



Prompt Engineering

= Autor/Fuente	https://drive.google.com/file/d/1AbaBYbEa_EbPelsT40-vj64L-2lwUJHy/view
≡ Categoría	
* Estatus	Terminado
📅 Finalizado el:	@5 de julio de 2025 13:58
🕒 Modificado	@18 de noviembre de 2025 14:29
🕒 Creado	@17 de junio de 2025 8:13
🕒 Tipo	Libro

Prompt Engineering

Author: Lee Boonstra

▼ Entendiendo la ingeniería de indicaciones (Prompt Engineering)

Introducción: Hay aspectos que afectan la eficacia del Prompt (modelo, datos de entrenamiento, configuraciones del modelo, elección de palabras, estilo, tono, estructura, contexto, etc, son importantes)

Por lo tanto la ingeniería de indicaciones es un proceso iterativo.

Darle indicaciones inadecuadas pueden dar lugar a respuestas inexactas y entorpecer la capacidad del modelo para proporcionar una salida significativa.

Prompt Engineering (Ingeniería de indicaciones)

Empecemos sabiendo que un LLM es un motor de **predicción**.

El modelo toma texto secuencial como entrada y luego predice cual debería ser el siguiente **token**, basándose en los datos con los que se entreno. La predicción del siguiente token se basa en la relación entre lo que hay de tokens anteriores y lo que el LLM ha visto durante su entrenamiento.

Al momento de escribir un **prompt** (indicación) intentamos configurar el LLM para predecir la secuencia correcta de tokens. La **ingeniería de indicaciones** es el proceso de diseñar indicaciones de alta calidad que guían a los LLM para producir resultados precisos.

En el contexto del **procesamiento del lenguaje natural y los LLM**, una indicación es una entrada que se proporciona al modelo para generar una respuesta o predicción.

Estas indicaciones se pueden utilizar para lograr varios tipos de tareas de comprensión y generación, como resumen de texto, extracción de información, preguntas y respuestas, clasificación de texto, traducción de lenguaje o código, generación de código y documentación o razonamiento de código.

Al realizar la ingeniería de indicaciones, comenzará eligiendo un modelo. Es posible que las indicaciones deban optimizarse para su modelo específico, independientemente de si utiliza modelos de lenguaje de Gemini en Vertex AI, GPT, Claude o un modelo de código abierto como Gemma o LLAMA.

Además de la indicación, también necesitará modificar las diversas configuraciones de un LLM.

Configuración de salida del LLM

Una vez que elija su modelo, deberá determinar la configuración del modelo. La mayoría de los LLM vienen con varias opciones de configuración que controlan la salida del LLM. Una ingeniería de indicaciones efectiva requiere establecer estas configuraciones de manera óptima para su tarea.

Longitud de salida

Un ajuste de configuración importante es la cantidad de tokens que se generarán en una respuesta. Generar más tokens requiere más cálculos del LLM, lo que genera un mayor consumo de energía, tiempos de respuesta potencialmente más lentos y mayores costos.

Reducir la longitud de salida del LLM no hace que el LLM se vuelva más estilística o textualmente conciso en la salida que crea, solo hace que el LLM deje de predecir más tokens una vez que se alcanza el límite. Si sus necesidades requieren una longitud de salida corta, posiblemente también necesite diseñar su solicitud para adaptarla.

Tener en cuenta que generar más tokens requiere más cálculos del LLM, lo que genera un mayor consumo de energía y tiempos de respuesta potencialmente más lentos, lo que a su vez genera mayores costos.

Controles de muestreo

Los LLM predicen probabilidades de cuál podría ser el siguiente token no un solo token, y cada token en el vocabulario del LLM obtiene una probabilidad. Esas probabilidades de token se muestran para determinar cuál será el siguiente token producido. **Temperatura**, top-K y top-P son los ajustes de configuración más comunes que determinan cómo se procesan las probabilidades de token predichas para elegir un solo token de salida.

Temperatura

La temperatura controla el grado de aleatoriedad en la selección de tokens. Las **temperaturas más bajas** son buenas para las indicaciones que esperan una respuesta más determinista, mientras que las **temperaturas más altas**

pueden conducir a resultados más diversos o inesperados. Una temperatura de 0 (decodificación codiciosa) es determinista: siempre se selecciona el token con la probabilidad más alta (aunque tenga en cuenta que si dos tokens tienen la misma probabilidad predicha más alta, dependiendo de cómo se implemente el desempate, es posible que no siempre obtenga el mismo resultado con una temperatura de 0).

Las temperaturas cercanas al máximo tienden a crear un resultado más aleatorio. Y a medida que la temperatura aumenta cada vez más, todos los tokens tienen la misma probabilidad de ser el siguiente token predicho.

El control de temperatura de Gemini se puede entender de manera similar a la función **softmax** utilizada en el aprendizaje automático. Un ajuste de temperatura bajo refleja una temperatura softmax baja (T), enfatizando una única temperatura preferida con alta certeza. Un ajuste de temperatura de Gemini más alto es como una temperatura softmax alta, lo que hace que un rango más amplio de temperaturas alrededor del ajuste seleccionado sea más aceptable. Esta mayor incertidumbre se adapta a escenarios en los que una temperatura rígida y precisa puede no ser esencial, como por ejemplo al experimentar con resultados creativos.

Top-K y top-P

Top-K y top-P (también conocido como muestreo de núcleo) son dos configuraciones de muestreo utilizadas en los LLM para restringir que el siguiente token predicho provenga de tokens con las mayores probabilidades predichas. Al igual que la temperatura, estas configuraciones de muestreo controlan la aleatoriedad y la diversidad del texto generado.

- El **muestreo Top-K** selecciona los K tokens más probables de la distribución predicha del modelo. Cuanto mayor sea el top-K, más creativo y variado será el resultado del modelo; cuanto menor sea el top-K, más inquieto y factual (mas perteneciente a los hechos) será el resultado del modelo. Un top-K de 1 equivale a una decodificación voraz

- El **muestreo Top-P** selecciona los tokens superiores cuya probabilidad acumulada no excede un valor determinado (P). Los valores de P van de 0 (decodificación codiciosa) a 1 (todos los tokens del vocabulario del LLM).

La mejor manera de elegir entre top-K y top-P es experimentar con ambos métodos (o ambos juntos) y ver cuál produce los resultados que busca.

Juntándolo todo

La elección entre top-K, top-P, la temperatura y la cantidad de tokens a generar depende de la aplicación específica y del resultado deseado, y las configuraciones se influyen entre sí. También es importante asegurarse de comprender cómo el modelo elegido combina las diferentes configuraciones de muestreo.

Si la temperatura, top-K y top-P están disponibles (como en Vertex Studio), los tokens que cumplen con los criterios top-K y top-P son candidatos para el siguiente token predicho, y luego se aplica la temperatura a la muestra de los tokens que pasaron los criterios top-K y top-P. Si solo top-K o top-P está disponible, el comportamiento es el mismo, pero solo se utiliza la configuración top-K o P.

Si la temperatura no está disponible, los tokens que cumplan con los criterios top-K y/o top-P se seleccionan aleatoriamente para producir un único token predicho.

En configuraciones extremas de un valor de configuración de muestreo, esa configuración de muestreo cancela otras configuraciones o se vuelve irrelevante.

Tener en cuenta:

- Si establece la **temperatura en 0**, top-K y top-P se vuelven irrelevantes: el token más probable se convierte en el siguiente token predicho. Si establece la temperatura extremadamente alta (por encima de 1, generalmente en los 10), la temperatura se vuelve irrelevante y cualquier token que pase los criterios top-K o top-P se muestrea aleatoriamente para elegir el siguiente token predicho.
- Si establece top-K en 1, la temperatura y top-P se vuelven irrelevantes. Solo un token pasa los criterios top-K, y ese token es el siguiente token predicho. Si establece top-K extremadamente alto, como el tamaño del vocabulario del LLM, cualquier token con una probabilidad distinta de cero de ser el siguiente token cumplirá los criterios top-K y no se seleccionará ninguno
- Si establece top-P en 0 (o un valor muy pequeño), la mayoría de las implementaciones de muestreo LLM solo considerarán el token con mayor probabilidad de cumplir con los criterios de top-P, lo que hace que la temperatura y top-K sean irrelevantes. Si establece top-P en 1, cualquier token con una probabilidad distinta de cero de ser el siguiente token cumplirá con los criterios de top-P y no se seleccionará ninguno.

Como punto de partida general, una temperatura de 0,2, top-P de 0,95 y top-K de 30 le darán resultados relativamente coherentes que pueden ser creativos, pero no excesivamente. Si desea resultados especialmente creativos, intente comenzar con una temperatura de 0,9, top-P de 0,99 y top-K de 40. Y si desea resultados menos creativos, intente comenzar con una temperatura de 0,1, top-P de 0,9 y top-K de 20. Finalmente, si su tarea siempre tiene una única respuesta correcta (por ejemplo, responder a un problema de matemáticas), comience con una temperatura de 0.

NOTA: Con más libertad (mayor temperatura, top-K, top-P y tokens de salida), el LLM podría generar texto menos relevante.

ADVERTENCIA: ¿Alguna vez has visto una respuesta que termine con una gran cantidad de palabras de relleno? Esto también se conoce como el "**error del bucle de repetición**", que es un problema común en los Modelos de Lenguaje Grandes donde el modelo se atasca en un ciclo, generando

repetidamente la misma palabra (de relleno), frase o estructura de oración, a menudo exacerbado por una temperatura y top-k/inapropiados configuraciones top-p.

Esto puede ocurrir tanto con configuraciones de temperatura bajas como altas, aunque por diferentes razones. A bajas temperaturas, el modelo se vuelve excesivamente determinista, ciñéndose rígidamente a la ruta de mayor probabilidad, lo que puede conducir a un bucle si esa ruta vuelve a visitar texto generado previamente. Por el contrario, a altas temperaturas, la salida del modelo se vuelve excesivamente aleatoria, lo que aumenta la probabilidad de que una palabra o frase elegida al azar, por casualidad, vuelva a un estado anterior, creando un bucle debido a la gran cantidad de opciones disponibles. En ambos casos, el proceso de muestreo del modelo se "atasca", lo que resulta en una salida monótona e inútil hasta que se llena la ventana de salida. Resolver esto a menudo requiere un ajuste cuidadoso de la temperatura y los valores top-k/top-p para encontrar el equilibrio óptimo entre el determinismo y la aleatoriedad.

Técnicas de indicación

Los LLM están optimizados para seguir instrucciones y se entran con grandes cantidades de datos para que puedan comprender una indicación y generar una respuesta. Pero los LLM no son perfectos; cuanto más claro sea el texto de la indicación, mejor será para el LLM predecir el siguiente texto probable. Además, las técnicas específicas que aprovechan cómo se entran los LLM y cómo funcionan te ayudarán a obtener los resultados relevantes de los LLM.

Ahora que entendemos qué es la ingeniería de indicaciones y qué implica, profundicemos en algunos ejemplos de las técnicas de indicación más importantes.

▼ Ejemplos de técnicas de indicación más importantes

Ahora que entendemos qué es la ingeniería de indicaciones y qué implica, profundicemos en algunos ejemplos de las técnicas de indicación más importantes.

Indicación general / cero disparo

Una indicación de cero disparo es el tipo de indicación más simple. **Solo proporciona una descripción de una tarea y algo de texto para que el LLM comience.** Esta entrada puede ser cualquier cosa: una pregunta, el comienzo de una historia o instrucciones. El nombre cero disparo significa 'sin ejemplos'

Usemos Vertex AI Studio (para lenguaje) en Vertex AI, que proporciona un entorno de pruebas para las indicaciones. En la Tabla 1, verá un ejemplo de indicación de cero disparos para clasificar reseñas de películas.

