

Patterns of Application Development Using AI

Obie Fernandez

Foreword by Gregor Hohpe



Edición en Español

Patrones de Desarrollo de Aplicaciones

Usando IA (Edición en Español)

Obie Fernandez

Este libro está a la venta en

<http://leanpub.com/patterns-of-application-development-using-ai-es>

Esta versión se publicó en 2025-01-23



Éste es un libro de [Leanpub](#). Leanpub anima a los autores y publicadoras con el proceso de publicación. [Lean Publishing](#) es el acto de publicar un libro en progreso usando herramientas sencillas y muchas iteraciones para obtener retroalimentación del lector hasta conseguir el libro adecuado.

© 2025 Obie Fernandez

¡Tuitea sobre el libro!

Por favor ayuda a Obie Fernandez hablando sobre el libro en [Twitter](#)!

El hashtag sugerido para este libro es [#poaduai](#).

Descubre lo que otra gente dice sobre el libro haciendo clic en este enlace para buscar el hashtag en Twitter:

[#poaduai](#)

Para mi increíble reina, mi musa, mi luz y amor, Victoria

También por Obie Fernandez

Patterns of Application Development Using AI

The Rails 8 Way

The Rails 7 Way

XML The Rails Way

Serverless

El Libro Principiante de Node

The Lean Enterprise

Índice general

Prólogo por Gregor Hohpe	i
Prefacio	ii
Acerca del Libro	iii
Acerca de los Ejemplos de Código	iii
Lo Que No Cubro	iii
Para Quién Es Este Libro	iii
Construyendo un Vocabulario Común	iii
Participación	iii
Agradecimientos	iii
¿Qué hay con las ilustraciones?	iv
Sobre Lean Publishing	iv
Sobre el Autor	v
Introducción	1
Reflexiones sobre la Arquitectura de Software	2
¿Qué es un Modelo de Lenguaje Grande?	3
Entendiendo la Inferencia	5
Pensando en el Rendimiento	28
Experimentando con Diferentes Modelos LLM	30
Sistemas de IA Compuestos	30

Parte 1: Enfoques y Técnicas Fundamentales 39

Estrechar el Camino	40
Espacio Latente: Incomprensiblemente Vasto	42
Cómo Se “Estrecha” El Camino	46
Modelos Sin Procesar Versus Modelos Ajustados por Instrucciones	50
Ingeniería de Prompts	57
Destilación de Prompts	73
¿Qué hay del ajuste fino?	80
Retrieval Augmented Generation (RAG)	82
¿Qué es la Generación Aumentada por Recuperación?	82
¿Cómo funciona RAG?	82
¿Por qué usar RAG en sus aplicaciones?	82
Implementando RAG en Tu Aplicación	82
Fragmentación por Proposiciones	83
Ejemplos del Mundo Real de RAG	83
Optimización Inteligente de Consultas (IQO)	84
Reordenamiento	84
Evaluación RAG (RAGAs)	84
Desafíos y Perspectivas Futuras	86
Multitud de Trabajadores	88
Trabajadores de IA Como Componentes Reutilizables Independientes	89
Gestión de Cuentas	91
Aplicaciones de Comercio Electrónico	92
Aplicaciones en el Sector Salud	101
El Trabajador de IA como Gestor de Procesos	105
Integrando Trabajadores de IA en su Arquitectura de Aplicación	108
Composición y Orquestación de Workers de IA	112

ÍNDICE GENERAL

Combinando PLN Tradicional con MLGs	121
Uso de Herramientas	125
¿Qué es el Uso de Herramientas?	125
El Potencial del Uso de Herramientas	127
El Flujo de Trabajo del Uso de Herramientas	128
Mejores Prácticas para el Uso de Herramientas	143
Composición y Encadenamiento de Herramientas	148
Direcciones Futuras	150
Procesamiento de Flujo	152
Implementando un ReplyStream	153
El “Bucle de Conversación”	159
Continuación Automática	162
Conclusión	164
Datos Autorreparables	166
Caso Práctico: Arreglando JSON Dañado	168
Consideraciones y Contraindicaciones	173
Generación de Contenido Contextual	189
Personalización	190
Productividad	192
Iteración Rápida y Experimentación	194
Localización Impulsada por IA	197
La Importancia de las Pruebas de Usuario y la Retroalimentación	199
UI Generativa	200
Generando Texto para Interfaces de Usuario	201
Definiendo la UI Generativa	211
Ejemplo	213

ÍNDICE GENERAL

El Cambio hacia el Diseño Orientado a Resultados	216
Desafíos y Consideraciones	217
Perspectivas Futuras y Oportunidades	219
Orquestación Inteligente de Flujos de Trabajo	223
Necesidad Empresarial	224
Beneficios Clave	225
Patrones Clave	225
Manejo y Recuperación de Excepciones	228
Implementación Práctica de la Orquestación de Flujos de Trabajo Inteligentes	231
Monitoreo y Registro	246
Consideraciones de Escalabilidad y Rendimiento	251
Pruebas y validación de flujos de trabajo	256
Parte 2: Los Patrones	264
Ingeniería de Prompts	265
Cadena de Pensamiento	266
Mode Switch	267
Asignación de Rol	268
Prompt Object	269
Prompt Template	270
Structured IO	271
Encadenamiento de Prompts	272
Reescritor de Prompts	273
Response Fencing	274
Analizador de Consultas	275
Reescritor de Consultas	277
Ventriloquist	278

ÍNDICE GENERAL

Componentes Discretos	279
Predicado	280
Fachada de API	281
Intérprete de Resultados	283
Máquina Virtual	284
Especificación y Pruebas	284
Human In The Loop (HITL)	286
Patrones de Alto Nivel	286
Escalamiento	287
Bucle de Retroalimentación	288
Radiación Pasiva de Información	289
Toma de Decisiones Colaborativa (CDM)	291
Aprendizaje Continuo	292
Consideraciones Éticas	292
Avances Tecnológicos y Perspectivas Futuras	292
Manejo Inteligente de Errores	294
Enfoques Tradicionales de Manejo de Errores	294
Diagnóstico Contextual de Errores	295
Informes Inteligentes de Errores	296
Prevención Predictiva de Errores	297
Recuperación Inteligente de Errores	297
Comunicación Personalizada de Errores	298
Flujo de Trabajo Adaptativo de Manejo de Errores	299
Control de Calidad	300
Eval	301
Mecanismo de Protección	303
Guardarraíles y Evaluaciones: Dos Caras de la Misma Moneda	303

Glosario	305
Glosario	305
Index	310

Prólogo por Gregor Hohpe

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Prefacio

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Acerca del Libro

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Acerca de los Ejemplos de Código

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Lo Que No Cubro

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Para Quién Es Este Libro

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Construyendo un Vocabulario Común

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Participación

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Agradecimientos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

¿Qué hay con las ilustraciones?

