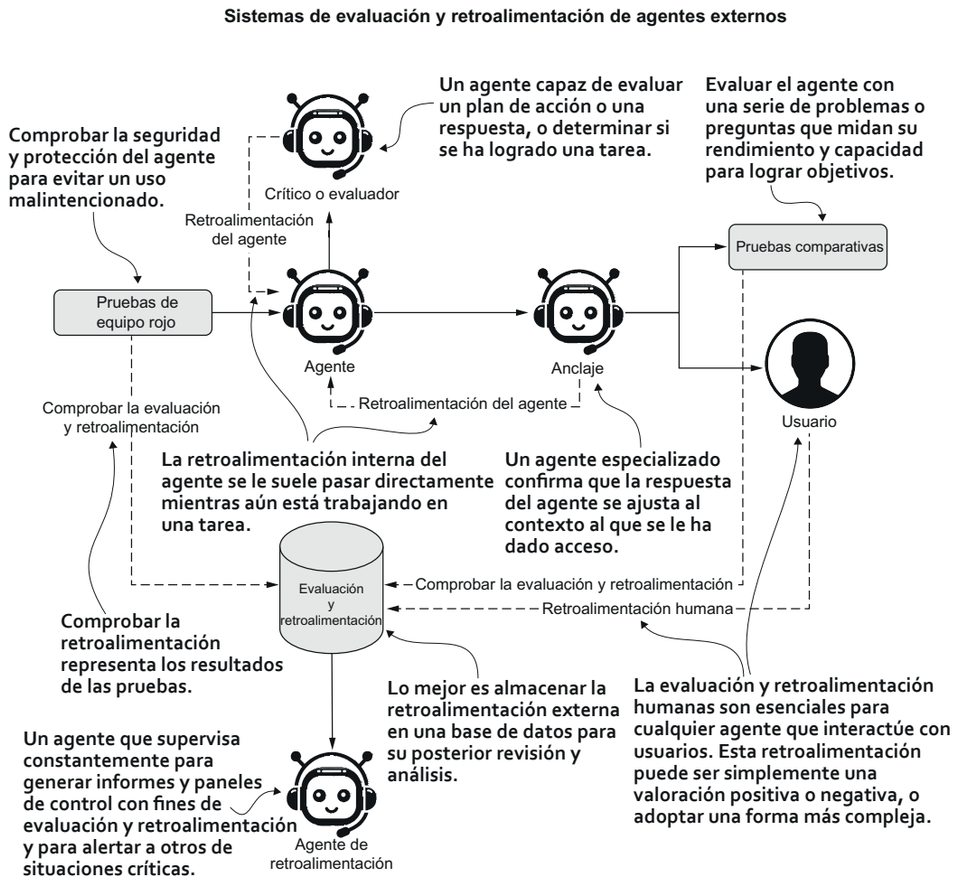


Figura 7.1. Los sistemas de evaluación y retroalimentación de agentes externos para un único agente que interactúa con el usuario. En la figura, un agente generativo puede ser evaluado por un crítico, que le proporciona retroalimentación general sobre sus resultados, o por un agente de anclaje, que le ofrece retroalimentación y evaluación específicas del contexto.

Cada tipo de evaluación puede proporcionar retroalimentación a través de una de dos vías. Puede llegar directamente al agente a través de agentes críticos, comprobaciones de anclaje u otros evaluadores integrados en el bucle que influyen en la siguiente acción del agente, o bien puede dirigirse a una base de datos de evaluación que almacena los resultados de las pruebas y la retroalimentación humana para su posterior análisis, detección de regresiones y como señal de entrenamiento.

La evaluación en sí misma adopta tres formas principales, cada una adecuada para diferentes tipos de corrección. Los casos de prueba deterministas se ocupan de salidas estructuradas, en las que la corrección puede comprobarse con exactitud (el agente devolvió el esquema esperado, invocó la herramienta correcta o generó una salida que coincide con una respuesta válida conocida). Los evaluadores agénticos y de grandes modelos de lenguaje se ocupan de juicios más difusos en lenguaje natural, en los que la corrección es una cuestión de grado más que de coincidencia (puntuación según una rúbrica, clasificación o comparación de dos salidas en cuanto a calidad relativa). Los evaluadores humanos se encargan de los casos que los otros dos pasan por alto: casos límite, cuestiones relacionadas con políticas, problemas de tono y cualquier aspecto que requiera un juicio basado en el mundo real.