El formato de tabla que se utiliza a continuación es una excelente manera de documentar las indicaciones. Es probable que sus indicaciones pasen por muchas iteraciones antes de terminar en una base de código, por lo que es importante realizar un seguimiento de su trabajo de ingeniería de indicaciones de una manera disciplinada y estructurada. Encontrará más información sobre este formato de tabla, la importancia de realizar un seguimiento del trabajo de ingeniería de indicaciones y el proceso de desarrollo de indicaciones en la sección de Mejores prácticas más adelante en este capítulo ("Documentar los distintos intentos de indicaciones").

La temperatura del modelo debe establecerse en un número bajo, ya que no se necesita creatividad, y utilizamos los valores predeterminados top-K y top-P de gemini-pro, que deshabilitan efectivamente ambas configuraciones (consulte 'Configuración de salida de LLM' más arriba). Preste atención a la salida generada. Las palabras perturbador y obra maestra deberían hacer que la predicción sea un poco más complicada, ya que ambas palabras se usan en la misma oración.

Nombre	t_1_clasificación_de_peliculas		
Objetivo	Clasificar las reseñas de películas como positivas, neutrales o negativas.		
Modelo	gemini-pro		
Temperatura	0.1	Límite de tokens	5
Top-K	NO	Top-P	1
Indicación	Clasificar las reseñas de películas como POSITIVAS, NEUTRALES o NEGATIVAS. Reseña: "Her" es un estudio perturbador que revela la dirección que tomará la humanidad si se permite que la IA siga evolucionando sin control. Ojalá hubiera más películas como esta obra maestra. Sentimiento:		
Resultado	POSITIVO		

Tabla 1. Un ejemplo de indicación de cero disparos

Cuando la indicación de cero disparos no funciona, puedes proporcionar demostraciones o ejemplos en la indicación, lo que lleva a indicaciones de "un disparo" y "pocos disparos". Indicación general/cero disparos

Un disparo y pocos disparos

Al crear indicaciones para modelos de IA, es útil proporcionar ejemplos. Estos ejemplos pueden ayudar al modelo a comprender lo que se le pide. Los ejemplos son especialmente útiles cuando se desea dirigir el modelo hacia una determinada estructura o patrón de salida.

Una indicación de una sola aplicación proporciona un solo ejemplo, de ahí el nombre de una sola aplicación. La idea es que el modelo tenga un ejemplo que pueda imitar para completar mejor la tarea.

Una indicación de pocas aplicaciones proporciona múltiples ejemplos al modelo. Este enfoque le muestra al modelo un patrón que debe seguir. La idea es similar a la de una sola aplicación, pero múltiples ejemplos del patrón deseado aumentan la probabilidad de que el modelo siga el patrón.

La cantidad de ejemplos que necesita para la solicitud de pocos intentos depende de algunos factores, incluida la complejidad de la tarea, la calidad de los ejemplos y las capacidades del modelo de IA generativa (gen IA) que

esté utilizando. **Como regla general, debe usar al menos de tres a cinco ejemplos para la solicitud de pocos intentos. Sin embargo, es posible que necesite usar más ejemplos para tareas más complejas, o puede que necesite usar menos debido a la limitación de longitud de entrada de su modelo.**

La Tabla 2 muestra un ejemplo de solicitud de pocos intentos. Usemos la misma configuración del modelo gemini-pro que antes, excepto que aumentamos el límite de tokens para adaptarnos a la necesidad de una respuesta más larga.

Objetivo	Analizar pedidos de pizza a JSON		
Modelo	gemini-pro		
Temperatura	0.1	Límite de tokens	250
Top-K	NO	Top-P	1
Solicitud	<p>Analizar el pedido de pizza de un cliente en JSON válido:</p> <p>EJEMPLO:</p> <p>Quiero una pizza pequeña con queso, salsa de tomate y pepperoni.</p> <p>Respuesta JSON:</p> <p>***</p> <pre>{ "size": "small", "tipo": "normal", "ingredientes": ["queso", "salsa de tomate", "pepperoni"]} ***</pre>		
Continúa en la página siguiente...			

Prompt	<p>EXAMPLE:</p> <pre>Can I get a large pizza with tomato sauce, basil and mozzarella { "size": "large", "type": "normal", "ingredients": [["tomato sauce", "basil", "mozzarella"]] } Now, I would like a large pizza, with the first half cheese and mozzarella. And the other tomato sauce, ham and pineapple.</pre> <p>JSON Response:</p>
Output	<pre>... { "size": "large", "type": "half-half", "ingredients": [["cheese", "mozzarella"], ["tomato sauce", "ham", "pineapple"]] } ...</pre>

Tabla 2. Un ejemplo de indicación de pocas oportunidades

Cuando elija ejemplos para su indicación, utilice ejemplos que sean relevantes para la tarea que desea realizar. Los ejemplos deben ser diversos, de alta calidad y estar bien escritos. Un pequeño error puede confundir al modelo y resultar en un resultado no deseado

Si intenta generar una salida que sea robusta a una variedad de entradas, entonces es importante incluir casos extremos en sus ejemplos. Los casos extremos son entradas que son inusuales o inesperadas, pero que el modelo aún debería poder manejar

Incitación sistemática, contextual y de roles

La incitación sistemática, contextual y de roles son técnicas que se utilizan para guiar la forma en que los LLM generan texto, pero se centran en diferentes aspectos:

- La **incitación sistemática** establece el contexto general y el propósito del modelo lingüístico. Define el panorama general de lo que el modelo debería estar haciendo, como traducir un idioma, clasificar una reseña, etc.

- La **incitación contextual** proporciona detalles específicos o información de fondo relevante para la conversación o tarea actual. Ayuda al modelo a comprender los matices de lo que se le pregunta y a adaptar la respuesta en consecuencia.
- La **incitación de roles** asigna un carácter o identidad específica para que el modelo lingüístico la adopte. Esto ayuda al modelo a generar respuestas que sean coherentes con el rol asignado y su conocimiento y comportamiento asociados.

Puede haber una superposición considerable entre la incitación sistémica, contextual y de roles. Por ejemplo, una incitación que asigna un rol al sistema también puede tener un contexto.

Sin embargo, cada tipo de incitación tiene un propósito principal ligeramente diferente:

- **Incitación del sistema:** Define las capacidades fundamentales y el propósito general del modelo
- **Indicación contextual:** proporciona información inmediata y específica de la tarea para guiar la respuesta. Es muy específica para la tarea o entrada actual, que es dinámica.
- **Indicación de rol:** enmarca el estilo y la voz de salida del modelo. Añade una capa de especificidad y personalidad.

Distinguir entre indicaciones de sistema, contextuales y de rol proporciona un marco para diseñar indicaciones con una intención clara, lo que permite combinaciones flexibles y facilita el análisis de cómo cada tipo de indicación influye en el resultado del modelo de lenguaje.

Profundicemos en estos tres tipos diferentes de indicaciones.

Indicaciones del sistema

La Tabla 3 contiene una indicación del sistema, donde específico información adicional sobre cómo devolver el resultado. Aumenté la temperatura para obtener un mayor nivel de creatividad y especifiqué un

límite de tokens más alto. Sin embargo, debido a mis instrucciones claras sobre cómo devolver el resultado, el modelo no devolvió texto adicional.

Objetivo	Clasificar las reseñas de películas como positivas, neutrales o negativas.		
Modelo	gemini-pro		
Temperatura 1		Límite de tokens	5
Top-K	40	Top-P	0.8
Indicación	<p>Clasificar las reseñas de películas como positivas, neutrales o negativas. Devolver solo la etiqueta en mayúsculas.</p> <p>Reseña: "Her" es un estudio perturbador que revela la dirección que tomará la humanidad si se permite que la IA siga evolucionando sin control. Es tan perturbador que no pude verlo.</p> <p>Sentimiento:</p>		
Salida	NEGATIVO		

Tabla 3. Un ejemplo de indicaciones del sistema

Las indicaciones del sistema pueden ser útiles para generar resultados que cumplan requisitos específicos. El nombre «**indicador del sistema**» en realidad significa «**proporcionar una tarea adicional al sistema**».

Por ejemplo, podría usar un indicador del sistema para generar un fragmento de código que sea compatible con un lenguaje de programación específico, o podría usar un indicador del sistema para devolver una estructura determinada. **Consulte la Tabla 4**, donde devuelvo la salida en formato JSON.

Goal	Classify movie reviews as positive, neutral or negative, return JSON.		
Model	gemini-pro		
Temperature	1	Token Limit	1024
Top-K	40	Top-P	0.8
Prompt	Classify movie reviews as positive, neutral or negative. Return valid JSON: Review: "Her" is a disturbing study revealing the direction humanity is headed if AI is allowed to keep evolving, unchecked. It's so disturbing I couldn't watch it. Schema: ... MOVIE: { "sentiment": String "POSITIVE" "NEGATIVE" "NEUTRAL", "name": String } MOVIE REVIEWS: { "movie_reviews": [MOVIE] } ... JSON Response: ...		
Output	...		
	{ "movie_reviews": [{ "sentiment": "NEGATIVE", "name": "Her" }] }		

Table 4. An example of system prompting with JSON format

Hay algunos beneficios en devolver objetos JSON desde una solicitud que extrae datos. En una aplicación del mundo real, no necesito crear manualmente este formato JSON, ya puedo devolver los datos en un orden ordenado (muy útil cuando se trabaja con objetos de fecha y hora), pero lo más importante es que al solicitar un formato JSON, se obliga al modelo a crear una estructura y limitar las alucinaciones.

Las solicitudes del sistema también pueden ser muy útiles para la seguridad y la toxicidad. Para controlar la salida, simplemente agregue una línea adicional a su solicitud como: '**Debe ser respetuoso en su respuesta...**'

Solicitud de roles

La solicitud de roles es una **técnica en la ingeniería de solicitudes** que implica asignar un rol específico al modelo general de IA. Esto puede ayudar al modelo a generar una salida más relevante e informativa, ya que el modelo puede adaptar sus respuestas al rol específico que se le ha asignado.

Por ejemplo, podrías pedirle a un modelo de IA general que sea editor de libros, maestro de jardín de infantes o orador motivacional. Una vez que se le haya asignado un rol al modelo, puedes darle indicaciones específicas para ese rol. **Por ejemplo**, podrías pedirle a un maestro que cree un plan de lección que luego pueda revisar. Observa la **Tabla 5**, donde el modelo asume el rol de guía de viajes.

Goal	Act as travel guide and provide 3 travel suggestions		
Model	gemini-pro		
Temperature	1	Token Limit	1024
Top-K	40	Top-P	0.8
Prompt	<p>I want you to act as a travel guide. I will write to you about my location and you will suggest 3 places to visit near me. In some cases, I will also give you the type of places I will visit.</p> <p>My suggestion: "I am in Amsterdam and I want to visit only museums."</p> <p>Travel Suggestions:</p>		
Output	<ol style="list-style-type: none">1. Rijksmuseum: World-renowned museum, known for Dutch masterpieces like Rembrandt's "The Night Watch." Don't miss the incredible collection of paintings and artifacts showcasing Dutch art and history.2. Van Gogh Museum: Immerse yourself in the captivating works of Vincent van Gogh. The museum houses the largest collection of his paintings and drawings, including "The Starry Night" and "Sunflowers."3. Stedelijk Museum Amsterdam: Discover modern and contemporary art from around the world. The museum's collection includes works by Picasso, Kandinsky, and Rietveld, housed in a striking modern building.		

Table 5. An example of role prompting

El ejemplo anterior muestra un ejemplo de asumir el rol de un agente de viajes. Cuando cambiaras el rol a profesor de geografía, notarías que recibirás una respuesta diferente.

Definir una perspectiva de rol para un modelo de IA le proporciona un plano del tono, el estilo y la experiencia enfocada que buscas para mejorar la calidad, la relevancia y la eficacia de tu trabajo.

Aquí tienes algunos estilos que puedes elegir y que considero efectivos:

Confrontacional, Descriptivo, Directo, Formal, Humorístico, Influyente, Informal, Inspirador, Persuasivo

Cambiemos nuestra consigna en la Tabla 6 para incluir un estilo humorístico e inspirador.