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

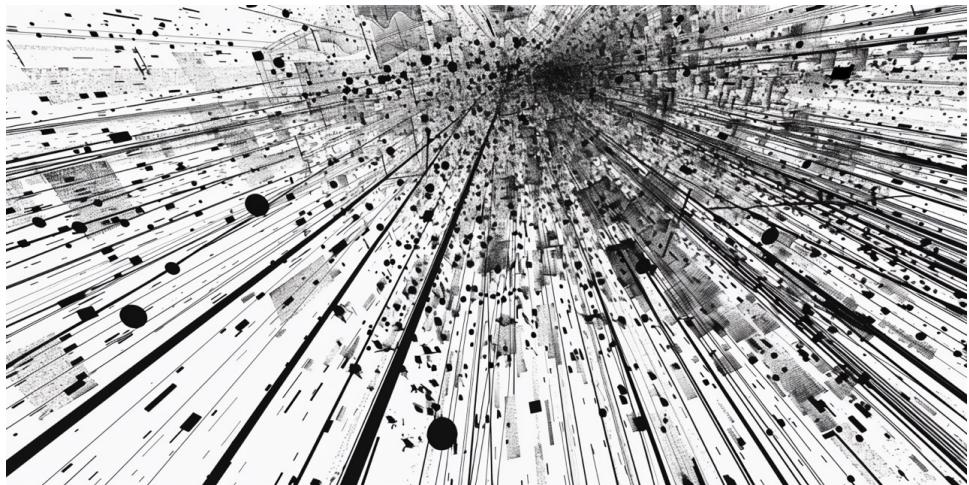
Sobre Lean Publishing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Sobre el Autor

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Introducción



Si estás ansioso por comenzar a integrar Modelos de Lenguaje Grande (LLMs) de IA en tus proyectos de programación, siéntete libre de sumergirte directamente en los patrones y ejemplos de código presentados en los capítulos posteriores. Sin embargo, para apreciar completamente el poder y el potencial de estos patrones, vale la pena tomarse un momento para entender el contexto más amplio y el enfoque cohesivo que representan.

Los patrones no son simplemente una colección de técnicas aisladas, sino más bien un marco unificado para integrar la IA en tus aplicaciones. Yo uso Ruby on Rails, pero estos patrones deberían funcionar en prácticamente cualquier otro entorno de programación. Abordan una amplia gama de preocupaciones, desde la gestión de datos y la optimización del rendimiento hasta la experiencia del usuario y la seguridad, proporcionando un conjunto completo de herramientas para mejorar las prácticas de programación tradicionales con las capacidades de la IA.

Cada categoría de patrones aborda un desafío u oportunidad específica que surge al incorporar componentes de IA en tu aplicación. Al comprender las relaciones y sinergias entre estos patrones, puedes tomar decisiones informadas sobre dónde y cómo aplicar la IA de manera más efectiva.

Los patrones nunca son soluciones prescriptivas y no deben ser tratados como tales. Están destinados a ser bloques de construcción adaptables que deben ajustarse a los requisitos y restricciones únicos de tu propia aplicación. La aplicación exitosa de estos patrones (como cualquier otro en el campo del software) se basa en una comprensión profunda del dominio del problema, las necesidades del usuario y la arquitectura técnica general de tu proyecto.

Reflexiones sobre la Arquitectura de Software

Comencé a programar en los años 80 y estuve involucrado en la escena hacker, y nunca perdí mi mentalidad hacker, incluso después de convertirme en un desarrollador de software profesional. Desde el principio, siempre tuve un sano escepticismo sobre el valor real que aportaban los arquitectos de software desde sus torres de marfil.

Una de las razones por las que personalmente estoy tan entusiasmado con los cambios provocados por esta poderosa nueva ola de tecnología de IA es su impacto en lo que consideramos decisiones de *arquitectura de software*. Desafía las nociones tradicionales de lo que constituye la manera “correcta” de diseñar e implementar nuestros proyectos de software. También cuestiona si la arquitectura todavía puede considerarse principalmente como *las partes de un sistema que son difíciles de cambiar*, ya que la mejora de la IA está haciendo que sea más fácil que nunca cambiar cualquier parte de tu proyecto, en cualquier momento.

Quizás estemos entrando en los años cumbre del enfoque “posmoderno” de la ingeniería de software. En este contexto, posmoderno se refiere a un cambio fundamental alejado de los paradigmas tradicionales, donde los desarrolladores eran responsables de escribir

y mantener cada línea de código. En su lugar, abraza la idea de delegar tareas, como la manipulación de datos, algoritmos complejos e incluso partes enteras de la lógica de la aplicación, a bibliotecas de terceros y APIs externas. Este cambio posmoderno representa una desviación significativa de la sabiduría convencional de construir aplicaciones desde cero, y desafía a los desarrolladores a repensar su papel en el proceso de desarrollo.

Siempre he creído que los buenos programadores solo escriben el código que es absolutamente necesario escribir, basándome en las enseñanzas de Larry Wall y otros luminarios hacker como él. Al minimizar la cantidad de código escrito, podemos movernos más rápido, reducir la superficie para errores, simplificar el mantenimiento y mejorar la fiabilidad general de sus aplicaciones. Menos código nos permite centrarnos en la lógica de negocio central y la experiencia del usuario, mientras delegamos otro trabajo a otros servicios.

Ahora que los sistemas impulsados por IA pueden manejar tareas que anteriormente eran dominio exclusivo del código escrito por humanos, deberíamos poder ser aún más productivos y ágiles, con un enfoque mayor que nunca en crear valor comercial y experiencia de usuario.

