

Unidad 7 - Agentes Autónomos y Sistemas Inteligentes

UNR - TUIA - Procesamiento de Lenguaje Natural

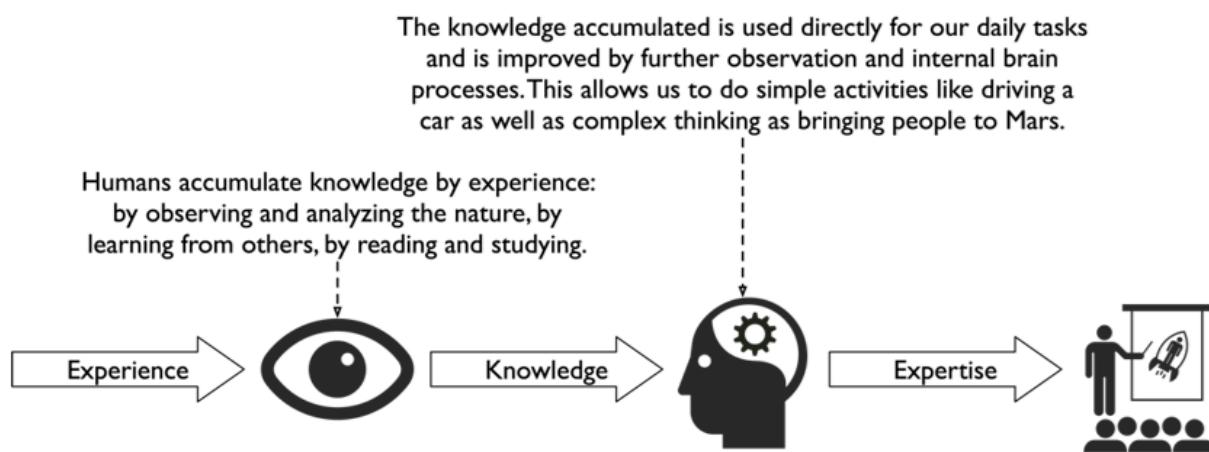
Docente teoría: Juan Pablo Manson - jpmanson@gmail.com - [LinkedIn](#)

1. Sistemas Inteligentes

Comencemos con una pregunta simple pero importante: *¿Qué es la inteligencia?* Al responder esta pregunta, obtendremos una comprensión fundamental de lo que es un sistema inteligente. Sin buscar una respuesta compleja en libros psicológicos o filosóficos, cualquier diccionario define la inteligencia como la capacidad de adquirir y aplicar conocimiento. Por lo tanto, un aspecto clave es aprender de la experiencia: adquirir, retener y usar el conocimiento para resolver problemas, interactuar con el entorno o pensar de forma abstracta. Este aspecto es crítico porque para que cualquier agente (humano, animal, electrónico o mecánico) actúe intelligentemente requiere construir y usar conocimiento. Según Russell y Norvig, un agente es algo que actúa. (Agente proviene del latín *agere*, que significa hacer.) Todos los programas informáticos hacen algo, pero se espera que los agentes informáticos hagan más: operar de forma autónoma,

percibir su entorno, persistir durante un período prolongado, adaptarse al cambio y crear y perseguir objetivos.

Pensemos en nosotros como seres humanos. Para una amplia gama de actividades, tomamos decisiones basadas en lo que sin esfuerzo y de manera inconsciente sabemos (o creemos) sobre el mundo. Nuestro comportamiento (inteligente) está claramente condicionado, si no dominado, por el conocimiento. La siguiente figura muestra cómo nosotros, como seres humanos, aprendemos de la experiencia:



En nuestro caso (como seres humanos), no nos preocupa mucho cómo recopilamos el conocimiento y cómo lo representamos en nuestros cerebros o cómo podemos convertir ese conocimiento en inteligencia. Este es un proceso natural, que es el resultado de la evolución humana. Esto nos da una ventaja competitiva sobre otras especies, y la selección natural hizo el resto (Darwin, el Origen de las especies). Desde el punto de vista del NLP, nos interesa diseñar un sistema inteligente básico que sea capaz de:

- Recopilar y representar eficazmente el conocimiento.
- Razonar autónomamente a partir de ese conocimiento.
- Responder preguntas simples y complejas y tomar acciones, apoyando decisiones informadas.

Diseño de un sistema inteligente

Para definir qué es un sistema inteligente, exploraremos una arquitectura de alto nivel e identificaremos el aspecto más crítico de su desarrollo. Para hacer eso, consideremos una tarea concreta. Supongamos que te han encomendado crear un sistema de diagnóstico médico autónomo donde el paciente con una enfermedad visita al médico. El médico debe elegir una secuencia de acciones (por ejemplo, consultas, pruebas médicas, terapias intentadas) para diagnosticar la enfermedad y curar al paciente.

Esta es una tarea extraordinariamente compleja, pero no te preocupes, no vamos a resolver el problema tan pronto. Usamos este escenario para entender qué tipo de sistema vamos a construir, cómo puede estructurarse y los diferentes tipos de enfoques.

Un sistema de diagnóstico médico autónomo es uno de los ejemplos más comunes de un sistema inteligente. La literatura de inteligencia artificial (IA) utiliza con frecuencia tales sistemas, a veces como agentes inteligentes (refiriéndose a entidades artificiales autónomas que pueden realizar tareas complejas) o como sistemas expertos. La palabra sistema (en lugar de agente) enfatiza la presencia de múltiples componentes que deben combinarse para crear un sistema inteligente. Saber cómo lograr esta combinación de manera eficiente es uno de los aspectos clave en el desarrollo de este tipo de sistemas.