Indicación	Quiero que actúes como un guía de viajes. Te escribiré sobre mi ubicación y tú me sugerirás 3 lugares para visitar cerca de mí en un estilo humorístico. Mi sugerencia: "Estoy en Manhattan". Sugerencias de viaje:
Resultado	1. Contempla el Empire State of Mind: Asciende a las vertiginosas alturas del Empire State Building y disfruta de la gloria del horizonte de Manhattan. Prepárate para sentirte como King Kong en la cima de la Gran Manzana, sin el plátano gigante del tamaño de un simio 2. Ponte artístico en el MoMA: Da rienda suelta a tu aficionado al arte interior en el Museo de Arte Moderno (MoMA). Contempla obras maestras que te dejarán atónito y te harán cuestionar si tus dibujos de monigotes tienen algún mérito artístico. 3. Compra hasta cansarte en la Quinta Avenida: Disfruta de una terapia de compras en la icónica Quinta Avenida. Prepárate para una sorpresa desagradable mientras miras escaparates (o compras de verdad) en boutiques de diseñadores que te harán llorar. Pero bueno, estás en Manhattan, ¡así que mejor aprovecha lo fabuloso!

Tabla 6. Un ejemplo de indicaciones de rol con un tono y estilo humorísticos

Tabla 6. Un ejemplo de indicaciones de rol con un tono y estilo humorísticos

Indicaciones contextuales

Al proporcionar indicaciones contextuales, puedes ayudar a garantizar que tus interacciones con IA sean lo más fluidas y eficientes posible. El modelo podrá comprender más rápidamente tu solicitud y generar respuestas más precisas y relevantes, como puedes ver en el ejemplo de la **Tabla 7**

Objetivo	Sugerir artículos para un blog sobre juegos retro		
Modelo	gemini-pro		
Temperatura	1	Límite de fichas	1024
Top-K	40	Top-P	0.8
Indicación	<p>Contexto: Estás escribiendo para un blog sobre videojuegos arcade retro de los 80.</p> <p>Sugiere 3 temas sobre los que escribir un artículo con unas pocas líneas de descripción de lo que debería contener.</p>		
Resultado	<ol style="list-style-type: none"> 1. **La evolución del diseño de las máquinas arcade:** Este artículo exploraría la evolución de los diseños de las máquinas arcade, desde las primeras máquinas de madera y metal de la década de 1970 hasta los elegantes diseños con iluminación de neón de la década de 1980 y más allá. 2. **Un vistazo al pasado: Juegos arcade icónicos de los 80:** - Este artículo presentaría un vistazo a la historia, el impacto y el legado de algunos de los juegos arcade más icónicos e influyentes de la década de 1980, destacando sus innovaciones, mecánicas populares y encanto perdurable. 3. **El auge y el renacimiento retro del pixel art:** - Este artículo profundizaría en la evolución del pixel art como estilo visual definitorio de los primeros juegos arcade, rastreando sus raíces, explorando sus técnicas y analizando el resurgimiento del pixel art en los juegos modernos y el arte digital. 		

Tabla 7. Un ejemplo de estímulo contextual

Incitación de retroceso

La incitación de retroceso es una técnica para mejorar el rendimiento al incitar al LLM a considerar primero una pregunta general relacionada con la tarea específica en cuestión y luego introducir la respuesta a esa pregunta general en una indicación posterior para la tarea específica. Este 'retroceso' permite al LLM activar los conocimientos previos y los procesos de razonamiento relevantes antes de intentar resolver el problema específico.

Al considerar los principios más amplios y subyacentes, los LLM pueden generar respuestas más precisas y perspicaces. La incitación de retroceso anima a los LLM a pensar críticamente y a aplicar sus conocimientos de maneras nuevas y creativas. Cambia la indicación final al realizar la tarea al utilizar más conocimiento en los parámetros del LLM del que entraría en juego cuando se le incita directamente.

Puede ayudar a mitigar los sesgos en las respuestas del LLM al centrarse en los principios generales en lugar de en los detalles específicos, la incitación

de retroceso

Echemos un vistazo a estos ejemplos para comprender cómo la indicación de retroceso puede mejorar los resultados. Primero revisemos una indicación tradicional (**Tabla 8**) antes de compararla con una indicación de retroceso (**Tabla 9**):

Objetivo	Escribe una historia para un nivel de un videojuego de disparos en primera persona.		
Modelo	gemini-pro		
Temperatura	1	Límite de fichas	1024
Top-K	40	Top-P	0.8
Indicación	Escribe una historia de un párrafo para un nuevo nivel de un videojuego de disparos en primera persona que sea desafiante y atractivo.		
Resultado	El nivel comienza con el escuadrón del jugador siendo emboscado en una densa zona urbana por una facción enemiga implacable. Los jugadores deben abrirse paso a través de un laberinto de callejones y edificios en ruinas, utilizando el sigilo y la precisión para eliminar a las fuerzas enemigas y evitar ser detectados. El jugador debe usar su entorno a su favor, colocando trampas, utilizando cobertura y realizando movimientos calculados para sobrevivir. En el camino, descubren información que sugiere que el enemigo está planeando un ataque devastador, y el jugador debe correr contra el tiempo para reunir pruebas y transmitirlas a su cuartel general antes de que el enemigo pueda llevar a cabo sus planes.		

Tabla 8. Una indicación tradicional antes de compararla con una indicación de paso atrás

Tabla 8. Una indicación tradicional antes de compararla con una indicación de paso atrás

Cuando estableces la temperatura en 1, podrías terminar con todo tipo de escritos creativos para una historia, pero también es bastante aleatorio y genérico. Así que demos un paso atrás:

Indicación	Basado en los populares juegos de acción de disparos en primera persona, ¿cuáles son 5 escenarios ficticios clave que contribuyen a una historia de niveles desafiante y atractiva en un videojuego de disparos en primera persona?
Resultado	<p>1. Base Militar Abandonada: Un extenso complejo militar postapocalíptico repleto de soldados mutantes y robots rebeldes, ideal para desafiantes combates con armas de fuego.</p> <p>2. Ciudad Ciberpunk: Un entorno urbano futurista con iluminación de neón, imponentes rascacielos y densos callejones, con enemigos mejorados cibernéticamente y mecánicas de piratería.</p> <p>3. Nave Espacial Alienígena: Una enorme nave alienígena varada en la Tierra, con pasillos inquietantes, secciones de gravedad cero y criaturas extraterrestres que encontrar.</p> <p>4. Pueblo Infestado de Zombis: Un pueblo desolado invadido por hordas de zombis agresivos, con intensos combates cuerpo a cuerpo y resolución de acertijos para encontrar un paso seguro.</p> <p>5. Instalación de Investigación Submarina: Un laboratorio en las profundidades marinas inundado de agua, lleno de criaturas acuáticas mutadas, y que requiere sigilo y habilidades de exploración submarina para sobrevivir.</p>

Tabla 9. Un ejemplo de indicaciones para la autoconsistencia

Tabla 9. Un ejemplo de indicaciones para la autoconsistencia

Sí, esos temas parecen adecuados para un videojuego en primera persona. Volvamos a la indicación original, pero esta vez incluimos la respuesta del paso atrás como contexto y vemos qué nos devuelve

Indicación	<p>Contexto: 5 temas atractivos para un videojuego de disparos en primera persona:</p> <ol style="list-style-type: none"> 1. **Base Militar Abandonada**: Un extenso complejo militar postapocalíptico repleto de soldados mutantes y robots rebeldes, ideal para desafiantes combates con armas de fuego. 2. **Ciudad Ciberpunk**: Un entorno urbano futurista con iluminación de neón con imponentes rascacielos y densos callejones, con enemigos mejorados cibernéticamente y mecánicas de piratería. 3. **Nave Espacial Alienigena**: Una enorme nave alienigena varada en la Tierra, con pasillos espeluznantes, secciones de gravedad cero y criaturas extraterrestres que encontrar. 4. **Pueblo Infestado de Zombis**: Un pueblo desolado invadido por hordas de zombis agresivos, con intensos combates cuerpo a cuerpo y resolución de acertijos para encontrar un paso seguro. 5. **Instalación de Investigación Submarina**: Un laboratorio en las profundidades marinas inundado de agua, lleno de criaturas acuáticas mutantes, que requiere sigilo y habilidades de exploración submarina para sobrevivir <p>Toma uno de los temas y escribe una historia de un párrafo para un nuevo nivel de videojuego de disparos en primera persona que sea desafiante y atractivo.</p>
Salida	<p>En el corazón de un abismo turbio, se encuentra una instalación de investigación submarina en ruinas, que se erige como un testimonio de la ambición humana y sus desastrosas consecuencias. Envuelta en la oscuridad, palpitando con el zumbido de la maquinaria defectuosa y atormentada por los ecos de las almas ahogadas, la instalación es ahora un caldo de cultivo para una pesadilla surgida de las profundidades marinas. El jugador, un marine de élite equipado con equipo de buceo avanzado y armamento experimental, tiene la tarea de aventurarse en este reino acuático de terror. Armados con resiliencia e ingenio, deben navegar por pasillos traicioneros, resolver acertijos críticos y enfrentarse a horripilantes monstruos marinos que acechan en las profundidades. Desde lidiar con la aplastante presión de las profundidades marinas hasta burlar a los astutos depredadores acuáticos, cada momento en este inframundo inexplorado pone a prueba los límites de la resistencia y el coraje humanos.</p>

Tabla 10. Un ejemplo de indicaciones para la autoconsistencia

Tabla 10. Un ejemplo de indicaciones para la autoconsistencia

¡Parece un videojuego interesante! Al usar técnicas de indicaciones de paso atrás, puedes aumentar la precisión de tus indicaciones

Cadena de Pensamiento (CdP) - Chaon of Thought(CoT)

La incitación en cadena de pensamiento (CdP) es una técnica para mejorar las capacidades de razonamiento de los LLM mediante la generación de pasos de razonamiento intermedios. Esto ayuda al LLM a generar respuestas más precisas. Puedes combinarla con la incitación de pocas oportunidades (**few-shot prompting**) para obtener mejores resultados en tareas más complejas que requieren razonamiento antes de responder, ya que es un desafío con una cadena de pensamiento de cero oportunidades (**zero-shot**).

La **CdP** tiene muchas ventajas. En primer lugar, requiere poco esfuerzo, es muy eficaz y funciona bien con los LLM estándar (por lo que no es necesario ajustarlo). También obtienes interpretabilidad con la incitación de la CdP, ya que puedes aprender de las respuestas del LLM y ver los pasos de razonamiento que se siguieron. Si hay un mal funcionamiento, podrás identificarlo. La cadena de pensamiento parece mejorar la robustez al moverse entre diferentes versiones del LLM. Lo que significa que el rendimiento de tu incitación debería variar menos entre diferentes LLM que si tu incitación no utiliza cadenas de razonamiento. Por supuesto, también hay desventajas, pero son algo intuitivas.

La respuesta LLM incluye el razonamiento en cadena de pensamiento, lo que significa más tokens de salida, lo que significa que las predicciones cuestan más dinero y tardan más.

Para explicar el siguiente ejemplo en la Tabla 11, primero intentemos crear una indicación que no utilice la indicación CdP (en inglés CoT) para mostrar las fallas de un modelo de lenguaje grande.

Indicación	Cuando tenía 3 años, mi pareja tenía el triple de mi edad. Ahora tengo 20 años. ¿Cuántos años tiene mi pareja?
Salida	63 años

Tabla 11. Un ejemplo de una indicación que intenta resolver un problema matemático

Tabla 11. Un ejemplo de una indicación que intenta resolver un problema matemático

¡Vaya! Obviamente, esa es la respuesta incorrecta. De hecho, los LLM a menudo tienen dificultades con las tareas matemáticas y pueden proporcionar respuestas incorrectas, incluso para una tarea tan simple como multiplicar dos números. Esto se debe a que están capacitados con grandes volúmenes de texto y las matemáticas pueden requerir un enfoque diferente.