Por supuesto, hay compensaciones al delegar grandes partes de tu proyecto a sistemas de IA, como la posible pérdida de control y la necesidad de mecanismos robustos de monitoreo y retroalimentación. Por eso requiere un nuevo conjunto de habilidades y conocimientos, incluida al menos una comprensión fundamental de cómo funciona la IA.

¿Qué es un Modelo de Lenguaje Grande?

Los Modelos de Lenguaje Grande (LLMs) son un tipo de modelo de inteligencia artificial que ha ganado una atención significativa en los últimos años, desde el lanzamiento de GPT-3 por OpenAI en 2020. Los LLMs están diseñados para procesar, comprender y generar lenguaje humano con notable precisión y fluidez. En esta sección, daremos

un breve vistazo a cómo funcionan los LLMs y por qué son adecuados para construir componentes de sistemas inteligentes.

En su núcleo, los LLMs se basan en algoritmos de aprendizaje profundo, específicamente redes neuronales. Estas redes están compuestas por nodos interconectados, o neuronas, que procesan y transmiten información. La arquitectura de elección para los LLMs es a menudo el modelo Transformer, que ha demostrado ser altamente efectivo en el manejo de datos secuenciales como texto.

Los modelos transformer se basan en el mecanismo de atención y se utilizan principalmente para tareas que involucran datos secuenciales, como el procesamiento del lenguaje natural. Los transformers procesan los datos de entrada todos a la vez en lugar de secuencialmente, lo que les permite capturar dependencias de largo alcance de manera más efectiva. Tienen capas de mecanismos de atención que ayudan al modelo a enfocarse en diferentes partes de los datos de entrada para comprender el contexto y las relaciones.

El proceso de entrenamiento de los LLMs implica exponer el modelo a grandes cantidades de datos textuales, como libros, artículos, sitios web y repositorios de código. Durante el entrenamiento, el modelo aprende a reconocer patrones, relaciones y estructuras dentro del texto. Captura las propiedades estadísticas del lenguaje, como las reglas gramaticales, las asociaciones de palabras y los significados contextuales.

Una de las técnicas clave utilizadas en el entrenamiento de LLMs es el aprendizaje no supervisado. Esto significa que el modelo aprende de los datos sin etiquetado o guía explícita. Descubre patrones y representaciones por sí mismo al analizar la co-ocurrencia de palabras y frases en los datos de entrenamiento. Esto permite que los LLMs desarrollen una comprensión profunda del lenguaje y sus complejidades.

Otro aspecto importante de los LLMs es su capacidad para manejar el *contexto*. Al procesar un texto, los LLMs consideran no solo las palabras individuales sino también el contexto circundante. Tienen en cuenta las palabras, oraciones e incluso párrafos anteriores para comprender el significado y la intención del texto. Esta comprensión

contextual permite a los LLMs generar respuestas coherentes y relevantes. Una de las principales formas en que evaluamos las capacidades de un modelo LLM específico es considerando el tamaño del contexto que pueden tener en cuenta para generar respuestas.

Una vez entrenados, los LLMs pueden utilizarse para una amplia gama de tareas relacionadas con el lenguaje. Pueden generar texto similar al humano, responder preguntas, resumir documentos, traducir idiomas e incluso escribir código. La versatilidad de los LLMs los hace valiosos para construir componentes de sistemas inteligentes que pueden interactuar con usuarios, procesar y analizar datos textuales, y generar resultados significativos.

Al incorporar LLMs en la arquitectura de la aplicación, puedes crear componentes de IA que entiendan y procesen la entrada del usuario, generen contenido dinámico y proporcionen recomendaciones o acciones inteligentes. Pero trabajar con LLMs requiere una consideración cuidadosa de los requisitos de recursos y las compensaciones de rendimiento. Los LLMs requieren un uso intensivo de recursos computacionales y pueden necesitar una cantidad significativa de potencia de procesamiento y memoria (en otras palabras, dinero) para operar. La mayoría de nosotros necesitaremos evaluar las implicaciones de costos de integrar LLMs en nuestras aplicaciones y actuar en consecuencia.

Entendiendo la Inferencia

La inferencia se refiere al proceso mediante el cual un modelo genera predicciones o salidas basadas en datos nuevos y no vistos. Es la fase donde el modelo entrenado se utiliza para tomar decisiones o generar texto, imágenes u otro contenido en respuesta a las entradas del usuario.

Durante la fase de entrenamiento, un modelo de IA aprende de un gran conjunto de datos ajustando sus parámetros para minimizar el error en sus predicciones. Una vez

entrenado, el modelo puede aplicar lo que ha aprendido a nuevos datos. La inferencia es cómo el modelo utiliza sus patrones y conocimientos aprendidos para generar salidas.

Para los LLMs, la inferencia implica tomar un prompt o texto de entrada y producir una respuesta coherente y contextualmente relevante, como un flujo de *tokens* (de los que hablaremos pronto). Esto podría ser responder una pregunta, completar una oración, generar una historia o traducir texto, entre muchas otras tareas.



En contraste con la forma en que tú y yo pensamos, el “pensamiento” de un modelo de IA a través de la inferencia ocurre todo en una única operación sin estado. Es decir, su pensamiento está limitado a su proceso de generación. Literalmente tiene que pensar en voz alta, como si te hiciera una pregunta y solo aceptara una respuesta tuya en estilo “flujo de conciencia”.

Los Modelos de Lenguaje Grandes Vienen en Muchos Tamaños y Sabores

Si bien prácticamente todos los modelos de lenguaje grandes (LLMs) populares se basan en la misma arquitectura transformer básica y están entrenados en enormes conjuntos de datos de texto, vienen en una variedad de tamaños y están ajustados para diferentes propósitos. El tamaño de un LLM, medido por el número de parámetros en su red neuronal, tiene un gran impacto en sus capacidades. Los modelos más grandes con más parámetros, como GPT-4, que se rumorea que cuenta con 1 a 2 billones de parámetros, son generalmente más conocedores y capaces que los modelos más pequeños. Sin embargo, los modelos más grandes también requieren mucha más potencia de cómputo para ejecutarse, lo que se traduce en un mayor gasto cuando los utilizas a través de llamadas API.