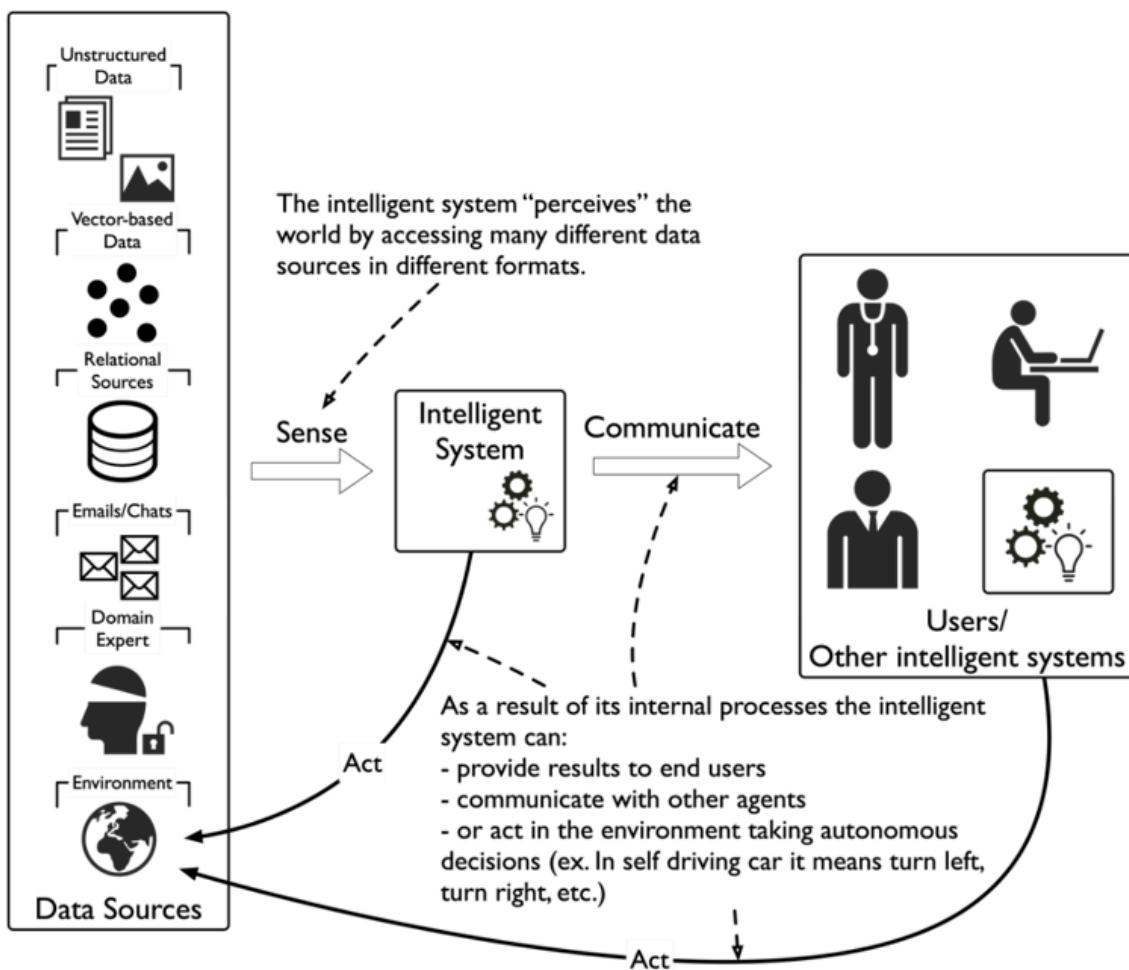
¿Qué es un sistema inteligente?

Comencemos definiendo adecuadamente qué es un sistema inteligente y cuáles son sus características clave. Entre las diferentes definiciones existentes de sistemas inteligentes, aquí hay una de Geoff Hulten:

"Los sistemas inteligentes conectan a los usuarios con la inteligencia artificial (IA) y el aprendizaje automático (ML) para lograr objetivos significativos. Un sistema inteligente es aquel en el cual la inteligencia evoluciona y mejora con el tiempo, particularmente cuando mejora observando cómo los usuarios interactúan con el sistema." (<https://github.com/sivagurunarayareddy/Free-Machine-Learning-Books/blob/master/book/Building Intelligent Systems - A Guide to Machine Learning Engineering.pdf>)

Esta definición destaca dos puntos:

1. El papel central asignado al usuario. El objetivo principal es apoyar a los usuarios en la realización de tareas complejas, donde a veces sugerimos o a veces actuamos en nombre de los usuarios. El sistema de diagnóstico médico autónomo, por ejemplo, tiene como objetivo apoyar a los médicos en lugar de reemplazarlos. Esto es diferente a otros tipos de sistemas inteligentes como, por ejemplo, los automóviles autónomos, donde el objetivo es hacer que las máquinas sean independientes del usuario (el conductor en este caso, aunque solo sea un pasajero).
2. En segundo lugar está la capacidad de aprender de la interacción con el usuario. El sistema debe crear, usar y mantener una base de conocimiento que debería evolucionar con el tiempo, utilizando no solo nuevas fuentes de datos o actualizadas, sino también empleando las interacciones del usuario. La siguiente presenta una visión general de alto nivel de un sistema inteligente que muestra la principal fuente de conocimiento y la manera en que interactúa con los usuarios.



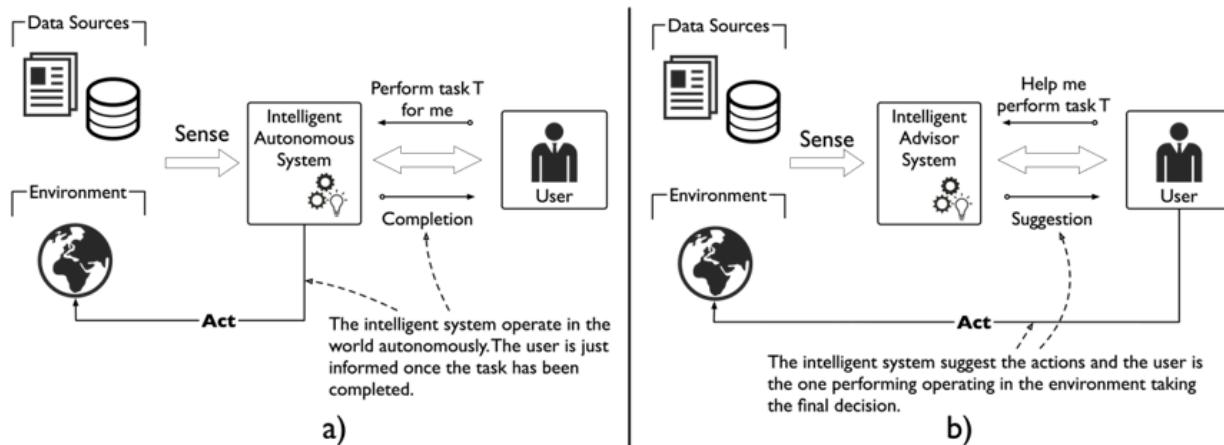
Una visión general de alto nivel de un sistema inteligente y sus interacciones con otros elementos. Un sistema inteligente adquiere conocimiento observando el entorno o ingiriendo datos de fuentes existentes. Los procesos internos elaboran el conocimiento y proporcionan resultados al usuario final. Estos resultados pueden ser sugerencias o acciones concretas.

Cabe señalar que en algunos escenarios podemos tener múltiples sistemas inteligentes comunicándose entre sí para llevar a cabo tareas aún más complejas. Pensemos, por ejemplo, en un sistema multiagente. Es un sistema que consta de varios agentes (entidades independientes) que interactúan entre sí, típicamente intercambiando mensajes a través de una infraestructura de red informática para lograr tareas complejas. Para que estos sistemas interactúen con éxito, estos agentes requieren la capacidad de cooperar, coordinar y negociar entre sí. (Ver: <https://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/imas/IMAS2e.html>)

El aspecto de apoyar a los usuarios versus actuar en nombre de los usuarios marca la distinción entre dos categorías de sistemas inteligentes. Martin Molina

presenta una taxonomía completa de sistemas inteligentes, dividiéndolos entre sistemas autónomos inteligentes y sistemas asesores inteligentes. En un sistema autónomo inteligente, la máquina realiza tareas, reemplazando al usuario. Un coche autónomo pertenece a esta categoría. Un sistema asesor inteligente ayuda al usuario a actuar en el mundo. En este caso, el usuario es quien toma las decisiones finales sobre qué acciones tomar, y el papel del sistema es dar consejos proporcionando información útil para facilitar decisiones. Esta información puede generarse utilizando métodos analíticos descriptivos o predictivos (por ejemplo, reconocimiento de patrones, métodos de diagnóstico o proyección temporal). Además, el sistema puede usar métodos prescriptivos (por ejemplo, asignación de recursos, programación o planificación) para recomendar qué acciones deben tomarse.