A partir de esta base de datos, podemos incorporar un agente de retroalimentación especializado, cuya función es revisar constantemente las evaluaciones y los comentarios. Este agente puede generar informes y paneles de control o enviar alertas críticas. Aunque no todos los sistemas requieren todas las formas de evaluación y retroalimentación, por lo general es recomendable emplear los siguientes métodos cuando sea apropiado:

- **Pruebas de equipo rojo o *red teaming*:** Evalúan si determinados términos, frases o patrones pueden provocar fallos en el sistema. Cualquier agente o sistema de agentes orientado al usuario debería someterse a alguna prueba de este tipo.
- **Pruebas comparativas:** Evalúan la eficacia con la que un agente resuelve un problema o logra un objetivo. Estas pruebas exhaustivas están pensadas para abarcar sistemas de agentes completos, pero también pueden utilizarse para evaluar agentes individuales.
- **Retroalimentación humana:** Cualquier sistema de agentes orientado al usuario debería ofrecer a los usuarios una forma de dar su opinión mediante valoraciones positivas o negativas, comentarios, o ambas cosas. La retroalimentación humana es la base sobre la que se evalúan y mejoran los sistemas de agentes reales en producción; no se trata de un aporte complementario. Sin embargo, la retroalimentación sin procesar contiene ruido, así que debe ser tratada antes de que pueda servir de base para la toma de decisiones. La agregación estadística, la detección de valores atípicos para identificar patrones coordinados o anómalos, el muestreo estratificado que pondera a los usuarios expertos frente a los ocasionales, y la revisión de seguimiento de los resultados señalados son técnicas prácticas para separar la señal del ruido.
- **Agente de anclaje o *grounding*:** Este agente aplica la evaluación de anclaje a los sistemas RAG, comprobando si una respuesta está fundamentada en el contexto que el agente de conocimiento ha recuperado. El anclaje en sí mismo es una técnica general para verificar que la salida del modelo se remonta a una fuente verificable, y se aplica más allá de RAG a la validación de llamadas a herramientas, a la verificación de la memoria y a la auditoría de citas. El agente de anclaje es la implementación; el

anclaje es la técnica. Toma la fuente y el resultado y comprueba si las afirmaciones del resultado están respaldadas por la fuente. Aplícalo siempre que el agente disponga de una fuente fidedigna a la que remitirse.

- **Agente de evaluación o agente crítico:** Este agente proporciona un nivel adicional de evaluación y retroalimentación para el razonamiento interno durante la planificación, la determinación de la consecución de objetivos y la comprensión del resultado de las acciones. Aunque este tipo de agente puede añadirse a cualquier flujo de trabajo agéntico, se utiliza habitualmente en los patrones multiagente de flujo y colaboración.

Es posible que te encuentres con otros patrones de evaluación y retroalimentación que se derivan de estos métodos o los amplían. No esperes hasta las últimas fases del desarrollo para aplicar esta quinta capa. Al igual que hacer pruebas impulsa el desarrollo dirigido por pruebas, estas mismas prácticas se aplican a la hora de crear agentes.

Gobernanza de los sistemas multiagente

Los sistemas multiagente presentan una clase de modos de fallo que los sistemas de agente único no tienen. El más preocupante es la colusión entre agentes: una situación en la que varios agentes refuerzan mutuamente las respuestas erróneas porque comparten supuestos, sesgos o datos de entrenamiento, y ningún control externo rompe ese bucle. Un agente de anclaje que utilice el mismo modelo que el agente al que está sustentando tenderá a validar las mismas alucinaciones que produjo el agente original. Existen varias prácticas que reducen el riesgo de colusión. Utiliza modelos diferentes para los agentes evaluadores que para los evaluados, ya que los modelos de distintos proveedores tienden a fallar en direcciones diferentes, y un evaluador con los mismos datos de entrenamiento comparte los mismos puntos ciegos. Cuando no resulte viable utilizar modelos diferentes, utiliza *prompts* y perfiles distintos para orientar al evaluador hacia una postura diferente (un evaluador al que se le pide que encuentre problemas se comporta de manera distinta a uno al que se le pide que confirme la corrección de la información, incluso con el mismo modelo). Incluye periódicamente la revisión humana de los resultados señalados y de una muestra aleatoria de los no señalados, ya que un sistema de evaluación automatizado que nunca se compara con el dato verdadero de referencia puede desviarse sin que se detecte.