Para hacer que los LLMs sean más prácticos y adaptados a casos de uso específicos, los modelos base a menudo se ajustan finamente con conjuntos de datos más específicos. Por ejemplo, un LLM puede ser entrenado en un gran corpus de diálogo para especializarlo

en IA conversacional. Otros son [entrenados en código](#) para dotarlos de conocimientos de programación. ¡Incluso hay modelos que están [especialmente entrenados para interacciones de juego de roles con usuarios!](#)

Modelos de Recuperación vs Modelos Generativos

En el mundo de los modelos de lenguaje grandes (LLMs), existen dos enfoques principales para generar respuestas: los modelos basados en recuperación y los modelos generativos. Cada enfoque tiene sus propias fortalezas y debilidades, y comprender las diferencias entre ellos puede ayudarte a elegir el modelo adecuado para tu caso de uso específico.

Modelos Basados en Recuperación

Los modelos basados en recuperación, también conocidos como modelos de recuperación de información, generan respuestas mediante la búsqueda en una gran base de datos de texto preexistente y la selección de los pasajes más relevantes según la consulta de entrada. Estos modelos no generan texto nuevo desde cero, sino que unen extractos de la base de datos para formar una respuesta coherente.

Una de las principales ventajas de los modelos basados en recuperación es su capacidad para proporcionar información precisa y actualizada. Dado que dependen de una base de datos de texto curado, pueden extraer información relevante de fuentes confiables y presentarla al usuario. Esto los hace especialmente adecuados para aplicaciones que requieren respuestas precisas y factuales, como los sistemas de pregunta-respuesta o bases de conocimiento.

Sin embargo, los modelos basados en recuperación tienen algunas limitaciones. Son tan buenos como la base de datos que están consultando, por lo que la calidad y cobertura de la base de datos impactan directamente en el rendimiento del modelo. Además, estos modelos pueden tener dificultades para generar respuestas coherentes y naturales, ya que están limitados al texto disponible en la base de datos.

No cubrimos el uso de modelos de recuperación puros en este libro.

Modelos Generativos

Los modelos generativos, por otro lado, crean texto nuevo desde cero basándose en los patrones y relaciones que aprendieron durante el entrenamiento. Estos modelos utilizan su comprensión del lenguaje para generar respuestas novedosas que se adaptan al prompt de entrada.

La principal fortaleza de los modelos generativos es su capacidad para producir texto creativo, coherente y contextualmente relevante. Pueden participar en conversaciones abiertas, generar historias e incluso escribir código. Esto los hace ideales para aplicaciones que requieren interacciones más abiertas y dinámicas, como chatbots, creación de contenido y asistentes de escritura creativa.

Sin embargo, los modelos generativos pueden producir ocasionalmente información inconsistente o factualmente incorrecta, ya que se basan en los patrones aprendidos durante el entrenamiento en lugar de una base de datos curada de hechos. También pueden ser más propensos a sesgos y alucinaciones, generando texto que es plausible pero no necesariamente verdadero.

Ejemplos de LLMs generativos incluyen la serie GPT de OpenAI (GPT-3, GPT-4) y Claude de Anthropic.

Modelos Híbridos

Varios LLMs disponibles comercialmente combinan tanto enfoques de recuperación como generativos en un modelo híbrido. Estos modelos utilizan técnicas de recuperación para encontrar información relevante de una base de datos y luego utilizan técnicas generativas para sintetizar esa información en una respuesta coherente.

Los modelos híbridos buscan combinar la precisión factual de los modelos basados en recuperación con las capacidades de generación de lenguaje natural de los modelos

generativos. Pueden proporcionar información más confiable y actualizada mientras mantienen la capacidad de participar en conversaciones abiertas.

Al elegir entre modelos basados en recuperación y modelos generativos, debes considerar los requisitos específicos de tu aplicación. Si el objetivo principal es proporcionar información precisa y factual, un modelo basado en recuperación puede ser la mejor opción. Si la aplicación requiere interacciones más abiertas y creativas, un modelo generativo puede ser más adecuado. Los modelos híbridos ofrecen un equilibrio entre los dos enfoques y pueden ser una buena opción para aplicaciones que requieren tanto precisión factual como generación de lenguaje natural.

En última instancia, la elección entre modelos basados en recuperación y modelos generativos depende del caso de uso específico y los compromisos entre precisión, creatividad y flexibilidad. Al comprender las fortalezas y limitaciones de cada enfoque, puedes tomar decisiones informadas.

Ajuste por Instrucciones

El ajuste por instrucciones es un subconjunto del entrenamiento conversacional que utiliza prompts y respuestas escritas por humanos para hacer que los LLMs sean mejores en la comprensión y seguimiento de instrucciones. Aquí hay algunos ejemplos simples para ilustrar cómo el ajuste por instrucciones en prompts y respuestas escritas por humanos hace que los LLMs sean mejores en la comprensión y seguimiento de instrucciones:

Ejemplo 1: Atención al Cliente

Instrucción: “Proporciona una respuesta a un cliente que solicita un reembolso.”

Prompt Escrito por Humano: “Me gustaría solicitar un reembolso por mi compra reciente. El producto no cumplió con mis expectativas.”

Respuesta Escrita por Humano: “Lamento escuchar que el producto no cumplió con sus expectativas. Para procesar su reembolso, por favor proporcione su número de pedido y el motivo de la devolución. Revisaremos su solicitud y le responderemos en 2-3 días hábiles.”

Ejemplo 2: Asistencia Educativa

Instrucción: “Explica el concepto de fotosíntesis a un niño de 10 años.”

Prompt escrito por humano: “¿Puedes explicar cómo las plantas producen su alimento?”

Respuesta escrita por humano: “¡Claro! Las plantas producen su alimento mediante un proceso llamado fotosíntesis. Utilizan la luz solar, el agua y el dióxido de carbono para crear alimento y oxígeno. Las hojas de la planta tienen una parte especial llamada

clorofila que les ayuda a captar la luz solar.”

Ejemplo 3: Asistente Personal

Instrucción: “Programa una reunión para el próximo martes a las 3 PM.”