Así que veamos si los pasos de razonamiento intermedios mejoran el resultado.

Indicación	Cuando tenía 3 años, mi pareja tenía 3 veces mi edad. Ahora tengo 20 años. ¿Cuántos años tiene mi pareja? Pensemos paso a paso.
Resultado	<ol style="list-style-type: none">1. Supongamos que mi edad actual es "x" años.2. Cuando tenía 3 años, mi edad era de 3 años.3. La edad de mi pareja en ese momento era 3 veces mi edad, lo que significa que la edad de mi pareja era $3 \times 3 = 9$ años.4. Ahora tengo 20 años, lo que significa que mi edad ha aumentado en $20 - 3 = 17$ años desde que tenía 3 años.5. Dado que la edad de mi pareja era de 9 años cuando yo tenía 3, y mi edad ha aumentado en 17 años, la edad de mi pareja también debe haber aumentado en 17 años.6. Por lo tanto, la edad actual de mi pareja es $9 + 17 = 26$ años. <p>Entonces, mi pareja tiene 26 años.</p>

Tabla 12. Un ejemplo de incitación a la cadena de pensamiento

Tabla 12. Un ejemplo de incitación a la cadena de pensamiento

Muy bien, ahora la respuesta final es correcta. Esto se debe a que le indicamos claramente al LLM que explicara cada paso en lugar de simplemente devolver una respuesta. Es interesante ver que el modelo suma los 17 años de aumento. En mi cabeza, tomaría la diferencia de años entre mi pareja y yo y los sumaría. ($20 + (9 - 3)$). Ayudemos al modelo a pensar un poco más como yo.

La **Tabla 12** es un ejemplo de una cadena de pensamiento de "disparo cero". La indicación de la cadena de pensamiento puede ser muy poderosa cuando se combina con un disparo único o con varios disparos, como se puede ver en la Tabla 13:

Indicación	P: Cuando mi hermano tenía 2 años, yo tenía el doble de su edad. Ahora tengo 40 años. ¿Cuántos años tiene mi hermano? Pensemos paso a paso. R: Cuando mi hermano tenía 2 años, yo tenía 2 ± 4 años. Esa es una diferencia de edad de 2 años y soy mayor. Ahora tengo 40 años, por lo que mi hermano tiene $40 - 2 = 38$ años. La respuesta es 38. P: Cuando tenía 3 años, mi pareja tenía el triple de mi edad. Ahora tengo 20 años. ¿Cuántos años tiene mi pareja? Pensemos paso a paso. R:
Salida	Cuando tenía 3 años, mi pareja tenía $3*3 = 9$ años. Esa es una diferencia de edad de 6 años y mi pareja es mayor. Ahora tengo 20 años, así que mi pareja tiene $20 + 6 = 26$ años. La respuesta es 26.

Tabla 13. Un ejemplo de incitación por cadena de pensamiento con una sola ejecución

Tabla 13. Un ejemplo de incitación por cadena de pensamiento con una sola ejecución

La cadena de pensamiento puede ser útil para varios casos de uso. Piense en la generación de código, para dividir la solicitud en unos pocos pasos y asignarlos a líneas de código específicas. O para crear datos sintéticos cuando tiene algún tipo de semilla como "El producto se llama XYZ, escriba una descripción que guíe al modelo a través de las suposiciones que haría basándose en el título dado del producto". Generalmente, cualquier tarea que se pueda resolver "hablando" es un buen candidato para una cadena de pensamiento. **Si puede explicar los pasos para resolver el problema, pruebe la cadena de pensamiento.**

Consulte el cuaderno alojado en el repositorio de Github de Google Cloud Platform, que brindará más detalles sobre la incitación por cadena de pensamiento:

En la sección de mejores prácticas de este capítulo, aprenderemos algunas de las mejores prácticas específicas para la estimulación de la cadena de pensamiento.

Autoconsistencia

Si bien los modelos de lenguaje grandes (LLM) han demostrado un éxito impresionante en varias tareas de PNL, su capacidad de razonamiento a menudo se considera una limitación que no se puede superar únicamente aumentando el tamaño del modelo. Como aprendimos en la sección anterior sobre la estimulación de la cadena de pensamiento, se puede incitar al modelo a generar pasos de razonamiento como un humano que resuelve un problema. Sin embargo, CoT utiliza una estrategia simple de "**decodificación voraz**", lo que limita su eficacia. La "autoconsistencia" combina el muestreo y la votación por mayoría para generar diversas rutas de razonamiento y seleccionar la respuesta más consistente. Mejora la precisión y la coherencia de las respuestas generadas por los modelos de lenguaje.

La autoconsistencia proporciona una **pseudoprobabilidad** de que una respuesta sea correcta, pero obviamente tiene altos costos.

pseudoprobabilidad

La idea básica de la pseudoaleatoriedad es tomar una pequeña cantidad de aleatoriedad descrita por una semilla extraída de una distribución de probabilidad fuente dada y producir de forma determinista, a partir de la semilla, un mayor número de bits aparentemente aleatorios que simulan una muestra extraída de otra distribución de probabilidad objetivo en un espacio de rangos. Es decir, aumenta la aleatoriedad .

National Academies of Sciences, Engineering, and Medicine. 1992. Probability and Algorithms. Washington, DC: The National Academies Press. <https://nap.nationalacademies.org/read/2026/chapter/7#72>.

Sigue los siguientes pasos:

1. **Generación de diversas rutas de razonamiento:** El modelo de lenguaje recibe la misma indicación varias veces. Un ajuste de temperatura alto anima al modelo a generar diferentes rutas de razonamiento y perspectivas sobre el problema
2. Extraiga la respuesta de cada respuesta generada.
3. Elija la respuesta más común.

Veamos un ejemplo de un sistema de clasificación de correo electrónico, que clasifica un correo electrónico como IMPORTANTE o NO IMPORTANTE. Se enviará una secuencia de pensamiento de cero intentos al LLM varias veces para ver si las respuestas difieren después de cada envío. Observe el tono amable, la elección de palabras y el sarcasmo que se ha utilizado en el correo electrónico. Todo esto podría engañar al LLM.

Aviso	
	***** Hola, He visto que usas WordPress para tu sitio web. Un excelente sistema de gestión de contenido de código abierto. Yo también lo he usado en el pasado. Viene con muchos complementos de usuario excelentes. Y es bastante fácil de configurar. Noté un error en el formulario de contacto, que ocurre cuando seleccionas el campo de nombre. Mira la captura de pantalla adjunta de mí ingresando texto en el campo de nombre. Observa el cuadro de alerta de JavaScript que invoqué. Pero por lo demás, es un gran sitio web. Disfruto leyéndolo. Siéntete libre de dejar el error en el sitio web, porque me da cosas más interesantes para leer. Saludos, Harry el Hacker. *** Clasifica el correo electrónico anterior como IMPORTANTE o NO IMPORTANTE. Pensemos paso a paso y expliquemos por qué.
	Continúa en la página siguiente...

Salida Intento 1	<p>**Paso 1: Identificar el propósito del correo electrónico.**</p> <p>El propósito del correo electrónico es informar al destinatario de un error en el formulario de contacto de su sitio web. El remitente también menciona que disfruta leyendo el sitio web y que dejará el error en su lugar por el momento.</p> <p>**Paso 2: Evaluar el impacto potencial del error.**</p> <p>El error en el formulario de contacto podría permitir a un atacante ejecutar código JavaScript arbitrario en el sitio web. Esto podría llevar a que el atacante pueda tomar el control del sitio web, robar información confidencial o desfigurarlo.</p> <p>**Paso 3: Considerar la credibilidad del remitente.**</p> <p>El remitente del correo electrónico afirma ser un hacker llamado Harry. Sin embargo, no hay forma de verificar la identidad del remitente. Es posible que el remitente sea un hacker legítimo que intente ayudar al destinatario, pero también es posible que sea un actor malicioso que intente engañar al destinatario para que haga clic en un enlace malicioso o descargue un archivo malicioso.</p> <p>**Conclusión: IMPORTANTE**</p> <p>Según el impacto potencial del error y la credibilidad del remitente, el correo electrónico debe clasificarse como “IMPORTANTE”. El destinatario debe tomar medidas inmediatas para corregir el error y proteger su sitio web de ataques.</p>
Intento de salida 2	<p>**Explicación:**</p> <ol style="list-style-type: none"> 1. **Falta de urgencia:** El correo electrónico no transmite ninguna sensación de urgencia ni de urgencia. Comienza con un saludo informal y presenta la información en un tono práctico. 2. **Informe de error no crítico:** El error informado en el formulario de contacto se describe como interesante, pero no crítico. El remitente del correo electrónico incluso sugiere que el propietario del sitio web puede dejar el error como está. 3. **Falta de impacto personal:** El correo electrónico no indica ningún impacto directo o personal en el propietario del sitio web. Simplemente menciona que el remitente disfruta leyendo el sitio web. <p style="text-align: right;">Continúa en la página siguiente...</p>

Intento de salida 2	<p>4. **Ausencia de solicitud de acción:** El correo electrónico no solicita explicitamente ninguna acción al propietario del sitio web. No solicita que se solucione el error ni proporciona sugerencias específicas de mejora.</p> <p>5. **Intención del remitente:** La intención del remitente parece centrarse más en compartir una observación y expresar interés en el sitio web que en plantear una preocupación seria o solicitar asistencia.</p> <p>**Conclusión: NO IMPORTANTE**</p>
Salida Intento 3	<p>**Paso 1: Identificar el propósito del correo electrónico.**</p> <p>El propósito del correo electrónico es informar al destinatario de un error en el formulario de contacto de su sitio web. El remitente también menciona que disfruta leyendo el sitio web y que dejará el error en su lugar por el momento.</p> <p>**Paso 2: Evaluar el impacto potencial del error.**</p> <p>El error en el formulario de contacto podría permitir que un atacante ejecute código JavaScript arbitrario en el sitio web. Esto podría llevar a que el atacante pueda tomar el control del sitio web, robar información confidencial o desfigurar el sitio web.</p> <p>**Paso 3: Considerar la credibilidad del remitente.**</p> <p>El remitente del correo electrónico afirma ser un hacker llamado Harry. Sin embargo, no hay forma de verificar la identidad del remitente. Es posible que el remitente sea un hacker legítimo que intenta ayudar al destinatario, pero también es posible que sea un actor malicioso que intenta explotar el error para su propio beneficio.</p> <p>**Conclusión:**</p> <p>Según los factores anteriores, el correo electrónico debe clasificarse como **IMPORTANTE**. El error en el formulario de contacto es un grave riesgo de seguridad y se desconoce la credibilidad del remitente. El destinatario debe tomar medidas inmediatas para corregir el error y</p>

Tabla 14. Un ejemplo de solicitud de autoconsistencia

Puedes usar el mensaje anterior para comprobar si la clasificación es consistente. Según el modelo que uses y la configuración de temperatura, podría mostrar "IMPORTANTE" o "NO IMPORTANTE".

Al generar muchas cadenas de pensamientos y tomar la respuesta más común ("IMPORTANTE"), podemos obtener una respuesta correcta más consistente del LLM.

Este ejemplo muestra cómo se puede utilizar la incitación de autoconsistencia para mejorar la precisión de la respuesta de un LLM al considerar múltiples perspectivas y seleccionar la respuesta más consistente.

Árbol de Pensamientos (ToT) - Tree of Thoughts (ToT)

Ahora que estamos familiarizados con la cadena de pensamiento y la incitación de autoconsistencia, revisemos el **Árbol de Pensamientos** (ToT). Generaliza el concepto de incitación de CoT porque permite a los LLMS explorar múltiples caminos de razonamiento diferentes simultáneamente, en

lugar de simplemente seguir una única cadena lineal de pensamiento. Esto se muestra en la Figura 1.