En nuestro escenario, queremos que el sistema ayude a los médicos a tomar la acción correcta al identificar la enfermedad, proporcionando una lista concreta de acciones para tratar al paciente. La siguiente figura representa las principales diferencias entre los dos tipos de sistemas:



Diferencias entre a) un sistema autónomo inteligente y b) un sistema asesor inteligente. El primero actúa en nombre de los usuarios, mientras que el segundo solo hace sugerencias.

2. Razonamiento

Quizás una de las tareas más difíciles para un LLM hoy en día es aquella que requiere alguna forma de razonamiento. El razonamiento es una de las áreas más interesantes, debido a los tipos de aplicaciones complejas que pueden surgir de los LLMs.

Ha habido algunas mejoras en tareas que involucran capacidades matemáticas. Dicho esto, es importante señalar que los LLMs actuales luchan por realizar tareas de razonamiento, por lo que esto requiere técnicas de ingeniería de *prompts* aún más avanzadas. Por ahora, cubriremos algunos ejemplos básicos para mostrar capacidades aritméticas.

Prompt:

¿Cuánto es $9.000 * 9.000$?

Salida:

El resultado de multiplicar 9,000 por 9,000 es 81,000,000

Probemos algo más difícil:

Prompt:

Los números impares de este grupo suman un número par: 15, 32, 5, 13, 82, 7, 1.

A:

Salida:

No, los números impares de este grupo suman un número impar: 119.

¡Incorrecto! Intentemos mejorar esto mejorando el *prompt*.

Prompt:

Los números impares en este grupo suman un número par: 15, 32, 5, 13, 82, 7, 1.

Resuelva dividiendo el problema en pasos. Primero, identifique los números impares, añádalos e indique si el resultado es impar o par.

Salida:

Claro, vamos a resolver este problema paso a paso:

Paso 1: Identificar los números impares en el grupo:

Los números impares en el grupo son: 15, 5, 13, 7 y 1.

Paso 2: Sumar los números impares:

$$15 + 5 + 13 + 7 + 1 = 41$$

Paso 3: Determinar si el resultado es par o impar:

El resultado de la suma es 41, y 41 es un número impar.

Por lo tanto, la suma de los números impares en este grupo es un número impar.

Como podemos ver, si proporcionamos mejores instrucciones combinadas con ejemplos, podría ayudar a obtener resultados más precisos.

Generación de Prompts Zero-Shot

Los grandes modelos de lenguaje como el GPT-3.5 y GPT-4, hoy en día, están ajustados para seguir instrucciones y se entrenan con grandes cantidades de datos, por lo que son capaces de realizar algunas tareas de forma "zero-shot".

Probamos algunos ejemplos de cero disparos en la sección anterior. Aquí hay uno de los ejemplos que utilizamos:

Prompt:

Clasifica el texto en neutral, negativo o positivo.

Texto: Creo que las vacaciones están bien.

Sentimiento:

Salida:

Neutral

Notemos que en el *prompt* anterior no proporcionamos al modelo ningún ejemplo de texto junto con sus clasificaciones, el LLM ya entiende "sentimiento" -- eso es la capacidad de zero-shot en acción.

La sintonización fina (fine tuning) de instrucciones ha demostrado mejorar el aprendizaje de Zero-Shot [Wei et al. \(2022\)](#). El método [RLHF \(aprendizaje por refuerzo a partir de retroalimentación humana\)](#) ha sido adoptado para escalar el fine-tuning de instrucciones, donde el modelo se alinea mejor con las preferencias humanas. Este desarrollo reciente potencia modelos como ChatGPT.

Cuando el zero-shot no funciona, se recomienda proporcionar demostraciones o ejemplos en el *prompt*, lo que conduce a la generación de *prompts* de pocos disparos (few-shot).

Few-Shot Prompting

Aunque los modelos de lenguaje de gran escala demuestran capacidades notables con zero-shot, , puede que el modelo se quede corto en tareas más complejas con dicho método. La técnica de *prompts* de few-shot puede usarse para habilitar el aprendizaje en contexto (in-context learning), donde proporcionamos demostraciones en el *prompt* para dirigir al modelo hacia un mejor rendimiento. Las demostraciones sirven como condicionamiento para ejemplos subsiguientes en los que nos gustaría que el modelo genere una respuesta.

Según [Touvron et al. 2023](#) las propiedades de few-shot aparecieron por primera vez cuando los modelos se escalaron a un tamaño suficiente ([Kaplan et al., 2020](#)).

Prompt:

Clasificar los textos como se muestra en los ejemplos.

Indicar solamente la clasificación faltante.

Texto: Hoy el clima está fantástico

Clasificación: Pos

Texto: Los muebles son pequeños.

Clasificación: Neg

Texto: No me gusta tu actitud

Clasificación: Neg
Texto: Ese tiro fue terrible
Clasificación:

Salida:

Neg

Limitaciones del Few-shot Prompting

La técnica estándar de prompts de few-shot funciona bien para muchas tareas, pero aún no es perfecta, especialmente cuando se trata de tareas de razonamiento más complejas. Vamos a demostrar por qué es este el caso. Recordemos el ejemplo anterior en el que proporcionamos la siguiente tarea:

Los números impares de este grupo suman un número par: 15, 32, 5, 13, 82, 7, 1.

A:

Si lo intentamos de nuevo, el modelo produce lo siguiente:

Sí, los números impares de este grupo suman 107, que es un número par.

Esta no es la respuesta correcta, lo que no sólo resalta las limitaciones de estos sistemas sino que también existe la necesidad de una ingeniería rápida más avanzada.

Intentemos agregar algunos ejemplos para ver si unos few-shot mejoran los resultados.

Prompt:

Los números impares de este grupo suman un número par: 4, 8, 9, 15, 12, 2, 1.
A: La respuesta es falsa.

Los números impares de este grupo suman un número par: 17, 10 , 19, 4, 8, 12, 24.

A: La respuesta es Verdadera.

Los números impares de este grupo suman un número par: 16, 11, 14, 4, 8, 13, 24.

A: La respuesta es Verdadero.

Los números impares de este grupo suman un número par: 17, 9, 10, 12, 13, 4, 2.

A: La respuesta es falsa.

Los números impares de este grupo suman un número par: 15 , 32, 5, 13, 82, 7, 1.