Prompt escrito por humano: “¿Puedes programar una reunión para el martes a las 3 PM?”

Respuesta escrita por humano: “Claro, he programado una reunión para el próximo martes a las 3 PM. ¿Necesitas algo más?”

El resultado es un ecosistema diverso de LLMs de diferentes tamaños y con diferentes especialidades. Los modelos más pequeños en el rango de 1-7 mil millones de parámetros proporcionan buenas capacidades generales de lenguaje mientras son más eficientes de ejecutar.

- Mistral 7B
- Llama 3 8B
- Gemma 7B

Los modelos de tamaño medio de alrededor de 30-70 mil millones de parámetros ofrecen capacidades más sólidas de razonamiento y seguimiento de instrucciones.

- Llama 3 70B
- Qwen2 70B
- Mixtral 8x22B

Al elegir un LLM para incorporarlo en una aplicación, debes equilibrar las capacidades del modelo con factores prácticos como el costo, la latencia, la longitud de contexto y el filtrado de contenido. Los modelos más pequeños, ajustados por instrucciones,

son a menudo la mejor opción para tareas de lenguaje más simples, mientras que los modelos más grandes pueden ser necesarios para razonamiento o análisis complejos. Los datos de entrenamiento del modelo también son una consideración importante, ya que determinan la fecha límite de conocimiento del modelo.



Ciertos modelos, como algunos de Perplexity, están conectados a fuentes de información en tiempo real, por lo que efectivamente no tienen fecha límite. Cuando les haces preguntas, pueden decidir independientemente realizar búsquedas web y obtener páginas web arbitrarias para generar una respuesta.

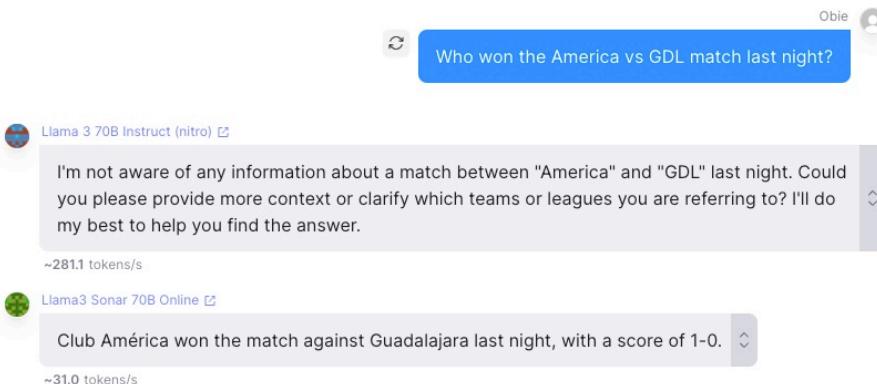


Figura 1. Llama3 con y sin acceso en línea

En última instancia, no existe un LLM que sirva para todo. Comprender las variaciones en el tamaño del modelo, la arquitectura y el entrenamiento es clave para seleccionar el modelo adecuado para un caso de uso específico. Experimentar con diferentes modelos es la única manera práctica de revelar cuáles proporcionan el mejor rendimiento para la tarea en cuestión.

Tokenización: Dividiendo el Texto en Piezas

Antes de que un modelo de lenguaje grande pueda procesar texto, ese texto necesita ser dividido en unidades más pequeñas llamadas *tokens*. Los tokens pueden ser palabras individuales, partes de palabras o incluso caracteres individuales. El proceso de dividir texto en tokens se conoce como tokenización, y es un paso crucial en la preparación de datos para un modelo de lenguaje.

The process of splitting text into tokens is known as tokenization, and it's a crucial step in preparing data for a language model.

Figura 2. Esta oración contiene 27 tokens

Diferentes LLMs utilizan diferentes estrategias de tokenización, lo que puede tener un impacto significativo en el rendimiento y las capacidades del modelo. Algunos tokenizadores comunes utilizados por LLMs incluyen:

- **GPT (Codificación por Pares de Bytes):** Los tokenizadores GPT utilizan una técnica llamada codificación por pares de bytes (BPE) para dividir el texto en unidades de subpalabras. BPE fusiona iterativamente los pares de bytes más frecuentes en un corpus de texto, formando un vocabulario de tokens de subpalabras. Esto permite que el tokenizador maneje palabras raras y nuevas dividiéndolas en piezas de subpalabras más comunes. Los tokenizadores GPT son utilizados por modelos como GPT-3 y GPT-4.
- **Llama (SentencePiece):** Los tokenizadores de Llama utilizan la biblioteca SentencePiece, que es un tokenizador y detokenizador de texto no supervisado. SentencePiece trata el texto de entrada como una secuencia de caracteres Unicode y aprende un vocabulario de subpalabras basado en un corpus de entrenamiento. Puede manejar cualquier idioma que pueda codificarse en Unicode, lo que lo hace adecuado para modelos multilingües. Los tokenizadores de Llama son utilizados por modelos como Llama y Alpaca de Meta.

- **SentencePiece (Unígrafo):** Los tokenizadores SentencePiece también pueden utilizar un algoritmo diferente llamado Unígrafo, que se basa en una técnica de regularización de subpalabras. La tokenización por Unígrafo determina el vocabulario óptimo de subpalabras basándose en un modelo de lenguaje unígrafo, que asigna probabilidades a unidades individuales de subpalabras. Este enfoque puede producir subpalabras más significativas semánticamente en comparación con BPE. SentencePiece con Unígrafo es utilizado por modelos como T5 y BERT de Google.
- **Google Gemini (Tokenización Multimodal):** Google Gemini utiliza un esquema de tokenización diseñado para manejar varios tipos de datos, incluyendo texto, imágenes, audio, videos y código. Esta capacidad multimodal permite a Gemini procesar e integrar diferentes formas de información. Notablemente, Google Gemini 1.5 Pro tiene una ventana de contexto que puede manejar millones de tokens, mucho más grande que los modelos anteriores. Esta extensa ventana de contexto permite al modelo procesar un contexto más amplio, potencialmente conduciendo a respuestas más precisas. Sin embargo, es importante señalar que el esquema de tokenización de Gemini está mucho más cerca de un token por carácter que otros modelos. Esto significa que el costo real de usar modelos Gemini puede ser significativamente más alto de lo esperado si estás acostumbrado a usar modelos como GPT, ya que el precio de Google se basa en caracteres en lugar de tokens.