La gobernanza más allá de la colusión incluye algunas medidas estándar que merece la pena mencionar. Define una jerarquía de autoridad clara (qué agente tiene la última palabra cuando los evaluadores no se ponen de acuerdo). Registra todos los mensajes entre agentes para que los desacuerdos y las decisiones sean auditables *a posteriori*. Establece reglas de derivación para los casos en los que los evaluadores no se pongan de acuerdo o en los que la confianza caiga por debajo de un umbral, con una revisión humana como último recurso. Limita la autoridad de cada agente a lo que requiere su función, de modo que un solo agente comprometido o que funcione mal no pueda provocar un efecto en cadena en las decisiones de todo el sistema.

Estas prácticas no son exclusivas de los agentes evaluadores, pero se manifiestan con mayor intensidad en este ámbito, ya que la evaluación es donde reside la medida que el sistema utiliza para valorar su propia corrección. Una capa de evaluación que actúe en connivencia con los agentes que evalúa es peor que la ausencia de dicha capa, ya que proporciona una confianza falsa. Considerar la gobernanza como parte del diseño de la evaluación, y no como una cuestión independiente añadida posteriormente, constituye el marco más sólido.

7.2. Implementación del desarrollo de agentes dirigido por pruebas

El desarrollo dirigido por pruebas o TDD (*Test-Driven Development*) es una práctica consolidada de ingeniería de software, en la que se escriben pruebas para expresar requisitos antes de la implementación que los satisface. A decir verdad, este método suele ser más una aspiración que una norma universal: muchos equipos de trabajo escriben las pruebas después de lograr un producto mínimo viable o MVP, sobre todo bajo la presión de los plazos, y esta disciplina resulta más útil como hábito que como regla. Lo que sí es cierto es que definir cómo debe ser el resultado final antes de crearlo da lugar a sistemas más limpios que la alternativa.

El mismo principio se aplica a los agentes. Definir cómo debe ser una buena respuesta del agente y, a continuación, construirlo para que cumpla esa definición es más disciplinado que construir el agente y esperar que la evaluación detecte los problemas más adelante. Como se muestra en la figura 7.2, este enfoque se conoce como desarrollo de agentes dirigido por pruebas (TDAD, *Test-Driven Agent Development*).

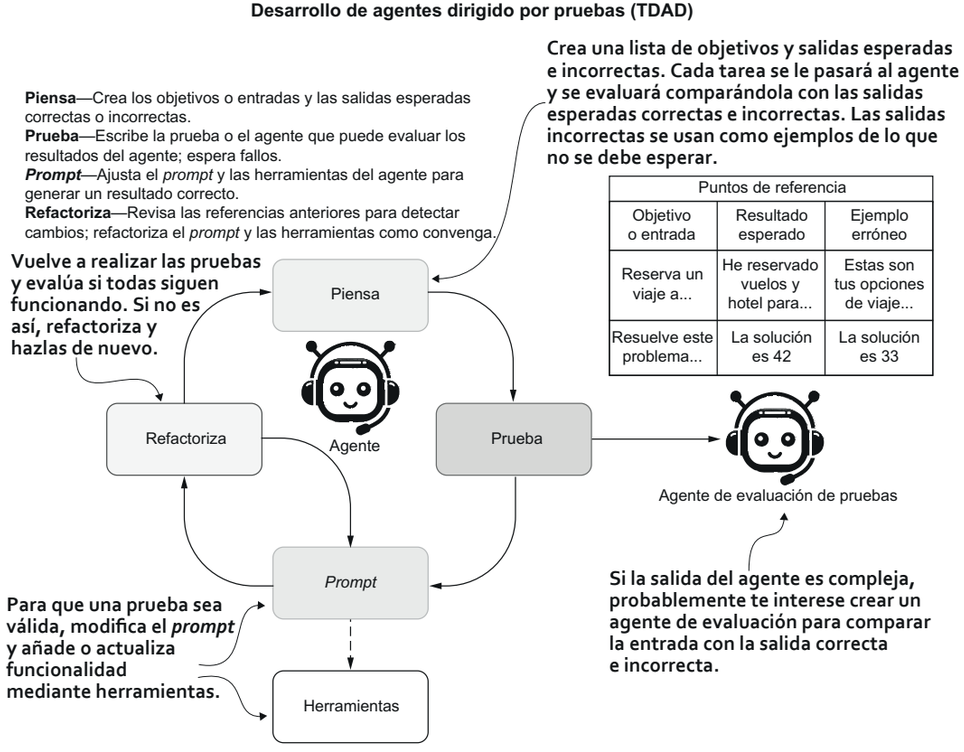


Figura 7.2. El proceso humano de desarrollo de agentes TDAD: primero, definimos los objetivos y los puntos de referencia; después, realizamos pruebas, desarrollamos el *prompt*, refactorizamos y repetimos el proceso.