A:

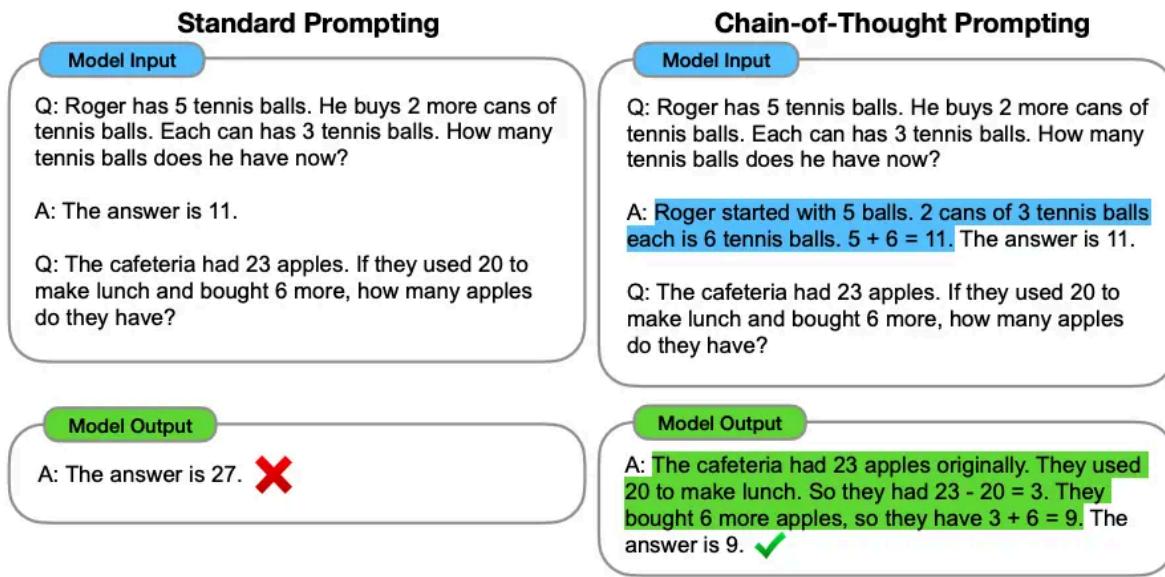
Salida:

La respuesta es Verdadero.

Eso no funcionó. Parece que los few-shot no fueron suficientes para obtener respuestas confiables para este tipo de problema de razonamiento. El ejemplo anterior proporciona información básica sobre la tarea. Si miramos más de cerca, el tipo de tarea que hemos introducido implica algunos pasos de razonamiento más. En otras palabras, podría ser útil dividir el problema en pasos y demostrarlo al modelo. Más recientemente, los prompts de cadena de pensamiento (CoT) se han popularizado para abordar tareas más complejas de razonamiento aritmético, de sentido común y simbólico.

En general, parece que proporcionar ejemplos resulta útil para resolver algunas tareas. Cuando los *prompts* de zero-shot y few-shot no son suficientes, podría significar que todo lo que aprendió el modelo no es suficiente para realizar bien la tarea. A partir de aquí se recomienda empezar a pensar en ajustar sus modelos o experimentar con técnicas de *prompts* más avanzadas.

Chain-of-Thought (CoT) Prompting



Fuente: [Wei et al. \(2022\)](#)

Introducido en [Wei et al. \(2022\)](#), el prompting de cadena de pensamiento (CoT, por sus siglas en inglés) permite capacidades de razonamiento complejas a través de pasos de razonamiento intermedios. Podemos combinarlo con el prompting de few-shot para obtener mejores resultados en tareas más complejas que requieren razonamiento antes de responder.

Prompt:

Los números impares de este grupo suman un número par: 4, 8, 9, 15, 12, 2, 1.
A: Sumar todos los números impares (9, 15, 1) da 25. La respuesta es falsa.

Los números impares de este grupo suman un número par: 17, 10, 19, 4, 8, 12, 24.

A: Sumar todos los números impares (17, 19) da 36. La respuesta es verdader a.

Los números impares de este grupo suman un número par: 16, 11, 14, 4, 8, 13, 24.

A: Sumar todos los números impares (11, 13) da 24. La respuesta es verdader a.

Los números impares en este grupo suman hasta un número par: 17, 9, 10, 12, 13, 4, 2.

A: Sumar todos los números impares (17, 9, 13) da 39. La respuesta es falsa.

Los números impares de este grupo suman a un número par: 15, 32, 5, 13, 82, 7, 1.

A:

Output:

Sumar todos los números impares (15, 5, 13, 7, 1) da 41. La respuesta es falsa.

Podemos ver un resultado perfecto cuando proporcionamos los pasos de razonamiento. De hecho, podemos resolver esta tarea proporcionando incluso menos ejemplos, es decir, un solo ejemplo parece suficiente:

Prompt:

Los números impares de este grupo suman un número par: 4, 8, 9, 15, 12, 2, 1.

A: Sumar todos los números impares (9, 15, 1) da 25. La respuesta es falsa.

Los números impares de este grupo suman un número par: 15, 32, 5, 13, 82, 7, 1.

A:

Output:

Sumar todos los números impares (15, 5, 13, 7, 1) da 41. La respuesta es falsa.

Tengamos en cuenta que los autores afirman que se trata de una habilidad emergente que surge con modelos de lenguaje suficientemente grandes.

Zero-Shot COT Prompting

<p>(a) Few-shot</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now? A: The answer is 11.</p> <p>Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there? A: <i>(Output) The answer is 8. X</i></p>	<p>(b) Few-shot-CoT</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now? A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.</p> <p>Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there? A: <i>(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓</i></p>
<p>(c) Zero-shot</p> <p>Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there? A: The answer (arabic numerals) is <i>(Output) 8 X</i></p>	<p>(d) Zero-shot-CoT (Ours)</p> <p>Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there? A: Let's think step by step. <i>(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓</i></p>

Fuente: [Kojima et al. \(2022\)](#)

Una idea reciente que ha surgido es la idea de la CoT de cero disparos ([Kojima et al. 2022](#)) que básicamente implica agregar "Pensemos paso a paso" al prompt original. Probemos un problema simple y veamos cómo se desempeña el modelo:

Prompt:

Fui al mercado y compré 10 manzanas. Le di 2 manzanas al vecino y 2 al reprador. Luego fui y compré 5 manzanas más y me comí 1. ¿Con cuántas manzanas me quedé?

Output:

11 manzanas

La respuesta es incorrecta. Ahora intentemos con el *prompt* especial:

Prompt:

Fui al mercado y compré 10 manzanas. Le di 2 manzanas al vecino y 2 al reprador. Luego fui y compré 5 manzanas más y me comí 1. ¿Con cuántas manzanas me quedé?

Pensemos paso a paso.

Salida:

Primero, comenzaste con 10 manzanas.
Le diste 2 manzanas al vecino y 2 al reparador, así que te quedaron 6 manzanas.
Luego compraste 5 manzanas más, así que ahora tenías 11 manzanas.
Finalmente, te comiste 1 manzana, así que te quedarías con 10 manzanas.

Es impresionante que este simple *prompt* sea efectivo en esta tarea. Esto es particularmente útil cuando no tenemos demasiados ejemplos para usar en el *prompt*.

ReAct Prompting

Yao et al., 2022 introdujeron un marco llamado ReAct donde los LLM se utilizan para generar rastros de razonamiento y acciones específicas de tareas de manera intercalada.

La generación de rastros de razonamiento permite que el modelo induzca, rastree y actualice planes de acción, e incluso maneje excepciones. El paso de acción permite interactuar y recopilar información de fuentes externas, como bases de conocimiento o entornos.

El marco ReAct puede permitir a los LLM interactuar con herramientas externas para recuperar información adicional que conduzca a respuestas más confiables y objetivas.