La elección del tokenizador afecta varios aspectos de un LLM, incluyendo:

- **Tamaño del vocabulario:** El tokenizador determina el tamaño del vocabulario del modelo, que es el conjunto de tokens únicos que reconoce. Un vocabulario más grande y detallado puede ayudar al modelo a manejar una gama más amplia de palabras y frases e incluso volverse multimodal (capaz de entender y generar más que solo texto), pero también aumenta los requisitos de memoria y la complejidad computacional del modelo.

- **Manejo de palabras raras y desconocidas:** Los tokenizadores que utilizan unidades de subpalabras, como BPE y SentencePiece, pueden descomponer palabras raras y desconocidas en piezas de subpalabras más comunes. Esto permite que el modelo haga suposiciones educadas sobre el significado de palabras que no ha visto antes, basándose en las subpalabras que contienen.
- **Soporte multilingüe:** Los tokenizadores como SentencePiece, que pueden manejar cualquier lenguaje codificable en Unicode, son adecuados para modelos multilingües que necesitan procesar texto en múltiples idiomas.

Al elegir un LLM para una aplicación particular, es importante considerar el tokenizador que utiliza y qué tan bien se alinea con las necesidades específicas de procesamiento del lenguaje de la tarea en cuestión. El tokenizador puede tener un impacto significativo en la capacidad del modelo para manejar terminología específica del dominio, palabras raras y texto multilingüe.

Tamaño de Contexto: ¿Cuánta Información Puede Usar un Modelo de Lenguaje Durante la Inferencia?

Al hablar de modelos de lenguaje, el tamaño de contexto se refiere a la cantidad de texto que un modelo puede considerar al procesar o generar sus respuestas. Es esencialmente una medida de cuánta información el modelo puede “recordar” y usar para informar sus salidas (expresada en tokens). El tamaño de contexto de un modelo de lenguaje puede tener un impacto significativo en sus capacidades y los tipos de tareas que puede realizar efectivamente.

¿Qué es el Tamaño de Contexto?

En términos técnicos, el tamaño de contexto está determinado por el número de tokens (palabras o piezas de palabras) que un modelo de lenguaje puede procesar en una sola secuencia de entrada. Esto a menudo se conoce como la “capacidad de atención” o

“ventana de contexto” del modelo. Cuanto mayor sea el tamaño de contexto, más texto puede considerar el modelo a la vez al generar una respuesta o realizar una tarea.

Diferentes modelos de lenguaje tienen diversos tamaños de contexto, que van desde unos cientos de tokens hasta millones de tokens. Como referencia, un párrafo típico de texto podría contener alrededor de 100-150 tokens, mientras que un libro completo podría contener decenas o cientos de miles de tokens.

Incluso hay trabajo en métodos eficientes para escalar Modelos de Lenguaje Grandes (LLMs) basados en Transformers para [entradas infinitamente largas](#) con memoria y computación limitadas.

¿Por qué es importante el tamaño de contexto?

El tamaño de contexto de un modelo de lenguaje tiene un impacto significativo en su capacidad para comprender y generar texto coherente y contextualmente relevante. Aquí hay algunas razones clave por las que el tamaño de contexto es importante:

- 1. Comprensión de contenido extenso:** Los modelos con tamaños de contexto más grandes pueden comprender y analizar mejor textos más largos, como artículos, informes o incluso libros completos. Esto es crucial para tareas como el resumen de documentos, respuesta a preguntas y análisis de contenido.
- 2. Mantenimiento de la coherencia:** Una ventana de contexto más grande permite que el modelo mantenga la coherencia y consistencia a lo largo de segmentos más extensos de texto. Esto es importante para tareas como la generación de historias, sistemas de diálogo y creación de contenido, donde mantener una narrativa o tema consistente es esencial. También es absolutamente crucial cuando se utilizan LLMs para generar o transformar datos estructurados.

3. **Captura de dependencias de largo alcance:** Algunas tareas de lenguaje requieren comprender relaciones entre palabras o frases que están alejadas en un texto. Los modelos con tamaños de contexto más grandes están mejor equipados para capturar estas dependencias de largo alcance, lo cual puede ser importante para tareas como el análisis de sentimiento, la traducción y la comprensión del lenguaje.
4. **Manejo de instrucciones complejas:** En aplicaciones donde los modelos de lenguaje se utilizan para seguir instrucciones complejas de múltiples pasos, un tamaño de contexto mayor permite que el modelo considere el conjunto completo de instrucciones al generar una respuesta, en lugar de solo las últimas palabras.

Ejemplos de Modelos de Lenguaje con Diferentes Tamaños de Contexto

Aquí hay algunos ejemplos de modelos de lenguaje con diferentes tamaños de contexto:

- OpenAI GPT-3.5 Turbo: 4.095 tokens
- Mistral 7B Instruct: 32.768 tokens
- Anthropic Claude v1: 100.000 tokens
- OpenAI GPT-4 Turbo: 128.000 tokens
- Anthropic Claude v2: 200.000 tokens
- Google Gemini Pro 1.5: 2,8M tokens

Como pueden ver, existe una amplia gama de tamaños de contexto entre estos modelos, desde aproximadamente 4.000 tokens para el modelo OpenAI GPT-3.5 Turbo hasta 200.000 tokens para el modelo Anthropic Claude v2. Algunos modelos, como Google PaLM 2 y OpenAI GPT-4, ofrecen diferentes variantes con tamaños de contexto más grandes (por ejemplo, versiones “32k”), que pueden manejar secuencias de entrada aún más largas. ¡Y en este momento (abril de 2024) Google Gemini Pro está presumiendo de casi 3 millones de tokens!

Vale la pena mencionar que el tamaño de contexto puede variar según la implementación específica y la versión de un modelo particular. Por ejemplo, el modelo original OpenAI GPT-4 tiene un tamaño de contexto de 8.191 tokens, mientras que las variantes posteriores de GPT-4 como Turbo y 4o tienen un tamaño de contexto mucho mayor de 128.000 tokens.