La versión para agentes del aspecto que ha de tener lo correcto suele expresarse mediante rúbricas en lugar de pruebas unitarias. Las rúbricas especifican los criterios que debe cumplir una respuesta (precisión factual, anclaje en el contexto recuperado, tono adecuado, corrección estructural), y el proceso de evaluación comprueba la salida del agente en función de dichos criterios. Volveremos a las rúbricas más adelante en este capítulo, donde se concretará la conexión entre TDAD y la evaluación basada en rúbricas.

La figura ilustra el proceso humano de creación de un agente (*prompts* y herramientas) siguiendo el proceso TDAD. Este proceso comienza con reflexión, prueba, formulación de *prompts* y refactorización. En TDAD, primero establecemos un conjunto de objetivos o criterios de referencia de entrada con los que se evaluará al agente, identificando las respuestas esperadas y las conocidas como erróneas para cada uno de ellos.

La evaluación del agente va más allá de la comparación entre entradas y salidas. Una prueba completa observa al agente de forma holística: qué herramientas seleccionó, qué pasos de razonamiento siguió, si se mantuvo fiel al contexto recuperado, cuántos pasos tardó en llegar a una respuesta, cuánta latencia y coste acumuló la ejecución y cómo se comportó cuando algo falló. Cada uno de estos aspectos es una propiedad que puede comprobarse, y cualquiera de ellos puede ser erróneo incluso aunque la salida final parezca correcta.

Los ejemplos de resultados incorrectos y comportamientos intermedios erróneos ayudan a los agentes de evaluación a distinguir las ejecuciones correctas de las incorrectas. El mismo agente que ha generado una respuesta correcta tras cinco llamadas a herramientas erróneas no es el mismo que el que la ha generado tras dos llamadas correctas, y una prueba completa detectará esa diferencia.

A continuación, realizamos las pruebas, y al inicio de las mismas debemos esperar que el agente falle. Dado que los grandes modelos de lenguaje que impulsan a los agentes son variables, ejecuta cada prueba varias veces para confirmar que el fallo es constante y no se trata de un caso aislado. A partir del fallo, identifica las actualizaciones mínimas que hay que realizar en el *prompt* del agente y en las herramientas para superar la prueba de rendimiento.

Vale la pena definir el término "mínimo", ya que es el aspecto en el que la mayoría de los profesionales se equivocan. La progresión que funciona en la práctica, de menor a mayor cambio, es la siguiente: cambiar una sola palabra para ajustar el tono o la precisión, añadir una cláusula a una frase existente, añadir una sola frase a una sección existente, añadir una nueva sección al *prompt*, añadir o modificar una herramienta y cambiar el modelo o sus parámetros. Prueba primero con los cambios más pequeños y solo ve aumentando cuando estos no solucionen el fallo.

Hacer lo mínimo mantiene las instrucciones concisas, preservando el rendimiento del modelo y reduciendo la latencia en cada llamada. Los *prompts* sobrecargados surgen de equipos de trabajo que añaden un párrafo cada vez que algo sale mal, nunca eliminan nada y acaban con instrucciones del sistema de varias páginas, en las que cada sección se contradice con otra. La disciplina no consiste solo en realizar el cambio más pequeño, sino en eliminar aquello que ya no tiene cabida.