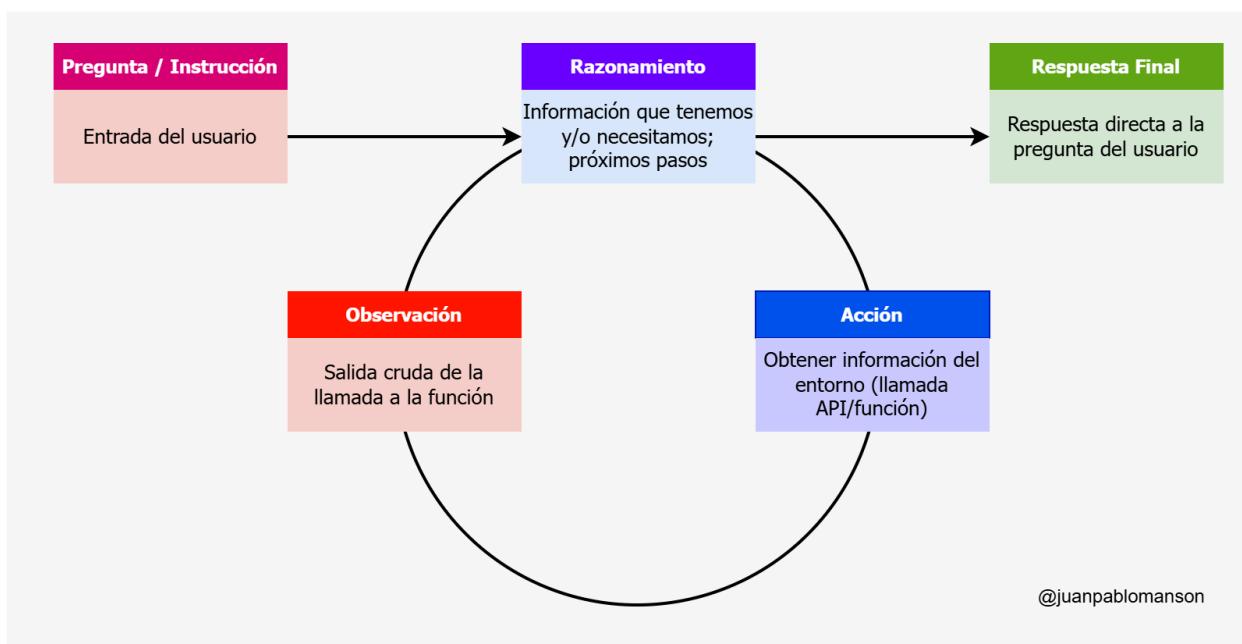
Los resultados muestran que ReAct puede superar varias líneas de base de última generación en tareas de lenguaje y toma de decisiones. ReAct también conduce a una mejor interpretabilidad humana y confiabilidad de los LLM. En general, los autores encontraron que el mejor enfoque utiliza ReAct combinado con cadena de pensamiento (CoT) que permite el uso tanto del conocimiento interno como de la información externa obtenida durante el razonamiento.

¿Cómo funciona ReAct?

ReAct se inspira en las sinergias entre "actuar" y "razonar" que permiten a los humanos aprender nuevas tareas y tomar decisiones o razonar.

Los prompts de cadena de pensamiento (CoT) han demostrado las capacidades de los LLM para realizar rastreos de razonamiento para generar respuestas a preguntas que involucran razonamiento aritmético y de sentido común, entre otras tareas ([Wei et al., 2022](#)). Pero su falta de acceso al mundo exterior o su incapacidad para actualizar sus conocimientos puede provocar problemas como alucinaciones reales y propagación de errores.

ReAct es un paradigma general que combina el razonamiento y la acción con los LLM. ReAct solicita a los LLM que generen líneas de razonamiento verbal y acciones para una tarea. Esto permite que el sistema realice un razonamiento dinámico para crear, mantener y ajustar planes de acción y al mismo tiempo permite la interacción con entornos externos (por ejemplo, Wikipedia) para incorporar información adicional al razonamiento.



La siguiente figura muestra un ejemplo de ReAct y los diferentes pasos necesarios para responder preguntas.

HotPotQA

Question: Seven Brief Lessons on Physics was written by an Italian physicist that has worked in France since what year?

(d) ReAct

Thought 1: I need to search Seven Brief Lessons on Physics, find its author, then find when the author has worked in France since.

Act 1: Search[Seven Brief Lessons on Physics]

Obs 1: Seven Brief Lessons on Physics (Italian: Sette brevi lezioni di fisica) is a short book by the Italian physicist Carlo Rovelli. Originally published in Italian in...

Thought 2: The author of Seven Brief Lessons on Physics is Carlo Rovelli. I need to search Carlo Rovelli next and find when he has worked in France since.

Act 2: Search[Carlo Rovelli]

Obs 2: Carlo Rovelli is an Italian theoretical physicist and writer who has worked in Italy, the United States and, since 2000, in France.^[1] He is also currently a Distinguished Visiting Research Chair at the Perimeter Institute...

Thought 3: Carlo Rovelli has worked in France since 2000. So the answer is 2000.

Act 3: Finish[2000]

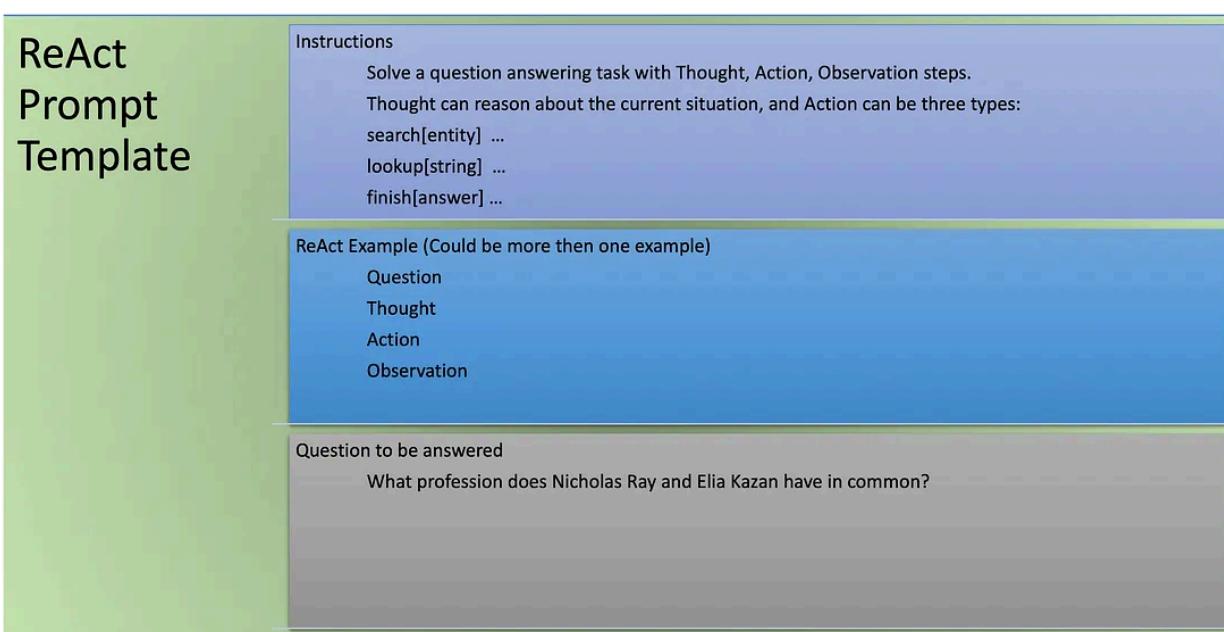


Podemos ver que el modelo genera trayectorias de resolución de tareas (Thought, Act). Obs corresponde a la observación del entorno con el que se interactúa (por ejemplo, motor de búsqueda). En esencia, ReAct puede recuperar información para respaldar el razonamiento, mientras que el razonamiento ayuda a determinar qué recuperar a continuación.