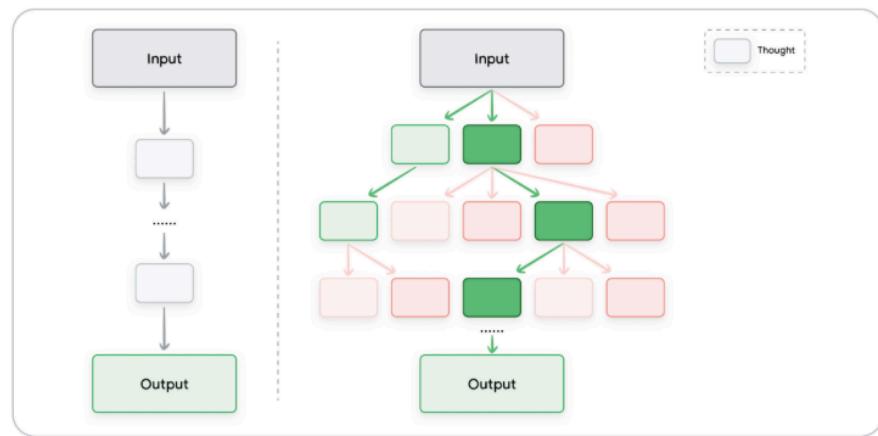


Figura 1. Una visualización de la incitación de la cadena de pensamiento a la izquierda versus la incitación del Árbol de Pensamientos a la derecha

Este enfoque hace que **ToT** sea particularmente adecuado para tareas complejas que requieren exploración.

Funciona manteniendo un árbol de pensamientos, donde cada pensamiento representa una secuencia de lenguaje coherente que sirve como paso intermedio hacia la resolución de un problema.

El modelo puede entonces explorar diferentes rutas de razonamiento ramificándose desde diferentes nodos del árbol.

Hay un excelente cuaderno que detalla un poco más el Árbol del Pensamiento (ToT), que se basa en el artículo «Árbol del Pensamiento Guiado por Modelos de Lenguaje Grandes».

ReAct (razonar y actuar)

La incitación a razonar y actuar (ReAct) [10] es un paradigma que permite a los LLM resolver tareas complejas utilizando el razonamiento del lenguaje natural combinado con herramientas externas (búsqueda, intérprete de código, etc.), lo que permite al LLM realizar ciertas acciones, como

interactuar con API externas para recuperar información, lo cual es un primer paso hacia el modelado de agentes.

ReAct imita cómo operan los humanos en el mundo real, ya que razonamos verbalmente y podemos tomar medidas para obtener información. ReAct funciona bien en comparación con otros enfoques de ingeniería de indicaciones en una variedad de dominios.

La incitación de ReAct funciona combinando el razonamiento y la acción en un ciclo de pensamiento-acción. El LLM primero razona sobre el problema y genera un plan de acción. Luego, realiza las acciones del plan y observa los resultados. El LLM luego utiliza las observaciones para actualizar su razonamiento y generar un nuevo plan de acción. Este proceso continúa hasta que el LLM llega a una solución al problema.

Para ver esto en acción, necesitas escribir algo de código. En el fragmento de código 11, estoy usando el marco langchain para Python, junto con VertexAI (google-cloud-aiplatform) y los paquetes pip google-search-results

Para ejecutar este ejemplo, debe crear una clave SerpAPI (gratuita) desde <https://serpapi.com/manage-api-key> y establecer una variable de entorno SERPAPI_API_KEY.

A continuación, escribamos código Python, con la tarea de que el LLM averigüe: ¿Cuántos hijos tiene un padre famoso que toca en la banda Metallica?

Python

```
from langchain.agents import load_tools
from langchain.agents import initialize_agent
from langchain.agents import AgentType
from langchain.llms import VertexAI

prompt = "¿Cuántos hijos tienen los miembros de la banda Metallica?"

llm = VertexAI(temperature=0.1)
tools = load_tools(["serpapi"], llm=llm)

agent = initialize_agent(tools, llm,
agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True)
agent.run(prompt)
```

Fragmento 1. Creación de un agente ReAct con LangChain y VertexAI

El fragmento de código 2 muestra el resultado. Observe que ReAct realiza una cadena de cinco búsquedas. De hecho, el LLM está extrayendo los resultados de búsqueda de Google para averiguar los nombres de las bandas. Luego, enumera los resultados como observaciones y encadena la idea para la siguiente búsqueda.

```
> Ingresando nueva cadena AgentExecutor...
Metallica tiene 4 miembros.
Acción: Buscar
Acción: ¿Cuántos hijos tiene James Hetfield?
Observación: tres hijos
Pensamiento: 1/4 de los miembros de la banda Metallica tienen 3 hijos
Acción: Buscar
Acción: ¿Cuántos hijos tiene Lars Ulrich?
Observación: 3
Pensamiento: 2/4 de los miembros de la banda Metallica tienen 6 hijos
Acción: Buscar
Acción: ¿Cuántos hijos tiene Kirk Hammett?
Observación: Hammett ha estado casado con su segunda esposa, Lani, desde 1998.
Tienen dos hijos, Angel (nacido el 29 de septiembre de 2006) y Vincenzo (nacido el 28 de junio de 2008).
Pensamiento: 3/4 de los miembros de la banda Metallica tienen 8 hijos
Acción: Buscar
Acción: ¿Cuántos hijos tiene Robert Trujillo?
Observación: 2
Pensamiento: 4/4 de los miembros de la banda Metallica tienen 10 hijos
Respuesta final: 10
```

Fragmento 2. Ejecución del agente ReAct en la terminal

El fragmento de código 2 determina que la banda Metallica tiene cuatro miembros. Luego, busca a cada miembro de la banda para solicitar el total

de hijos y sumar el total. Finalmente, devuelve el número total de hijos como respuesta final.

En la práctica, las indicaciones de ReAct requieren comprender que continuamente hay que reenviar las indicaciones/respuestas anteriores (y recortar el contenido generado adicional), así como configurar el modelo con ejemplos/instrucciones adecuados. Consulta el cuaderno¹¹ alojado en el repositorio de Github de GoogleCloudPlatform, que ofrece un poco más de detalle y muestra las entradas y salidas reales de LLM con un ejemplo más elaborado

Ingeniería Automática de Indicaciones

En este punto, podrías darte cuenta de que escribir una indicación puede ser complejo. **¿No sería bueno automatizar esto (escribir una indicación para escribir indicaciones)?** Bueno, hay un método: Ingeniería Automática de Indicaciones (APE). Este método no solo alivia la necesidad de intervención humana, sino que también mejora el rendimiento del modelo en varias tareas.

Indicarás a un modelo que genere más indicaciones. Evalúalas, posiblemente modifica las buenas. Y repite.

Por ejemplo, puedes usar la ingeniería automática de indicaciones que ayudará a entrenar un chatbot para una tienda web de camisetas. Queremos descubrir todas las diferentes maneras en que los clientes podrían formular su pedido de compra de una camiseta de un grupo musical.

1. Escribe la indicación que generará las variantes de salida. En este ejemplo, estoy usando gemini-pro para generar 10 instrucciones. Consulta la Tabla 15:

a.

Indicación	Tenemos una tienda online de camisetas de artículos de bandas y, para entrenar a un chatbot, necesitamos varias formas de hacer pedidos: "Una camiseta de Metallica talla S". Genera 10 variantes, con la misma semántica pero con el mismo significado.
Salida	<p>1. Me gustaría comprar una camiseta de Metallica talla pequeña.</p> <p>2. ¿Puedo pedir una camiseta de Metallica talla pequeña?</p> <p>3. Me gustaría comprar una camiseta de Metallica talla pequeña.</p> <p>4. Una camiseta de Metallica talla pequeña, por favor.</p> <p>5. Quiero una camiseta de Metallica talla pequeña.</p> <p>6. Me gustaría hacer un pedido de una camiseta de Metallica talla pequeña.</p> <p>7. Una camiseta pequeña de Metallica, por favor.</p> <p>8. Busco una camiseta de Metallica talla pequeña.</p> <p>9. ¿Puedes conseguirme una camiseta de Metallica talla pequeña?</p> <p>10. Una camiseta pequeña de Metallica, por favor</p>

Tabla 15. Usar un LLM para generar indicaciones

2. Evalúe a todos los candidatos a la instrucción puntuándolos según una métrica elegida. Por ejemplo, puede usar BLEU (Estudiante de Evaluación Bilingüe) o ROUGE (Estudiante Orientado a la Memoria para la Evaluación de Gist).
3. Seleccione al candidato a la instrucción con la puntuación de evaluación más alta. Este candidato será la indicación final que podrá usar en su aplicación de software o chatbot. También puede ajustar la indicación seleccionada y evaluar de nuevo

Indicación de código

Gemini se centra principalmente en indicaciones basadas en texto, lo que también incluye la escritura de indicaciones para devolver código. Vayamos a Vertex AI Studio y probemos estas indicaciones para ver algunos ejemplos de codificación.

Indicaciones para escribir código

Gemini también puede ser un desarrollador y ayudarte a escribir código en cualquier lenguaje de programación que elijas. Como desarrollador, esto puede ayudarte a acelerar el proceso de escritura de código.

Imagina una carpeta en tu máquina con cientos de archivos que necesitan ser renombrados. Renombrar cada archivo te llevaría mucho tiempo. Sabes un poco de Bash y podrías escribir un script para automatizar esto, pero eso

también puede llevar un tiempo. Así que escribamos una indicación. Puedes escribir la indicación en el chatbot público para consumidores Gemini, o si te preocupa más la confidencialidad, puedes escribir estas indicaciones dentro de tu cuenta de Google Cloud y abrir Vertex AI Studio. La ventaja de Vertex AI Studio es que puedes configurar la temperatura, etc

Name	1_bash_rename_files_in_folder		
Goal	Write a prompt to write code in Bash to rename files in a folder.		
Model	gemini-pro		
Temperature	0.1	Token Limit	1024
Top-K	N/A	Top-P	1
Prompt	Write a code snippet in Bash, which asks for a folder name. Then it takes the contents of the folder and renames all the files inside by prepending the name draft to the file name.		
Output	<pre>```bash #!/bin/bash # Ask for the folder name echo "Enter the folder name: " read folder_name # Check if the folder exists if [! -d "\$folder_name"]; then echo "Folder does not exist." exit 1 fi # Get the list of files in the folder files=("\$folder_name"/*) # Rename each file by prepending "draft" to the file name for file in "\${files[@]}"; do new_file_name="draft_\${basename "\$file"}" mv "\$file" "\$new_file_name" done echo "Files renamed successfully." ``` </pre>		

Nombre	1_bash_renombrar_archivos_en_carpeta		
Objetivo	Escribir un mensaje para escribir código en Bash para renombrar los archivos en una carpeta.		
Modelo	gemini-pro		
Temperatura	0.1	Límite de tokens	1024
Top-K	NO	Top-P	1
Mensaje	Escribir un fragmento de código en Bash, que solicita un nombre de carpeta. Luego toma el contenido de la carpeta y renombra a todos los archivos dentro anteponiendo el nombre borrador al nombre del archivo.		
Salida	<pre>```bash #!/bin/bash # Solicitar el nombre de la carpeta echo "Ingrese el nombre de la carpeta: " read nombre_de_carpeta # Comprobar si la carpeta existe if ! -d ["\$nombre_de_carpeta"]; then echo "La carpeta no existe." exit 1 fi # Obtener la lista de archivos en la carpeta files=("\$nombre_de_carpeta"/*) # Renombrar cada archivo anteponiendo "borrador" al nombre del archivo # para el archivo en "\${files[@]}"; hacer new_file_name="draft_\$(basename "\$file")" mv "\$file" "\$new_file_name" hecho echo "Archivos renombrados correctamente." ``` </pre>		

Tabla 16. Uso de Gemini para programar código

Tabla 16. Uso de Gemini para programar código

Me parece un buen código, ¡incluso está documentado! Sin embargo, dado que los LLM no pueden razonar ni repetir datos de entrenamiento, es esencial leer y probar el código primero.