Después volvemos a realizar las pruebas y refactorizamos. Ejecutamos de nuevo todas las pruebas de referencia que antes se superaban para confirmar que el agente funciona según lo esperado. Si fallan pruebas que antes se superaban, volvemos atrás y refactorizamos el *prompt*, realizando cambios mínimos para que todas las pruebas se superen. Continuamos refactorizando y probando hasta confirmar que el agente supera todas las pruebas de referencia. Por lo general, querremos que el agente supere todas las pruebas de rendimiento antes de continuar. Sin embargo, no es raro que las pruebas de rendimiento anteriores fallen debido a los cambios realizados para dar soporte a otras pruebas. Es probable que esto ocurra porque una prueba de rendimiento puede contradecir a otra. Por ejemplo, una prueba puede obligar al agente a pensar paso a paso, mientras que otra puede exigirle que responda rápidamente sin pensar. En tales casos, cuando las pruebas de rendimiento empiecen a fallar, quizá quieras plantearte las siguientes preguntas y opciones:

- **¿Deben superarse todas las pruebas de rendimiento?:** Si la respuesta es no, utiliza las pruebas fallidas como ejemplo de resultado incorrecto. Otra razón por la que la respuesta podría ser negativa es que esperes que el agente funcione bien solo en el 90 % de las pruebas establecidas. Esto puede deberse a que tienes cientos de pruebas de rendimiento desarrolladas a lo largo del tiempo, y alcanzar una tasa de superación del 100 % no siempre es factible.
- **¿Tiene sentido añadir un agente?:** Si tu agente supera la mayoría de las pruebas de referencia, pero falla en un subconjunto, una opción es añadir un agente especializado que se ocupe únicamente de esos casos. Esto plantea un problema de enrutamiento (qué agente responde a una entrada determinada), y el propio enrutador puede cometer errores que aumenten la tasa de error global antes de que la especialización resulte rentable. La precisión del enrutamiento es un problema de evaluación en sí mismo.
- **¿Deberías refactorizar todo tu agente?:** Añadir agentes supone mucho más trabajo, y muchas veces tiene más sentido desmontar el agente y reconstruirlo. Afortunadamente, reconstruir un agente suele implicar únicamente recrear su *prompt* desde cero y, esta vez, cuando reconstruyas tu agente, centra las instrucciones en los casos de prueba que causaron más problemas o que son más complejos.

> **NOTA:** El término "mezcla de expertos" dentro de un LLM se refiere a una capa que redirige los *tokens* a subredes en una única pasada de avance, que no es lo mismo que varios agentes con llamadas independientes. La versión con varios agentes se asemeja más al enrutamiento especializado o al ensamble. Vale la pena mantener los términos diferenciados. La especialización aumenta la complejidad, la sobrecarga de evaluación y el coste. Antes de dar ese paso, hay que plantearse si la refactorización del agente único (mejor *prompt*, herramientas y anclaje) permite salvar la brecha. La especialización es la decisión acertada cuando el subconjunto que falla es genuinamente distinto, no cuando el agente único solo requiere una modificación para superar la prueba.

Por lo general, lo más recomendable es aplicar TDAD a todo un sistema de agentes y centrarse en los agentes especializados individuales, donde se realiza la mayor parte del trabajo. Para ello, se necesitan pruebas de rendimiento tanto de los agentes como del sistema agéntico que puedan aplicarse y probarse a diferentes niveles de granularidad. De lo contrario, se corre el riesgo de que surjan lagunas en la funcionalidad que podrían manifestarse durante el uso en producción.

7.2.1. TDAD en la práctica

Veamos el proceso de creación de un agente con TDAD. De este modo, podrás seguir los pasos para crear tú mismo un sistema de único agente o varios. La figura 7.3 muestra un sistema de un solo agente y algunas pruebas de referencias iniciales que cabría esperar que este agente superara. En este ejemplo, nuestras pruebas consisten en un conjunto de preguntas específicas sobre un material de referencia ficticio.