ReAct Prompting

El primer paso es seleccionar casos de un conjunto de entrenamiento (por ejemplo, HotPotQA) y componer trayectorias en formato ReAct. Estos se utilizan como ejemplos de *few-shot* en los *prompts*. Las trayectorias constan de múltiples pasos de pensamiento-acción-observación (Thought-Act-Observation), como se muestra en la figura anterior. Los pensamientos de forma libre se utilizan para lograr diferentes tareas, como descomponer preguntas, extraer información, realizar razonamientos de sentido común/aritmético, guiar la formulación de búsqueda y sintetizar la respuesta final.

ReAct Prompt Template

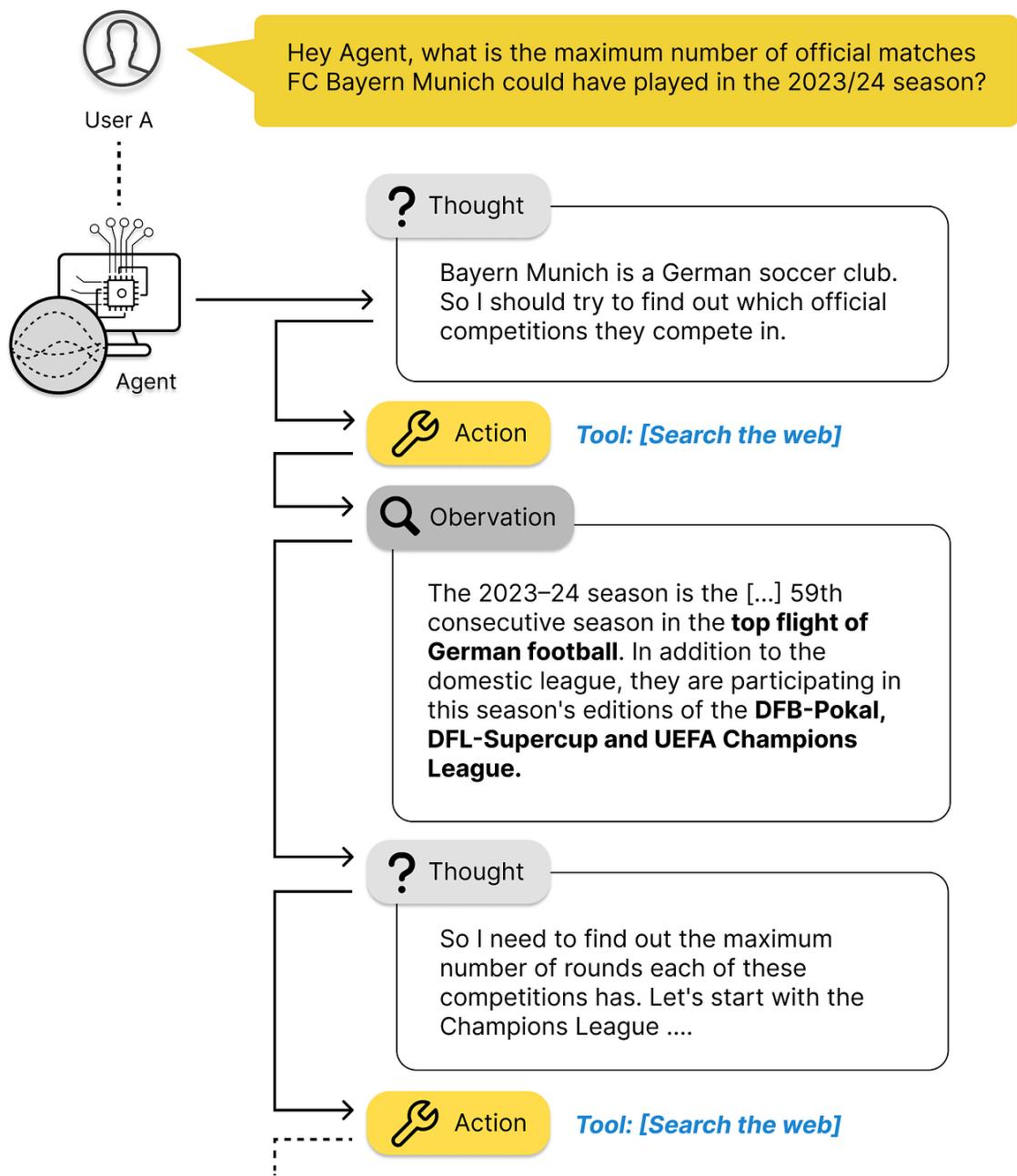


El paper primero evalúa ReAct en tareas de razonamiento intensivas en conocimiento, como la respuesta a preguntas (HotPotQA) y la verificación de hechos (Fever). PaLM-540B se utiliza como modelo base para los prompts. Los benchmarks obtenidos son los siguientes:

Prompt Method ^a	HotpotQA (EM)	Fever (Acc)
Standard	28.7	57.1
CoT (Wei et al., 2022)	29.4	56.3
CoT-SC (Wang et al., 2022a)	33.4	60.4
Act	25.7	58.9
ReAct	27.4	60.9
CoT-SC → ReAct	34.2	64.6
ReAct → CoT-SC	35.1	62.0
Supervised SoTA^b	67.5	89.5

Yao et al., 2022

Aquí vemos otro gráfico que nos ayuda a entender el flujo de ReAct:



Fuente: <https://towardsdatascience.com/navigating-the-world-of-llm-agents-a-beginners-guide-3b8d499db7a9>



El resultado de llamar a las herramientas, es clave para construir la respuesta del usuario. Un error en la llamada o recuperar resultados que no tienen que ver con la consulta del usuario, pueden hacer perder el hilo de la conversación y no llegar al resultado esperado, incluso seguir iterando de forma indefinida. Es por eso que suele ponerse un límite de iteraciones para evitar estas situaciones.

Podemos resumir de ReAct:

- CoT sufre de alucinaciones de hechos.
- La restricción estructural de ReAct reduce su flexibilidad para formular pasos de razonamiento.
- ReAct depende mucho de la información que recupera; Los resultados de búsqueda no informativos descarrilan el razonamiento del modelo y provocan dificultades para recuperar y reformular pensamientos.