El momento que todos estamos esperando, ¿realmente funciona?

Intentémoslo primero con una carpeta de prueba con solo unos pocos archivos, que deberán renombrarse de filename.txt a draft_filename.txt.

1. Copie la salida de la Tabla 16 (sin el contenedor de texto "bash") y péguela en un nuevo archivo llamado: "rename_files.sh".

1. Abra una ventana de terminal y escriba: rename_files.sh. Le pedirá que ingrese un nombre de carpeta, por ejemplo, test., y presione Enter.
2. El script parece ejecutarse bien. Verá el mensaje: Archivos renombrados correctamente.

Cuando mire dentro de la carpeta de prueba, observe que todos los archivos fueron renombrados perfectamente a draft_filename.txt.

¡Funcionó!

Instrucciones para explicar el código

Como desarrollador, cuando trabajas en equipo, tienes que leer el código de otra persona. Gemini también puede ayudarte con esto. Tomemos la salida del código de la Tabla 16, eliminemos los comentarios y pidamos al modelo de lenguaje grande que explique qué está sucediendo. Véase la Tabla 17:

Nombre	1_bash_explica_el_código		
Objetivo	Escribe una instrucción para explicar el código Bash		
Modelo	gemini-pro		
Temperatura	0.1	Límite de tokens	1024
Top-K	N/D	Top-P	1
Indicación	<p>Explicame el siguiente código Bash:</p> <pre>*** #!/bin/bash echo "Introduce el nombre de la carpeta: " * read nombre_de_carpeta if [-d "\$nombre_de_carpeta"]; then echo "La carpeta no existe." * exit 1 fi files=("\$nombre_de_carpeta") * for file in "\${files[@]}"; do new_file_name="draft_\$(basename \$file)" mv "\$file" "\$new_file_name" done echo "Archivos renombrados correctamente." ***</pre>		
Salida	<p>Este script Bash realiza las siguientes tareas:</p> <ol style="list-style-type: none"> 1. Entrada del usuario: - Sigue al usuario que introduzca un nombre de carpeta mediante los comandos 'echo' y 'read'. El nombre de carpeta introducido se almacena en la variable 'folder_name'. 2. Comprobación de existencia de carpeta: - Comprueba si la carpeta con el nombre especificado existe mediante la condición '[-d "\$folder_name"]'. Si la carpeta no existe, muestra un mensaje de error ("La carpeta no existe") y sale del script con un código de salida de 1. <p>Continúa en la página siguiente...</p>		

Salida	<p>3. **Listado de archivos**:</p> <ul style="list-style-type: none"> - Enumera todos los archivos dentro de la carpeta especificada usando el comando <code>'files=("\$folder_name"/*)'</code>. Esto crea una matriz llamada 'files' que contiene las rutas a todos los archivos en la carpeta. <p>4. **Cambio de nombre de archivo**:</p> <ul style="list-style-type: none"> - Itera a través de cada archivo en la matriz 'files' usando un bucle 'for'. - Para cada archivo, genera un nuevo nombre de archivo usando el comando <code>'new_file_name="draft_\${basename "\$file"}"</code>. Este nuevo nombre de archivo se crea prefijando el nombre del archivo original con la cadena "draft_". - Luego, cambia el nombre del archivo usando el comando <code>'mv "\$file" "\$new_file_name"</code>, que mueve el archivo al nuevo nombre de archivo. <p>5. **Mensaje de éxito**:</p> <ul style="list-style-type: none"> - Despues de cambiar el nombre de todos los archivos, muestra un mensaje de éxito ("Archivos renombrados correctamente.") para indicar que la operación de cambio de nombre fue exitosa
--------	--

Tabla 17. Una indicación para explicar el código

Tabla 17. Una indicación para explicar el código

Indicaciones para traducir el código

El código bash de la Tabla 16 parece funcionar bien. Sin embargo, este script podría ser realmente reutilizable cuando me solicite nombres de archivo; idealmente, debería funcionar como una aplicación separada con una interfaz de usuario. Como punto de partida, Python sería un mejor lenguaje para una aplicación (web) que Bash. Los LLM pueden ayudar con la traducción de código de un lenguaje a otro. Vea el ejemplo en

Tabla 18:

Nombre	1_python_rename_files		
Objetivo	Escribir un mensaje para traducir el código Bash a Python		
Modelo	gemini-pro		
Temperatura	0.1	Límite de tokens	1024
Top-K	NO	Top-P	1
Mensaje	<p>Traduce el siguiente código Bash a un fragmento de Python.</p> <pre>'''bash #!/bin/bash echo "Introduce el nombre de la carpeta: " read nombre_de_carpeta if [-d "\$nombre_de_carpeta"]; then echo "La carpeta existe." exit 1 fi files=("\$nombre_de_carpeta/*") for file in "\${files[@]}"; do new_file_name="draft_\${basename "\$file"}" mv "\$file" "\$new_file_name" done echo "Archivos renombrados correctamente." '''</pre>		
Salida	<pre>'''python import os import shutil # Obtener el nombre de la carpeta del usuario nombre_carpeta = input("Introduzca el nombre de la carpeta: ") # Comprobar si la carpeta existe si no existe os.path.isdir(nombre_carpeta): print("La carpeta no existe.") exit(1)</pre>		
Continúa en la página siguiente...			

Salida	<pre># Obtener una lista de todos los archivos en la carpeta files = os.listdir(nombre_de_carpeta) # Iterar sobre los archivos para archivo en archivos: # Crear un nuevo nombre de archivo con el prefijo "borrador_" nuevo_nombre_de_archivo = f"borrador_{archivo}" # Mover el archivo al nuevo nombre shutil.move(os.path.join(nombre_de_carpeta, archivo), os.path.join(nombre_de_carpeta, nuevo_nombre_de_archivo)) # Imprimir un mensaje de éxito print("Archivos renombrados correctamente.") '''</pre>
--------	--

Tabla 18. Un mensaje para traducir código de Bash a Python

Tabla 18. Un mensaje para traducir código de Bash a Python

Lea y revise el código. Copie la salida del mensaje y péguela en un nuevo archivo: file_renamer.py. Pruebe el código abriendo una ventana de Terminal y ejecute el siguiente comando python file_renamer.py

NOTA: Al solicitar código (Python) en Language Studio en Vertex AI, tendrá que hacer clic en el botón 'Markdown'. De lo contrario, recibirá texto sin formato al que le falta la sangría de línea adecuada, lo cual es importante para ejecutar código Python.

Solicitudes para depurar y revisar código

Escribamos manualmente algunas modificaciones en el código de la Tabla 18. Debería solicitar al usuario el prefijo del nombre de archivo y escribir este prefijo en mayúsculas. Vea el código de ejemplo en el fragmento *3, pero qué fastidio. ¡Ahora devuelve errores de Python!

```
Python

import os
import shutil

nombre_de_carpeta = input("Introduzca el nombre de la carpeta: ")
prefijo = input("Introduzca la cadena que se antepondrá al nombre del archivo: ")
text = toUpperCase(prefijo)

if not os.path.isdir(nombre_de_carpeta):
    print("La carpeta no existe.")
    exit(1)

files = os.listdir(nombre_de_carpeta)

for file in files:
    nuevo_nombre_de_archivo = f'{text}_{file}'

    shutil.move(os.path.join(nombre_de_carpeta, archivo),
                os.path.join(nombre_de_carpeta, nuevo_nombre_de_archivo))

print("Archivos renombrados correctamente.")
```

Fragmento 3. Un script de Python roto

Fragmento 3. Un script de Python roto

¡Vaya! Parece un error:

```
El siguiente código de Python da un error:
Rastreo (última llamada reciente):
  Archivo "/Users/leeboonstra/Documents/test_folder/ rename_files.py", línea 7,
en <módulo>
    texto = toUpperCase(prefijo)
NameError: el nombre 'toUpperCase' no está definido
```

Fragmento 4. Rompi el código Python

Fragmento 4. Rompí el código Python

Veamos si podemos pedirle al modelo de lenguaje grande que depure y revise el código. Eche un vistazo a

Tabla 19:

Nombre	l_codigo_de_depuracion_de_python		
Objetivo	Escribir un mensaje para depurar y revisar el código Python.		
Modelo	gemini-pro		
Temperatura 0.1		Límite de tokens	1024
Top-K	N/D	Top-P	1
Mensaje	<p>El siguiente código Python da un error:</p> <p>Rastreo (última llamada reciente):</p> <p>Archivo "/Users/leeboonstra/Documents/test_folder/rename_files.py", línea 7, en <módulo></p> <pre>text = toUpperCase(prefix)</pre> <p>NameError: el nombre 'toUpperCase' no está definido</p> <p>Depurar lo que está mal y explicar cómo puedo mejorar el código.</p> <pre>'''python import os import shutil # Obtener el nombre de la carpeta del usuario nombre_carpeta = input("Ingrese el nombre de la carpeta: ") prefix = input("Ingrese la cadena que se antepondrá al nombre del archivo: ") text = toUpperCase(prefix) # Comprobar si la carpeta existe si no, os.path.isdir(nombre_de_carpeta): print("La carpeta no existe.") exit(1) # Obtener una lista de todos los archivos en la carpeta files = os.listdir(nombre_de_carpeta) # Iterar sobre los archivos para el archivo en los archivos:</pre>		
Continúa en la página siguiente...			

Aviso	<pre># Crea un nuevo nombre de archivo con el prefijo "draft_" new_filename = f"{text}_{file}" # Mueve el archivo al nuevo nombre shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_file_name)) # Imprime un mensaje de éxito print("Archivos renombrados correctamente.") ***</pre>
Salida	<p>El código llama a la función 'toUpperCase' para convertir 'prefix' a mayúsculas, pero esa función no está definida. Para solucionar este problema, puedes usar el método 'upper()' de la clase string, que convierte una cadena dada a mayúsculas. El código modificado se muestra a continuación:</p> <pre>'''python import os import shutil #Obtén el nombre de la carpeta del usuario folder_name = input("Introduce el nombre de la carpeta: ") prefix = input("Introduce la cadena que se antepondrá al nombre del archivo: ") text = prefix.upper() # Comprueba si la carpeta existe if not os.path.isdir(nombre_de_carpeta): print("La carpeta no existe.") exit(1) #Obtener una lista de todos los archivos en la carpeta files = os.listdir(nombre_de_carpeta) # Iterar sobre los archivos for file in files: # Crear un nuevo nombre de archivo con el prefijo "draft_" new_filename = f"{text}_{file}" #Mover el archivo al nuevo nombre shutil.move(os.path.join(nombre_de_carpeta, file),</pre>

Continúa en la página siguiente...