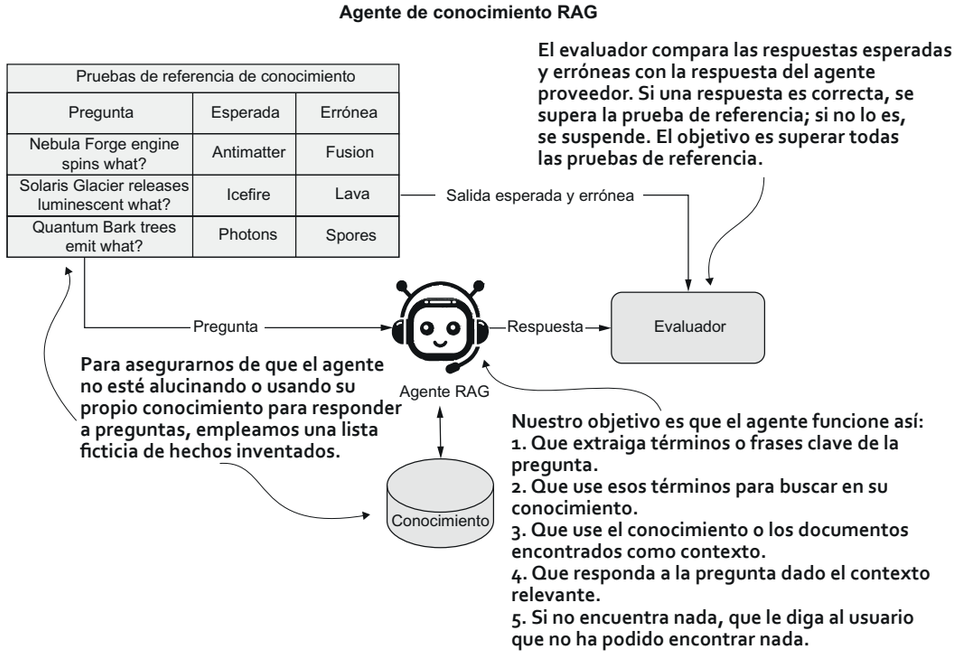


Figura 7.3. Arquitectura de un agente de conocimiento RAG sencillo. El objetivo del agente es responder a las preguntas de una prueba de referencia buscando en su base de conocimientos y devolviendo respuestas. Un evaluador sencillo basado en un LLM evalúa la respuesta según las respuestas esperadas e incorrectas de la prueba de referencia. La precisión del agente RAG puede determinarse mediante el porcentaje de preguntas de la prueba de referencia que responde correctamente.

En la figura, el agente RAG se basa en un patrón sencillo de arquitectura de recuperación, en el que un agente recibe una pregunta sobre algún conocimiento externo al que tiene acceso. A partir de ahí, desglosa la pregunta en palabras y frases y, a continuación, utiliza esos términos para buscar en su base de conocimientos. Si encuentra información, la utiliza como contexto para responder a la pregunta del usuario; de lo contrario, indica que no se ha encontrado nada. A continuación, esta respuesta se envía a un evaluador junto con las respuestas esperadas e incorrectas, emparejadas con la referencia. El evaluador utiliza entonces esta información para determinar si se ha superado o no la prueba de referencia.

7.2.2. Programación y pruebas del agente RAG

Ahora que tenemos una idea de las pruebas de referencia, debemos evaluar nuestro agente a medida que desarrollamos el código mínimo necesario para darles soporte. En este momento, nuestro objetivo es hacer lo suficiente para poder plantearle una pregunta al agente. Este responderá y evaluará esa respuesta. La forma de desarrollar este código depende de ti, y quizá te interese utilizar TDD para los componentes habituales del software, como la conectividad con la base de datos y otras herramientas.

El listado 7.1 muestra el código que ejecutará el agente a través de la tabla de pruebas de referencia. En esta fase, la evaluación es sencilla y utiliza la comparación de cadenas con la respuesta esperada. En cuanto al agente en sí, lo único que hemos definido es el rol o persona en las instrucciones, y nada más. Recuerda que nuestro objetivo con TDAD es crear el conjunto mínimo de instrucciones con el conjunto mínimo de herramientas para superar todos nuestros objetivos de las pruebas de referencia.