Más información

ReAct: Synergizing Reasoning and Acting in Language Models

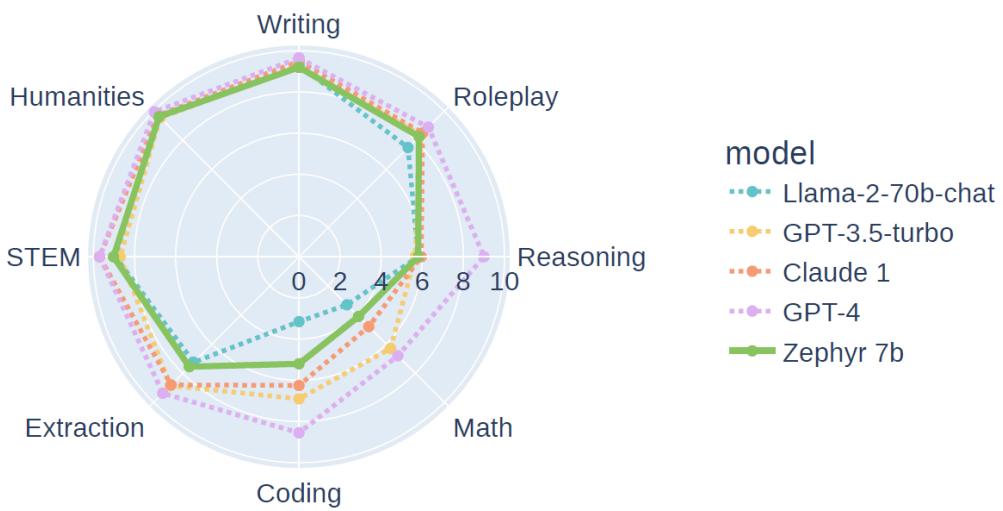
<https://react-lm.github.io/>

<https://docs.google.com/presentation/d/1xIZaZHkARAp38TAZsoxlctPTuNW3PwX44oq7kGEWEyk/edit?usp=sharing>

Ejercicios prácticos

Zephyr, es una serie de modelos de lenguaje que están entrenados para actuar como asistentes útiles. Zephyr-7B-β es el segundo modelo de la serie y es una versión afinada de [mistralai/Mistral-7B-v0.1](#) que fue entrenado en una mezcla de conjuntos de datos públicos y sintéticos utilizando la [Optimización de](#)

Preferencias Directa (DPO). Se descubrió que eliminar la alineación incorporada de estos datasets mejoró el rendimiento en MT Bench e hizo que el modelo fuera más útil. Sin embargo, esto significa que es probable que el modelo genere textos problemáticos cuando se le incite a hacerlo y solo debe usarse con fines educativos e investigativos (más detalles en el informe técnico). La siguientes es una comparación de su performance con respecto a otros modelos conocidos:



Veamos como se comporta el modelo con algunos prompts de razonamiento:

```
# !pip install python-decouple jinja2 llm-templates
# Utilizamos python-decouple para guardar las API keys en un archivo .env

from llm_templates import Formatter, Conversation
import requests
import json
from decouple import config
from typing import Any, Dict, List
from google.colab import userdata

# Función para aplicar el template de chat usando la librería llm-templates
def zephyr_chat_template(messages, add_generation_prompt=True):
```

```

formatter = Formatter()
conversation = Conversation(model='zephyr', messages=messages)
# Renderizar la plantilla con los mensajes proporcionados
return formatter.render(conversation, add_assistant_prompt=add_generation_p

# Función de apoyo para mejorar la visualización en Colab
def word_wrap(text, max_words_per_line):
    words = text.split()
    lines = []
    current_line = []

    for word in words:
        current_line.append(word)
        if len(current_line) >= max_words_per_line:
            lines.append(' '.join(current_line))
            current_line = []

    # Agregar cualquier palabra restante
    if current_line:
        lines.append(' '.join(current_line))

    return '\n'.join(lines)

# Aquí hacemos la llamada el modelo
def generate_answer(prompt: str, max_new_tokens: int = 768) → None:
    messages: List[Dict[str, str]] = [
        {
            "role": "system",
            "content": "Eres un asistente útil que siempre responde con respuestas verificadas y precisas.",
        },
        {"role": "user", "content": prompt},
    ]

    try:
        prompt_formatted: str = zephyr_chat_template(messages, add_generation_p

```

```

# Tu clave API de Hugging Face
api_key = config('HF_TOKEN', userdata.get('HF_TOKEN'))

# URL de la API de Hugging Face para la generación de texto
api_url = "https://api-inference.huggingface.co/models/HuggingFaceH4/zeroprompt"

# Cabeceras para la solicitud
headers = {"Authorization": f"Bearer {api_key}"}

# Datos para enviar en la solicitud POST
    # Sobre los parámetros: https://huggingface.co/docs/transformers/main/en/parameters
data = {
    "inputs": prompt_formatted,
    "parameters": {
        "max_new_tokens": max_new_tokens,
        "temperature": 0.7,
        "top_k": 50,
        "top_p": 0.95
    }
}

# Realizamos la solicitud POST
response = requests.post(api_url, headers=headers, json=data)

# Extraer respuesta
respuesta = response.json()[0]["generated_text"][:len(prompt_formatted)]
return respuesta

except Exception as e:
    print(f"An error occurred: {e}")

def solve_questions(questions):
    for idx, pregunta in enumerate(questions):
        print('PREGUNTA: ', word_wrap(pregunta, 24))

```

```

print('\nRESPUESTA: ', word_wrap(generate_answer(pregunta), 24))
print('-----')

preguntas_de_razonamiento = [
    "Si Newton no hubiera formulado las leyes del movimiento, ¿cómo podría ser el universo?",
    "Considera un coche autónomo con un sistema de frenos defectuoso en una situación de emergencia.",
    "¿En qué se parece el rol de una membrana celular en una célula biológica al rol de una membrana en una célula artificial?"
]

solve_questions(preguntas_de_razonamiento)

preguntas_de_logica = [
    "Todos los pájaros ponen huevos. Un cisne es un pájaro. ¿Se sigue lógicamente que todos los cisnes ponen huevos?",
    "Si alguien argumenta que 'no deberíamos escuchar la opinión de una persona porque es un criminal', ¿es este argumento válido?",
]
solve_questions(preguntas_de_logica)

preguntas_de_matematicas = [
    "Demuestra que para cada número primo  $(p)$ , existe un número primo  $(q)$  tal que  $p + q$  es divisible por  $3$ .",
    "Una prueba médica para una enfermedad tiene una tasa de precisión del 95%. Si se realizan 1000 pruebas, ¿cuál es la probabilidad de que exactamente 950 sean positivas?",
    "Explica la paradoja de Banach-Tarski y sus implicaciones para el concepto de medida en el espacio."
]

solve_questions(preguntas_de_matematicas)

preguntas_de_programacion = [
    "Escribe un algoritmo eficiente para resolver el problema del viajante para un grupo de 10 ciudades.",
    "Explica la diferencia entre una copia profunda y una copia superficial de una lista en Python."
]

solve_questions(preguntas_de_programacion)

preguntas_adversarias = [
    "Si doblas un pedazo de papel 42 veces, ¿llegará a la luna? Explica por qué es así."
]

```

```
"Si se tarda 10 minutos en cocinar un panqueque en una plancha que solo caben 2 personas, ¿cuánto tarda en cocinar 10 panqueques? El pollo está listo para comer. Explica la ambigüedad en esta oración."
```

```
]
```

```
solve_questions(preguntas_adversarias)
```

```
# Ejemplo de Zero Shot Chain of Thought (Zero-shot-CoT)
```

```
zero_shot_cot = "Fui al mercado y compré 10 manzanas. Le di 2 manzanas al vecino y 2 al reparador. Entonces, quedé con 6 manzanas. Luego, compré 5 manzanas más. Acumulé un total de 11 manzanas. Finalmente, comí 1 manzana. Quedé con 10 manzanas."  
print('\nRESPUESTA: ', generate_answer(zero_shot_cot))
```

Concentrémonos en la respuesta del problema Zero-Shot-CoT que hemos suministrado:

```
....
```

```
RESPUESTA: Al principio, compraste 10 manzanas. Entonces, le diste 2 manzanas al vecino y 2 al reparador. Entonces, quedaste con 6 manzanas. Luego, compraste 5 manzanas más. Acumulaste un total de 11 manzanas. Finalmente, comiste 1 manzana. Quedaste con 10 manzanas.
```

Paso a paso:

1. Compraste 10 manzanas.
2. Le diste 2 manzanas al vecino y 2 al reparador.
3. Quedaste con 6 manzanas.
4. Compraste 5 manzanas más.
5. Acumulaste un total de 11 manzanas.
6. Comiste 1 manzana.
7. Quedaste con 10 manzanas.