Sam Altman ha comparado las limitaciones actuales de contexto con los kilobytes de memoria de trabajo con los que tenían que lidiar los programadores de computadoras personales en los 80, y ha dicho que en un futuro cercano podremos ajustar “todos tus datos personales” en el contexto de un modelo de lenguaje grande.

Eligiendo el Tamaño de Contexto Adecuado

Al seleccionar un modelo de lenguaje para una aplicación particular, es importante considerar los requisitos de tamaño de contexto de la tarea en cuestión. Para tareas que involucran textos cortos y aislados, como el análisis de sentimiento o respuestas a preguntas simples, un tamaño de contexto más pequeño puede ser suficiente. Sin embargo, para tareas que requieren comprender y generar textos más largos y complejos, probablemente será necesario un tamaño de contexto más grande.

Vale la pena señalar que los tamaños de contexto más grandes a menudo conllevan mayores costos computacionales y tiempos de procesamiento más lentos, ya que el modelo necesita considerar más información al generar una respuesta. Como tal, debes encontrar un equilibrio entre el tamaño de contexto y el rendimiento al elegir un modelo de lenguaje para tu aplicación.

¿Por qué no simplemente elegir el modelo con el tamaño de contexto más grande y llenarlo con tanta información como sea posible? Bueno, además de los factores de rendimiento, la otra consideración principal es el costo. En marzo de 2024, un *solo* ciclo de pregunta-respuesta usando Google Gemini Pro 1.5 con un contexto completo te costará casi \$8 (USD). Si tienes un caso de uso que justifica ese gasto, ¡adelante! Pero para la mayoría de las aplicaciones, es demasiado costoso por órdenes de magnitud.

Encontrando Agujas en Pajares

El concepto de encontrar una aguja en un pajar ha sido durante mucho tiempo una metáfora para los desafíos de recuperación en grandes conjuntos de datos. En el ámbito de los LLMs, modificamos un poco esta analogía. Imagina que no solo estamos buscando un único hecho enterrado dentro de un texto extenso (como una antología completa de ensayos de Paul Graham), sino múltiples hechos dispersos por todas partes. Este escenario se asemeja más a encontrar varias agujas en un campo extenso, no solo en un pajar. Y aquí está lo interesante: no solo necesitamos localizar estas agujas, sino que también tenemos que entrelazarlas en un hilo coherente.

Cuando se enfrentan a la tarea de recuperar y razonar sobre múltiples hechos integrados en contextos largos, los LLMs se enfrentan a un doble desafío. Primero, está el problema directo de la precisión en la recuperación—naturalmente disminuye a medida que aumenta el número de hechos. Esto es esperado; después de todo, mantener el registro de múltiples detalles a través de un texto extenso pone a prueba incluso a los modelos más sofisticados.

Segundo, y quizás más crítico, es el desafío de razonar con estos hechos. Una cosa es extraer hechos; otra muy distinta es sintetizarlos en una narrativa o respuesta coherente. Aquí es donde viene la verdadera prueba. El rendimiento de los LLMs en tareas de razonamiento tiende a degradarse más que en tareas simples de recuperación. Esta

degradación no se trata solo del volumen; se trata de la intrincada danza del contexto, la relevancia y la inferencia.

¿Por qué sucede esto? Bien, considera la dinámica de la memoria y la atención en la cognición humana, que se refleja hasta cierto punto en los LLMs. Al procesar grandes cantidades de información, los LLMs, como los humanos, pueden perder el rastro de detalles anteriores mientras absorben otros nuevos. Esto es especialmente cierto en modelos que no están diseñados explícitamente para priorizar o revisar automáticamente segmentos anteriores del texto.

Además, la capacidad de un LLM para tejer estos hechos recuperados en una respuesta coherente es similar a la construcción narrativa. Esto requiere no solo la recuperación de información sino una comprensión profunda y ubicación contextual, lo que sigue siendo un gran desafío para la IA actual.

Entonces, ¿qué significa esto para nosotros como desarrolladores e integradores de estas tecnologías? Necesitamos ser agudamente conscientes de estas limitaciones al diseñar sistemas que dependen de LLMs para manejar tareas complejas y extensas. Entender que el rendimiento podría degradarse bajo ciertas condiciones nos ayuda a establecer expectativas realistas y diseñar mejores mecanismos de respaldo o estrategias complementarias.

Modalidades: Más Allá del Texto

Si bien la mayoría de los modelos de lenguaje actuales se centran en procesar y generar texto, existe una tendencia creciente hacia los modelos multimodales que pueden recibir y generar de forma nativa múltiples tipos de datos, como imágenes, audio y video. Estos modelos multimodales abren nuevas posibilidades para aplicaciones impulsadas por IA que pueden entender y generar contenido a través de diferentes modalidades.

¿Qué son las Modalidades?

En el contexto de los modelos de lenguaje, las modalidades se refieren a los diferentes tipos de datos que un modelo puede procesar y generar. La modalidad más común es el texto, que incluye lenguaje escrito en varias formas como libros, artículos, sitios web y publicaciones en redes sociales. Sin embargo, hay varias otras modalidades que se están incorporando cada vez más en los modelos de lenguaje:

- **Imágenes:** Datos visuales como fotografías, ilustraciones y diagramas.
- **Audio:** Datos de sonido como voz, música y sonidos ambientales.
- **Video:** Datos visuales en movimiento, a menudo acompañados de audio, como clips de video y películas.

Cada modalidad presenta desafíos y oportunidades únicas para los modelos de lenguaje. Por ejemplo, las imágenes requieren que el modelo entienda conceptos y relaciones visuales, mientras que el audio requiere que el modelo procese y genere voz y otros sonidos.

Modelos de Lenguaje Multimodales

Los modelos de lenguaje multimodales están diseñados para manejar múltiples modalidades dentro de un único modelo. Estos modelos típicamente tienen componentes o capas especializadas que pueden tanto entender las entradas como generar datos de salida en diferentes modalidades. Algunos ejemplos notables de modelos de lenguaje multimodales incluyen:

- **OpenAI's GPT-4o:** GPT-4o es un modelo de lenguaje grande que entiende y procesa de forma nativa el audio de voz además del texto. Esta capacidad permite a GPT-4o realizar tareas como transcribir lenguaje hablado, generar texto a partir de entradas de audio y proporcionar respuestas basadas en consultas habladas.