Listado 7.1. `01_RAG_agent.py`.

```
# Esta es la base de datos ficticia de hechos que se almacenarán en la memoria
_special_knowledge_db = [
    "Nebula Forge engine spins antimatter rings for gravity control.",
    "Solaris Glacier absorbs heat, releasing luminescent icefire at dusk.",
    "Quantum Bark trees emit entangled photons guiding nocturnal insect migrations.",
    "Aether Silk fabric self-weaves repairs within fourteen-millisecond microtears.",
    "Chrono Coral reefs rewind water currents three minutes every solstice.",
]

# pruebas de referencia para conocimientos especiales
# Las pruebas de rendimiento con las que queremos evaluar al agente
_benchmarks = [
    {"q": "Nebula Forge engine spins what?", "a": "antimatter", "wrong": "fusion"},
    {
        "q": "Solaris Glacier releases luminescent what?",
        "a": "icefire",
        "wrong": "lava",
    },
    {"q": "Quantum Bark trees emit what?", "a": "photons", "wrong": "spores"},
    {"q": "Aether Silk fabric repairs what?", "a": "microtears", "wrong": "threads"},
    {"q": "Chrono Coral reefs rewind what?", "a": "currents", "wrong": "tides"},
]

# La herramienta que queremos que utilice el agente para buscar información
@function_tool
def search_knowledge(query: str) -> dict:
    """Search the knowledge database for relevant facts."""
    matches = [doc for doc in _special_knowledge_db if query.lower() in doc.lower()]
    print(f"Found {len(matches)} matches for '{query}'")
    return {"status": "ok", "context": "\n".join(matches)}

# El código del agente, que incluye proporcionarle la herramienta de búsqueda
agent = Agent(
    name="RAG Agent",
    instructions="""
You are a retrieval-augmented knowledge agent.
""",
```

```
    tools=[search_knowledge],
)

for benchmark in _benchmarks:
    question = benchmark["q"]
    answer = benchmark["a"]
    wrong_answer = benchmark["wrong"]
    result = asyncio.run(Runner.run(agent, input=question)).final_output.strip()
    print("" + "=" * 40)
    print(f"Question: {question}")
    print(f"Answer: {result}")
    if result.lower() == answer.lower():
        print(f"Correct -> {answer}")
    else:
        print(f"Incorrect -> Expected: {answer}")
        if wrong_answer:
            print(f"Wrong answer was: {wrong_answer}")
```

Recorre cada prueba de rendimiento para formular preguntas y evaluar las respuestas

Si ejecutas el código verás que, como era de esperar, el agente fallará en todas las pruebas. Si supera alguna, no te precipites a celebrar antes de realizar unas cuantas pruebas más para confirmar que el resultado es coherente. Por regla general, es recomendable ejecutar cada prueba superada tres veces para estar razonablemente seguro de la coherencia del agente.

> **NOTA:** El hecho de tener que ejecutar pruebas de rendimiento varias veces contrasta con TDD, donde basta con una sola pasada. Por desgracia, dado que los LLM son variables por naturaleza, nos obligan a ejecutar pruebas de rendimiento varias veces durante el desarrollo, al menos tres veces y, a menudo, más. Un indicador clave del número de iteraciones que hay que llevar a cabo es la variabilidad en la salida del agente. Por supuesto, todo esto se puede automatizar e incluir el almacenamiento fuera de línea de las ejecuciones de evaluación.

7.2.3. Refactorización del agente

Ahora que contamos con un sistema de agentes totalmente operativo que no supera ninguna de las pruebas de referencia, podemos empezar a refactorizar y a elaborar los *prompts* para el agente. Queremos empezar por buscar soluciones sencillas, como corregir el formato de la salida o proporcionar instrucciones para gestionar los datos. Eso es precisamente lo que hace el siguiente fragmento de código.

Listado 7.2. `02_RAG_agent.py` **(solo el agente).**

```
agent = Agent(
    name="RAG Agent",
    instructions="""
You are a retrieval-augmented knowledge agent.
Break down the user's query into smaller parts if needed
to fetch relevant context for the user's query.
Always answer with a single word.
""",
    tools=[search_knowledge],
)
```

Ofrecemos instrucciones más claras sobre cómo buscar contexto

Indica al agente que responda siempre con una sola palabra

Lo que hemos hecho en este listado es asegurarnos de que las respuestas siempre sean de una sola palabra, en consonancia con nuestro formato de referencia. A continuación, ofrecemos instrucciones para desglosar la pregunta, pero, por ahora, evitamos dar más específicas sobre el uso de herramientas. A modo de orientación, queremos evitar incluir en el *prompt* instrucciones de formato y de uso de herramientas.

Regla de diseño de prompts *para agentes*

Antes, el desarrollo de los agentes se basaba en *prompts* que enumeraban explícitamente todas las herramientas y especificaban el formato de salida con todo detalle. La práctica actual consiste en registrar los metadatos de las herramientas (nombres, descripciones, esquemas de parámetros) a través del SDK, y el modelo utiliza esos metadatos para elegir las herramientas sin que el *prompt* tenga que repasar cada una de ellas. El cambio radica en que los *prompts* ya no necesitan repetir lo que ya indican los metadatos de las herramientas.

Sin embargo, las instrucciones sobre las herramientas no han desaparecido. Los archivos `AGENT.md`, las secciones de *prompts* del sistema y las orientaciones integradas siguen siendo habituales cuando determinadas herramientas requieren un tratamiento específico (utilizar esta herramienta solo para consultas jurídicas y dar prioridad a la herramienta local frente a la remota). El cambio radica en la responsabilidad, no en el requisito.