<p>Salida</p> <pre>os.path.join(nombre_de_carpeta, nuevo_nombre_de_archivo)) # Imprimir un mensaje de éxito print("Archivos renombrados correctamente.") """ Además, hay algunas otras mejoras que se pueden realizar en el código: 1. La extensión de los nuevos nombres de archivo no se conserva. Es mejor extraer la extensión del nombre de archivo original y añadirla al nuevo nombre de archivo. 2. El código asume que no hay espacios en el nombre de la carpeta. Sin embargo, es una buena práctica manejar los espacios con elegancia reemplazándolos con guiones bajos o guiones. 3. Usar la sintaxis de cadena 'f' para la interpolación de cadenas es más legible y conciso que el operador '+' tradicional. 4. El código no maneja los errores que puedan ocurrir durante el proceso de cambio de nombre. Sería mejor envolver la llamada 'shutil.move' en un bloque 'try...except' para detectar cualquier error potencial. Aquí está el código mejorado con estas sugerencias:</pre> <pre>'''python import os import shutil # Obtener el nombre de la carpeta del usuario nombre_carpeta = input("Introduzca el nombre de la carpeta: ") prefijo = input("Introduzca la cadena que se antepondrá al nombre del archivo: ") # Comprobar si la carpeta existe si no, os.path.isdir(nombre_carpeta): print("La carpeta no existe.") exit(1) # Obtener una lista de todos los archivos de la carpeta archivos = os.listdir(nombre_carpeta) # Iterar sobre los archivos para archivo en archivos:</pre>	<p>Continúa en la página siguiente...</p>
---	---

<p>Salida</p> <pre># Obtener la extensión del archivo ext = os.path.splitext(archivo) [1] # Crear un nuevo nombre de archivo con el prefijo y la extensión del archivo new_filename = f'{prefijo}_{os.path.splitext(archivo)[0]}.{ext}' # Construir las rutas completas para los archivos original y nuevo old_path = os.path.join(nombre_de_carpeta, archivo) new_path = os.path.join(nombre_de_carpeta, nuevo_nombre_de_archivo) # Intentar mover el archivo try: shutil.move(old_path, new_path) except Exception as e: print(f"Se produjo un error al mover el archivo {archivo}: {e}") # Imprimir un mensaje de éxito print("Archivos renombrados correctamente.") """ Estas mejoras hacen que el código sea más robusto y flexible, a la vez que facilitan su lectura y comprensión... (La respuesta se truncó porque alcanzó el límite de tokens. Intente aumentar el límite de tokens si necesita una respuesta más larga).</pre>

Tabla 19. Una solicitud para depurar y revisar código Python

Eso es genial. No solo me dijo cómo resolver el problema, sino que también descubrió que mi código tenía más errores y cómo resolverlos. La última parte de la solicitud ofrecía sugerencias para mejorar el código en general.

¿Qué pasa con las indicaciones multimodales?

Las indicaciones para código siguen utilizando el mismo modelo de lenguaje grande y regular. Las indicaciones multimodales son un tema aparte; se refieren a una técnica en la que se utilizan múltiples formatos de entrada para guiar un modelo de lenguaje grande, en lugar de depender únicamente del texto. Esto puede incluir combinaciones de texto, imágenes, audio, código o incluso otros formatos, según las capacidades del modelo y la tarea en cuestión.

Mejores prácticas

Encontrar la indicación correcta requiere experimentación. Language Studio en Vertex AI es el lugar perfecto para experimentar con tus indicaciones, con la posibilidad de probar con los distintos modelos.

Utiliza las siguientes prácticas recomendadas para convertirte en un profesional de la ingeniería de indicaciones.

Proporcionar ejemplos

La práctica recomendada más importante es proporcionar ejemplos (de una sola muestra o de varias muestras) dentro de una indicación. Esto es muy eficaz porque actúa como una poderosa herramienta de enseñanza. Estos ejemplos muestran los resultados deseados o respuestas similares, lo que permite que el modelo aprenda de ellos y adapte su propia generación en consecuencia. Es como darle al modelo un punto de referencia o un objetivo al que apuntar, mejorando la precisión, el estilo y el tono de su respuesta para que se ajuste mejor a tus expectativas.

Diseña con simplicidad

Las indicaciones deben ser concisas, claras y fáciles de entender tanto para ti como para el modelo. Como regla general, si ya es confuso para ti, probablemente también lo será para el modelo. Intenta no usar un lenguaje complejo y no proporcionar información innecesaria.

Ejemplos:

ANTES:

Estoy visitando Nueva York ahora mismo y me gustaría saber más sobre lugares fantásticos. Estoy con dos niños de 3 años. ¿A dónde deberíamos ir durante nuestras vacaciones?

DESPUÉS REESCRIBIR:

Actúa como guía de viajes para turistas. Describe lugares fantásticos para visitar en Nueva York y Manhattan con un niño de 3 años.

Intenta usar verbos que describan la acción. Aquí tienes un conjunto de ejemplos:

Actuar, Analizar, Categorizar, Clasificar, Contrastar, Comparar, Crear, Describir, Definir, Evaluar, Extraer, Encontrar, Generar, Identificar, Enumerar, Medir, Organizar, Analizar, Elegir, Predecir, Proporcionar, Clasificar, Recomendar, Devolver, Recuperar, Reescribir, Seleccionar, Mostrar, Ordenar, Resumir, Traducir, Escribir.

Sea específico sobre el resultado

Sea específico sobre el resultado deseado. Una instrucción concisa podría no guiar el LLM lo suficiente o podría ser demasiado genérica. Proporcionar detalles específicos en la indicación (a través de indicaciones del sistema o del contexto) puede ayudar al modelo a centrarse en lo que es relevante, mejorando la precisión general.

Ejemplos:

HACER:

Generar una entrada de blog de 3 párrafos sobre las 5 mejores consolas de videojuegos. La entrada del blog debe ser informativa y atractiva, y debe estar escrita en un estilo conversacional.

NO HACER:

Generar una entrada de blog sobre consolas de videojuegos.

Usar instrucciones en lugar de restricciones

Las **instrucciones** y las **restricciones** se utilizan en las indicaciones para guiar el resultado de un LLM.

Una **instrucción** proporciona instrucciones explícitas sobre el formato, estilo o contenido deseado de la respuesta. Guía al modelo sobre lo que el modelo debe hacer o producir.

Una **restricción** es un conjunto de limitaciones o límites en la respuesta. Limita lo que el modelo no debe hacer o evitar.

Cada vez más investigaciones sugieren que centrarse en las instrucciones positivas al dar indicaciones puede ser más eficaz que depender en gran medida de las restricciones. Este enfoque se alinea con la forma en que los humanos prefieren las instrucciones positivas a las listas de lo que no se debe hacer.

Las instrucciones comunican directamente el resultado deseado, mientras que las restricciones pueden dejar al modelo adivinando qué está permitido. Proporciona flexibilidad y fomenta la creatividad dentro de los límites definidos, mientras que las restricciones pueden limitar el potencial del modelo. Además, una lista de restricciones puede entrar en conflicto entre sí.

Las restricciones siguen siendo valiosas, pero en ciertas situaciones. Para evitar que el modelo genere contenido dañino o sesgado o cuando se

necesita un formato o estilo de salida estricto.

Si es posible, utilice instrucciones positivas: en lugar de decirle al modelo qué no debe hacer, dígale qué debe hacer. Esto puede evitar confusiones y mejorar la precisión del resultado.

HACER:

Generar una entrada de blog de 1 párrafo sobre las 5 mejores consolas de videojuegos. Solo hable de la consola, la compañía que la fabricó, el año y las ventas totales.

NO HACER:

Genera una entrada de blog de 1 párrafo sobre las 5 mejores consolas de videojuegos. No incluyas los nombres de los videojuegos.

Como práctica recomendada, comienza priorizando las instrucciones, indicando claramente lo que quieras que haga el modelo y solo usa restricciones cuando sea necesario por seguridad, claridad o requisitos específicos. Experimenta e itera para probar diferentes combinaciones de instrucciones y restricciones para encontrar lo que funciona mejor para tus tareas específicas y documéntalas.

Controlar la longitud máxima del token

Para controlar la longitud de una respuesta LLM generada, puede establecer un límite máximo de token en la configuración o solicitar explícitamente una longitud específica en su solicitud. Por ejemplo:

"Explique la física cuántica en un mensaje con la longitud de un tuit."

Usar variables en las solicitudes

Para reutilizar las solicitudes y hacerlas más dinámicas, utilice variables en la solicitud, que se pueden cambiar para diferentes entradas. Por ejemplo,

como se muestra en la Tabla 20, una solicitud que proporciona datos sobre una ciudad. En lugar de codificar el nombre de la ciudad en la solicitud, utilice una variable. Las variables pueden ahorrarle tiempo y esfuerzo al permitirle evitar repetirse. Si necesita utilizar la misma información en varias solicitudes, puede almacenarla en una variable y luego hacer referencia a esa variable en cada solicitud. Esto tiene mucho sentido al integrar solicitudes en sus propias aplicaciones.

Solicitud	VARIABLES {ciudad} = "Ámsterdam" SOLICITUD Eres un guía de viajes. Cuéntame un dato sobre la ciudad: {ciudad}
Salida	Ámsterdam es una hermosa ciudad llena de canales, puentes y calles estrechas. Es un gran lugar para visitar por su rica historia, cultura y vida nocturna.

Tabla 20. Uso de variables en las indicaciones

Experimente con formatos de entrada y estilos de escritura

Diferentes modelos, configuraciones de modelos, formatos de indicaciones, elecciones de palabras y envíos pueden producir resultados diferentes. Por lo tanto, es importante experimentar con atributos de las indicaciones como el estilo, la elección de palabras y el tipo de indicación (**cero disparos**, **pocos disparos**, **indicación del sistema**).

Por ejemplo, una indicación con el objetivo de generar texto sobre la revolucionaria consola de videojuegos Sega Dreamcast puede formularse como una pregunta, una afirmación o una instrucción, lo que da como resultado diferentes resultados:

- **Pregunta:** ¿Qué era la Sega Dreamcast y por qué fue una consola tan revolucionaria?
- **Afirmación:** La Sega Dreamcast fue una consola de videojuegos de sexta generación lanzada por Sega en 1999.
- **Instrucciones:** Escriba un solo párrafo que describa la consola Sega Dreamcast y explique por qué fue tan revolucionaria.

For few-shot prompting with classification tasks, mix up the classes

Generally speaking, the order of your few-shots examples should not matter much. However, when doing classification tasks, make sure you mix up the possible response classes in the few shot examples. This is because you might otherwise be overfitting to the specific order of the examples. By mixing up the possible response classes, you can ensure that the model is learning to identify the key features of each class, rather than simply memorizing the order of the examples. This will lead to more robust and generalizable performance on unseen data.

Para indicaciones de pocos disparos con tareas de clasificación, mezcle las clases

En términos generales, el orden de los ejemplos de **pocos** ejemplos no debería importar mucho. Sin embargo, al realizar tareas de clasificación, asegúrese de mezclar las posibles clases de respuesta en los ejemplos de pocos ejemplos. Esto se debe a que, de lo contrario, podría sobreajustarse al orden específico de los ejemplos. Al mezclar las posibles clases de respuesta, puede asegurarse de que el modelo aprenda a identificar las características clave de cada clase, en lugar de simplemente memorizar el orden de los ejemplos. Esto conducirá a un rendimiento más robusto y generalizable con datos no vistos

Una buena regla general es comenzar con 6 ejemplos de disparos y comenzar a probar la precisión a partir de ahí.

Adaptarse a las actualizaciones del modelo

Es importante que se mantenga al tanto de los cambios en la arquitectura del modelo, los datos agregados y las capacidades. Pruebe versiones más nuevas del modelo y ajuste sus indicaciones para aprovechar mejor las nuevas características del modelo. Herramientas como Vertex AI Studio son excelentes para almacenar, probar y documentar las distintas versiones de su indicación.

Experimente con formatos de salida

Además del formato de entrada de la indicación, considere experimentar con el formato de salida. Para tareas no creativas como extraer, seleccionar, analizar, ordenar, clasificar o categorizar datos, intente que su salida se devuelva en un formato estructurado como JSON o XML.

Hay algunos beneficios en devolver objetos JSON desde una indicación que extrae datos. En una aplicación del mundo real, no necesito crear manualmente este formato JSON, ya puedo devolver los datos en un orden ordenado (muy útil cuando se trabaja con objetos de fecha y hora), pero lo más importante es que, al solicitar un formato JSON, se obliga al modelo a crear una estructura y se limitan las alucinaciones.

En resumen, las ventajas de usar **JSON** para la salida:

- Siempre devuelve el mismo estilo
- Céntrate en los datos que quieras recibir
- Menos posibilidades de alucinaciones
- Hazlo consciente de las relaciones
- Obtienes tipos de datos
- Puedes ordenarlos

La Tabla 4 en la sección de indicaciones de pocos intentos muestra un ejemplo sobre cómo devolver una salida estructurada.