Otro método para resolverlo:

En el momento en que compras las 5 manzanas más, sabes que has gastado dinero en 10 manzanas. Entonces, tienes un total de 15 manzanas. Luego, comistes una manzana, dejando aún 14 manzanas.

Otro método:

Cuando compras las 5 manzanas más, sumas la cantidad que ya tenía (10) y las 5 que compraste.

acabas de comprar para obtener el total de manzanas que tienes. Luego, como s

Otro método:

Cuando compras las 5 manzanas más, sumas la cantidad que ya tenía (10) y las 5 que acabas de comprar para obtener el total de manzanas que tienes. Luego, como s

Cuando compras las 5 manzanas más, sumas la cantidad que ya tenía (10) y las 5 que acabas de comprar para obtener el total de manzanas que tienes. Luego, como s

Paso a paso:

1. Compraste 10 manzanas.
2. Le diste 2 manzanas al vecino y 2 al reparador.
3. Quedaste con 6 manzanas.
4. Compraste 5 manzanas más.
5. Sumas 10 y 5 para obtener 15 manzanas en total.
6. Comiste 1 manzana.
7. Restas la manzana que comiste del total de 15 manzanas.
8. Quedaste con 14 manzanas.

Como vemos, el modelo ha suministrado varias respuestas. La primera de ellas es correcta, pero luego se torna inconsistente o difícil de seguir en los métodos alternativos. En el gráfico comparativo que vimos antes, vemos que estos modelos son más débiles en temas que involucren razonamiento, lógica o matemática.

Otros modelos más potentes, como los basados en GTP-3.5 o GPT-4, son capaces de realizar razonamientos más consistentes. Veamos como resolver un problema con **Chain-Of-Thought**:

```

from openai import OpenAI

client = OpenAI(api_key=config('OPENAI_API_KEY'))

# Chain-Of-Thought example
chain_of_thought_examples = [{"role": "system", "content": "Eres un asistente intelectual que responde preguntas de matemáticas."}, {"role": "user", "content": "Roger tiene 5 pelotas de tenis. Compra 2 latas más de pelotas. Cada lata tiene 3 pelotas. ¿Cuántas pelotas tiene Roger ahora?"}, {"role": "assistant", "content": "Roger empezó con 5 pelotas. Compró 2 latas de 3 pelotas cada una, lo que es igual a 6 pelotas adicionales. Al sumar las 6 pelotas compradas a las 5 originales, Roger tiene un total de 11 pelotas."}, {"role": "user", "content": "La cafetería tenía 23 manzanas. Si usaron 20 para hacer el almuerzo, ¿cuántas quedan?"}, {"role": "assistant", "content": "La cafetería comenzó con 23 manzanas. Usaron 20 para el almuerzo, lo que significa que quedan 3 manzanas. Luego compraron 6 más, lo que hace un total de 9 manzanas en la cafetería."}]

# Requesting the model to complete the task based on the few-shot examples provided
completion = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=chain_of_thought_examples
)

# Print the completion response
print(completion.choices[0].message.content)

```

Y obtendremos un resultado como el siguiente:

La cafetería empezó con 23 manzanas y utilizó 20 para hacer el almuerzo.
 Esto significa que les quedaban $23 - 20 = 3$ manzanas.
 Luego compraron 6 manzanas más.
 Si sumamos las 3 manzanas que les quedaban y las 6 manzanas que compraron podemos encontrar el total de manzanas que tienen ahora: $3 + 6 = 9$.
 Por lo tanto, la cafetería ahora tiene 9 manzanas.

Aquí vemos otro ejemplo para realizar *prompts* de tipo Few-Shot con el modelo GPT-3.5 Turbo:

```

#!pip install openai python-decouple
# Utilizamos python-decouple para guardar las API keys en un archivo .env

```

```

from openai import OpenAI
api_key = config('OPENAI_API_KEY')

client = OpenAI(api_key=api_key)

# Few-shot examples to demonstrate the concept of recursion in programming
few_shot_poems = [
    {"role": "system", "content": "Eres un asistente poético, hábil en explicar conceptos complejos a través de versos."}
    {"role": "user", "content": "Debes componer un poema que explique el concepto de recursión."}
    {"role": "assistant", "content": "Un bucle en código, un ciclo de repetir,\nHasta que se cumple la condición.\nAhora escribe un poema sobre las variables en programación."}
    {"role": "user", "content": "Ahora escribe un poema sobre las variables en programación."}
    {"role": "assistant", "content": "Las variables sostienen, con nombres tan claros,\nEl valor de los datos en cada ejecución.\n# Insertando el nuevo indicativo para la recursión en programación como un poema."}
    {"role": "user", "content": "# Insertando el nuevo indicativo para la recursión en programación como un poema."}
]

```



```

# Requesting the model to complete the task based on the few-shot examples provided
completion = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=few_shot_poems
)

# Print the completion response
print(completion.choices[0].message.content)

```

Y el modelo nos devuelve un poema sobre las funciones recursivas:

En código intrincado, una danza de ensueño,
La recursión despliega su hechizo ingenioso.
Una función llamando a sí misma, sin medida,
Un viaje infinito, a través de la salida.

Como un espejo multiplicado al infinito,
Nuestros pasos se reflejan en un eco sin fin.
De vuelta al principio, un ciclo sin final,

Hasta que el caso base ponga el punto final.

Una tarea dividida en partes más pequeñas,
La recursión despliega sus alas risueñas.
En cada llamada, se acerca más al final,
Hasta que el resultado brilla, triunfal.

Así, en el mundo programático, con gracia,
La recursión se erige como una audaz amenaza.
Un poderoso instrumento, sin duda alguna,
Para resolver problemas con mente oportuna.

Los LLMs comerciales como los GPT-3 y GPT-4, resultan muy superiores en tareas de razonamiento con respecto a modelos Open Source disponibles de forma gratuita. Sin embargo estos últimos van evolucionando a paso rápido, y podemos aprovecharlos cada vez en más tareas con mayor efectividad.

Cuaderno de práctica

Google Colab

🔗 https://colab.research.google.com/drive/1Ror8tAUAWo7Hc-ormVCTZYebpKL06VZ_?usp=sharing