El formato de salida sigue el mismo patrón. Las bibliotecas de salida estructurada permiten declarar la forma de la respuesta (un modelo Pydantic o un esquema JSON), y el SDK expande esa declaración en el *prompt* y aplica la validación. El descubrimiento dinámico es cómodo, no mágico: el formato sigue apareciendo en el *prompt*, solo que expandido por el SDK en lugar de escrito a mano.

Las instrucciones de formato y uso de herramientas deben proporcionarse fuera del *prompt*. En cuanto al formato, ello significa establecer el tipo de salida en el agente y, a continuación, construir la salida deseada a partir de ese formato con tipos de datos bien definidos. En cuanto a las herramientas, la descripción de la herramienta y el comentario al principio de la función deben proporcionar la explicación que el agente necesita para utilizar la herramienta.

Mantener los detalles de formato y herramientas fuera del *prompt* facilita que el agente se adapte a nuevas herramientas o formatos de salida sin tener que reescribir el texto. Se trata de una separación clara, no de una norma en contra de los *prompts* largos. Los *prompts* del sistema largos y específicos son habituales en producción, especialmente para agentes que deben evitar ciertos comportamientos, hacer cumplir políticas u operar en ámbitos regulados; los que proporcionan los principales proveedores de LLM suelen tener varios miles de *tokens*.

Cada vez que se modifican las instrucciones de un agente, este puede comportarse de forma diferente. El riesgo varía en función de lo que se cambie: endurecer una restricción suele tener efectos predecibles, mientras que añadir una nueva sección o modificar el tono puede interactuar con el resto del *prompt*, de manera que solo se ponga de manifiesto durante la evaluación. Lo más acertado no es evitar los cambios en los *prompts*, sino evaluar el impacto de cada cambio con respecto a los criterios de referencia antes de considerar el nuevo *prompt* como base.

“Excelente contenido, muy completo y con ejercicios entretenidos. Este libro lo tiene todo”.

—Saurabh Sawant, Microsoft

Exhaustiva guía práctica para crear agentes de IA; no solo para comprender qué son, sino también para diseñarlos, implementarlos, evaluarlos y ponerlos en marcha. Su punto fuerte radica en la forma en que combina la claridad conceptual con ejemplos de código funcional, de modo que se aprende de forma progresiva en lugar de asimilar ideas aisladas. Los ejemplos conforman un itinerario de aprendizaje continuo, desde un agente mínimo hasta sistemas multiagente más capaces, que utilizan herramientas y que pueden desplegarse.

Comienza ofreciendo a los lectores un modelo mental útil para el diseño de agentes. Su idea central incluye las cinco capas funcionales: persona, herramientas y acciones, razonamiento y planificación, conocimiento y memoria, y evaluación y retroalimentación. Esta estructura ayuda a comprender de dónde proviene el comportamiento de un agente y cómo diagnosticar sus puntos débiles. En lugar de añadir de forma aleatoria prompts, herramientas o memoria, invitando a preguntarse qué capa necesita mejora, lo cual resulta especialmente valioso, porque el modelo no está vinculado a un único proveedor o marco de trabajo; de ahí que siga siendo útil incluso aunque las API y las herramientas sigan cambiando.

También aborda los componentes básicos prácticos de los agentes: los grandes modelos de lenguaje (LLM), la ingeniería de prompts, las salidas tipadas, el rastreo, el uso de herramientas y el SDK de agentes de OpenAI.

La obra definitiva para crear agentes más predecibles, en los que poder confiar y mantener.

"Una de las introducciones más prácticas al desarrollo de agentes que existe en la actualidad".
—Ajay Prakash, LinkedIn

"Un libro excepcional para quienes se inician en el campo del desarrollo de agentes de IA".
—Roger Meli, IT Architect

Lectores a los que está destinado

Dirigido a programadores de Python con un nivel intermedio. No se requiere experiencia previa con agentes de IA ni con sistemas agénticos.

Sobre el autor

Micheal Lanham es un innovador en el ámbito del software y la tecnología, con más de 20 años de experiencia en el sector. Es autor de varios libros sobre aprendizaje profundo, entre ellos *Evolutionary Deep Learning*, de la editorial Manning.

ISBN 978-84-415-5313-2
2315295
9 788441 553132

ANAYA MULTIMEDIA
www.anayamultimedia.es