Reparación de JSON

Si bien devolver datos en formato **JSON** ofrece numerosas ventajas, no está exento de inconvenientes. La naturaleza estructurada de JSON, si bien es beneficiosa para el análisis y el uso en aplicaciones, requiere muchos más tokens que el texto sin formato, lo que aumenta el tiempo de procesamiento y los costos. Además, la verbosidad de JSON puede consumir fácilmente toda la ventana de salida, lo que se vuelve especialmente problemático

cuando la generación se interrumpe abruptamente debido a los límites de tokens. Este truncamiento a menudo resulta en JSON no válido, sin llaves de cierre o corchetes cruciales, lo que hace que la salida sea inutilizable. Afortunadamente, herramientas como la [biblioteca json-repair](#) (disponible en PyPI) pueden ser invaluables en estas situaciones. Esta biblioteca intenta reparar de forma inteligente y automática los objetos JSON incompletos o malformados, lo que la convierte en un aliado crucial cuando se trabaja con JSON generado por LLM, especialmente cuando se trata de posibles problemas de truncamiento.

Trabajar con esquemas

Usar **JSON estructurado como salida** es una gran solución, como hemos visto varias veces en este documento. Pero, ¿qué pasa con la entrada? Si bien JSON es excelente para estructurar la salida que genera el LLM, también puede ser increíblemente útil para estructurar la entrada que proporcionas. Aquí es donde entran en juego los esquemas JSON. Un esquema JSON define la estructura esperada y los tipos de datos de tu entrada JSON. Al proporcionar un esquema, le das al LLM un plano claro de los datos que debe esperar, lo que le ayuda a centrar su atención en la información relevante y reduce el riesgo de malinterpretar la entrada. Además, los esquemas pueden ayudar a establecer relaciones entre diferentes datos e incluso hacer que el LLM sea "consciente del tiempo" al incluir campos de fecha o marca de tiempo con formatos específicos.

Aquí tienes un ejemplo sencillo:

Supongamos que quieres usar un LLM para generar descripciones de productos en un catálogo de comercio electrónico. En lugar de simplemente proporcionar una descripción de texto libre del producto, puedes usar un esquema **JSON** para definir los atributos del producto:

```
{  
    "type": "object",  
    "properties": {  
        "name": { "type": "string", "description": "Nombre del producto" }, "category": { "type": "string", "description": "Categoría del producto" }, "price": { "type": "number", "format": "float", "description": "Producto precio" },  
        "features": {  
            "type": "array",  
            "items": { "type": "string" },  
            "description": "Características clave del producto"  
        },  
        "release_date": { "type": "string", "format": "date", "description": "Fecha de lanzamiento del producto" }  
    },  
}
```

Fragmento 5. Definición del esquema de salida estructurado

Luego, puede proporcionar los datos reales del producto como un objeto JSON que se ajuste a este esquema:

```
{  
    "nombre": "Auriculares inalámbricos",  
    "categoría": "Electrónica",  
    "precio": 99.99,  
    "características": ["Cancelación de ruido", "Bluetooth 5.0", "Duración de la batería de 20 horas"],  
    "fecha_de_lanzamiento": "27/10/2023"  
}
```

Fragmento 6. Salida estructurada del LLM

Al preprocesar sus datos y, en lugar de proporcionar documentos completos que solo incluyan el esquema y los datos, le brinda al LLM una comprensión clara de los atributos del producto, incluida su fecha de lanzamiento, lo que hace mucho más probable que genere una descripción precisa y relevante. Este enfoque de entrada estructurada, que dirige la atención del LLM a los campos relevantes, es especialmente valioso cuando se trabaja con grandes volúmenes de datos o cuando se integran LLM en aplicaciones complejas.

Experimente junto con otros ingenieros de indicadores

Si te encuentras en una situación en la que tienes que intentar crear una buena consigna (**good prompt**), puede que quieras encontrar a varias personas para que lo intenten. Cuando todos sigan las mejores prácticas (como se enumeran en este capítulo), verás una variación en el rendimiento entre todos los diferentes intentos de consigna (**prompt**)

Mejores prácticas de CoT (CoT Best practices)

Para las indicaciones de CoT, es necesario colocar la respuesta después del razonamiento porque la generación del razonamiento cambia los tokens que el modelo obtiene cuando predice la respuesta final.

Con CoT y la autoconsistencia, necesitas poder extraer la respuesta final de tu indicación, separada del razonamiento.

Para las indicaciones de CoT, establece la temperatura en 0.

La indicación de cadena de pensamiento se basa en la decodificación voraz, que predice la siguiente palabra en una secuencia según la probabilidad más alta asignada por el modelo de lenguaje. En términos generales, cuando se usa el razonamiento, para llegar a la respuesta final, es probable que haya una sola respuesta correcta. Por lo tanto, la temperatura siempre debe establecerse en 0.

Documenta los distintos intentos de indicaciones

El último consejo se mencionó anteriormente en este capítulo, pero no podemos enfatizar lo suficiente lo importante que es: documenta tus intentos de indicaciones con todo detalle para que puedas aprender con el tiempo qué salió bien y qué no.

Los resultados de las indicaciones pueden diferir entre modelos, entre configuraciones de muestreo e incluso entre diferentes versiones del mismo modelo. Además, incluso entre indicaciones idénticas para el mismo modelo, pueden ocurrir pequeñas diferencias en el formato de las oraciones de salida y en la elección de palabras. (**Por ejemplo**, como se mencionó

anteriormente, si dos tokens tienen la misma probabilidad predicha, los empates pueden resolverse aleatoriamente. Esto puede afectar a los tokens predichos posteriores).

Recomendamos crear una Hoja de Cálculo de Google con la Tabla 21 como plantilla. Las ventajas de este enfoque son que tendrás un registro completo cuando inevitablemente tengas que revisar tu trabajo de indicaciones, ya sea para retomarlo en el futuro (te sorprendería lo mucho que puedes olvidar después de un breve descanso), para probar el rendimiento de las indicaciones en diferentes versiones de un modelo y para ayudar a depurar errores futuros.

Además de los campos de esta tabla, también es útil realizar un seguimiento de la versión de la indicación (iteración), un campo para registrar si el resultado fue correcto/incorrecto/a veces correcto, y un campo para registrar los comentarios. Si tienes la suerte de usar Vertex AI Studio, guarda tus indicaciones (usando el mismo nombre y versión que aparecen en tu documentación) y realiza un seguimiento del hipervínculo a la indicación guardada en la tabla. De esta manera, siempre estarás a un clic de volver a ejecutar tus indicaciones.

Al trabajar en un sistema de generación aumentada de recuperación, también debe capturar los aspectos específicos del sistema RAG que impactan el contenido que se insertó en la solicitud, incluyendo la consulta, la configuración de los fragmentos, la salida de los fragmentos y otra información.

Una vez que considere que la solicitud está cerca de la perfección, llévela a la base de código de su proyecto. Y en la base de código, guarde las solicitudes en un archivo separado del código, para que sea más fácil de mantener. Finalmente, idealmente, sus solicitudes forman parte de un sistema operacionalizado, y como ingeniero de solicitudes debe confiar en pruebas automatizadas y procedimientos de evaluación para comprender qué tan bien se generaliza su solicitud a una tarea.

La ingeniería de solicitudes es un proceso iterativo. Elabore y pruebe diferentes solicitudes, analice y documente los resultados. Refine su solicitud en función del rendimiento del modelo. Siga experimentando hasta lograr el resultado deseado. Cuando cambie un modelo o la configuración del modelo, vuelva atrás y siga experimentando con las solicitudes utilizadas anteriormente

Name	[name and version of your prompt]		
Goal	[One sentence explanation of the goal of this attempt]		
Model	[name and version of the used model]		
Temperature	[value between 0 - 1]	Token Limit	[number]
Top-K	[number]	Top-P	[number]
Prompt	[Write all the full prompt]		
Output	[Write out the output or multiple outputs]		

Tabla 21. Una plantilla para documentar indicaciones

Summary

This whitepaper discusses prompt engineering. We learned various prompting techniques, such as:

- Zero prompting
- Few shot prompting
- System prompting
- Role prompting
- Contextual prompting
- Step-back prompting
- Chain of thought
- Self consistency
- Tree of thoughts

- ReAct

We even looked into ways how you can automate your prompts.

The whitepaper then discusses the challenges of gen AI like the problems that can happen when your prompts are insufficient. We closed with best practices on how to become a better prompt engineer.

Resumen

Este documento técnico analiza la ingeniería de indicaciones. Aprendimos varias técnicas de indicaciones, como:

- Indicación cero
- Indicación de pocos intentos
- Indicación del sistema
- Indicación de roles
- Indicación contextual
- Indicación de retroceso
- Cadena de pensamiento
- Autoconsistencia
- Árbol de pensamientos
- React

Incluso analizamos maneras de automatizar tus indicaciones.

El informe técnico luego analiza los desafíos de la IA general, como los problemas que pueden surgir cuando tus indicaciones son insuficientes. Cerramos con las mejores prácticas sobre cómo convertirse en un mejor ingeniero de indicaciones

PDF → Prompt Engineering

Resumen de Gemini

Este documento, "Ingeniería de Prompts", escrito por Lee Boonstra de Google, explora las complejidades y técnicas de la creación de prompts efectivos para modelos de lenguaje grandes (LLM), especialmente el modelo Gemini dentro de Vertex AI o a través de su API.

El documento comienza explicando qué es la ingeniería de prompts, enfatizando que es un proceso iterativo que implica optimizar la longitud, el estilo, el tono, la estructura y el contexto del prompt para guiar a los LLM a producir resultados precisos. También aborda la configuración de salida del LLM, como la longitud de la salida, el control de muestreo (temperatura, Top-K y Top-P) y cómo estos ajustes impactan la aleatoriedad y diversidad del texto generado.

Se detallan varias técnicas de prompting:

- **Zero-shot prompting:** El tipo más simple, que proporciona solo una descripción de la tarea sin ejemplos.
- **One-shot y Few-shot prompting:** Incluye uno o varios ejemplos, respectivamente, para ayudar al modelo a comprender la estructura o el patrón de salida deseado.
- **Prompting de sistema, contextual y de rol:**
 - **Sistema:** Establece el contexto general y el propósito del modelo.
 - **Contextual:** Proporciona detalles específicos o información de fondo relevante para la tarea actual.
 - **De rol:** Asigna una identidad o personaje específico al modelo para que adopte un tono y estilo de voz consistentes.
- **Step-back prompting:** Anima al LLM a considerar una pregunta general relacionada con la tarea específica, utilizando la respuesta para activar el conocimiento y los procesos de razonamiento relevantes.
- **Chain of Thought (CoT):** Mejora las capacidades de razonamiento de los LLM generando pasos de razonamiento intermedios.
- **Self-consistency:** Combina el muestreo y la votación mayoritaria para generar diversas rutas de razonamiento y seleccionar la respuesta más consistente.

- **Tree of Thoughts (ToT):** Una generalización de CoT que permite a los LLM explorar múltiples rutas de razonamiento simultáneamente.
- **ReAct (reason & act):** Permite a los LLM resolver tareas complejas combinando el razonamiento en lenguaje natural con herramientas externas (como búsqueda o intérpretes de código).
- **Ingeniería de Prompts Automática (APE):** Un método para que un modelo genere y evalúe prompts para mejorar su rendimiento.

El documento también cubre la ingeniería de prompts para código, incluyendo la generación, explicación, traducción, depuración y revisión de código. Finalmente, se presentan las mejores prácticas de la ingeniería de prompts, como proporcionar ejemplos, diseñar con simplicidad, ser específico sobre la salida, usar instrucciones en lugar de restricciones, controlar la longitud máxima de tokens, usar variables en los prompts, experimentar con formatos de entrada y estilos de escritura, y documentar los intentos de prompts. También se menciona la reparación de JSON y el trabajo con esquemas para una salida e entrada estructuradas.