- **OpenAI's GPT-4 con entrada visual:** GPT-4 es un modelo de lenguaje grande que puede procesar tanto texto como imágenes. Cuando se le proporciona una imagen como entrada, GPT-4 puede analizar el contenido de la imagen y generar texto que describe o responde a la información visual.
- **Google's Gemini:** Gemini es un modelo multimodal que puede manejar texto, imágenes y video. Utiliza una arquitectura unificada que permite la comprensión y generación entre modalidades, permitiendo tareas como la generación de descripciones de imágenes, resumen de videos y respuesta a preguntas visuales.
- **DALL-E y Stable Diffusion:** Si bien no son modelos de lenguaje en el sentido tradicional, estos modelos demuestran el poder de la IA multimodal al generar imágenes a partir de descripciones textuales. Muestran el potencial de los modelos que pueden traducir entre diferentes modalidades.

Beneficios y Aplicaciones de los Modelos Multimodales

Los modelos de lenguaje multimodales ofrecen varios beneficios y permiten una amplia gama de aplicaciones, incluyendo:

- **Comprensión mejorada:** Al procesar información de múltiples modalidades, estos modelos pueden obtener una comprensión más completa del mundo, similar a cómo los humanos aprenden de varios estímulos sensoriales.
- **Generación intermodal:** Los modelos multimodales pueden generar contenido en una modalidad basándose en la entrada de otra, como crear una imagen a partir de una descripción de texto o generar un resumen en video de un artículo escrito.
- **Accesibilidad:** Los modelos multimodales pueden hacer que la información sea más accesible al traducir entre modalidades, como generar descripciones textuales de imágenes para usuarios con discapacidad visual o crear versiones de audio de contenido escrito.

- **Aplicaciones creativas:** Los modelos multimodales pueden utilizarse para tareas creativas como generar arte, música o videos basados en indicaciones textuales, abriendo nuevas posibilidades para artistas y creadores de contenido.

A medida que los modelos de lenguaje multimodales continúan avanzando, probablemente desempeñarán un papel cada vez más importante en el desarrollo de aplicaciones impulsadas por IA que pueden entender y generar contenido a través de múltiples modalidades. Esto permitirá interacciones más naturales e intuitivas entre humanos y sistemas de IA, además de desbloquear nuevas posibilidades para la expresión creativa y la difusión del conocimiento.

Ecosistemas de Proveedores

Cuando se trata de incorporar modelos de lenguaje grandes (MLGs) en aplicaciones, existe una creciente gama de opciones entre las cuales elegir. Cada proveedor importante de MLG, como OpenAI, Anthropic, Google y Cohere, ofrece su propio ecosistema de modelos, APIs y herramientas. Elegir el proveedor adecuado implica considerar varios factores, incluyendo precios, rendimiento, filtrado de contenido, privacidad de datos y opciones de personalización.

OpenAI

OpenAI es uno de los proveedores más conocidos de MLGs, con su serie GPT (GPT-3, GPT-4) siendo ampliamente utilizada en varias aplicaciones. OpenAI ofrece una API fácil de usar que permite integrar fácilmente sus modelos en aplicaciones. Proporcionan una variedad de modelos con diferentes capacidades y precios, desde el modelo básico Ada hasta el potente modelo Davinci.

El ecosistema de OpenAI también incluye herramientas como OpenAI Playground, que permite experimentar con prompts y realizar ajustes finos en los modelos para casos

de uso específicos. Ofrecen opciones de filtrado de contenido para ayudar a prevenir la generación de contenido inapropiado o dañino.

Cuando uso los modelos de OpenAI directamente, confío en la biblioteca [ruby-openai](#) de Alex Rudall.

Anthropic

Anthropic es otro actor importante en el espacio de los MLGs, con sus modelos Claude ganando popularidad por su fuerte rendimiento y consideraciones éticas. Anthropic se centra en desarrollar sistemas de IA seguros y responsables, con un fuerte énfasis en el filtrado de contenido y la prevención de resultados dañinos.

El ecosistema de Anthropic incluye la API de Claude, que permite integrar el modelo en sus aplicaciones, así como herramientas para la ingeniería de prompts y ajuste fino. También ofrecen el modelo Claude Instant, que incorpora capacidades de búsqueda web para obtener respuestas más actualizadas y precisas.

Cuando uso los modelos de Anthropic directamente, confío en la biblioteca [anthropic](#) de Alex Rudall.

Google

Google ha desarrollado varios MLGs potentes, incluyendo Gemini, BERT, T5 y PaLM. Estos modelos son conocidos por su fuerte rendimiento en una amplia gama de tareas de procesamiento del lenguaje natural. El ecosistema de Google incluye las bibliotecas TensorFlow y Keras, que proporcionan herramientas y marcos para construir y entrenar modelos de aprendizaje automático.

Google también ofrece una Plataforma de IA en la Nube, que permite implementar y escalar fácilmente sus modelos en la nube. Proporcionan una variedad de modelos preentrenados y APIs para tareas como análisis de sentimientos, reconocimiento de entidades y traducción.

Meta

Meta, anteriormente conocido como Facebook, está profundamente involucrado en el desarrollo de modelos de lenguaje grandes, destacado por su lanzamiento de modelos como LLaMA y OPT. Estos modelos sobresalen por su fuerte rendimiento en diversas tareas de lenguaje y están disponibles principalmente a través de canales de código abierto, respaldando el compromiso de Meta con la investigación y la colaboración comunitaria.

El ecosistema de Meta está construido principalmente alrededor de PyTorch, una biblioteca de aprendizaje automático de código abierto preferida por sus capacidades de computación dinámica y flexibilidad, facilitando la investigación y desarrollo innovador en IA.

Además de sus ofertas técnicas, Meta hace especial énfasis en el desarrollo ético de la IA. Implementan un sólido filtrado de contenido y se centran en reducir sesgos, alineándose con sus objetivos más amplios de seguridad y responsabilidad en las aplicaciones de IA.

Cohere

Cohere es un participante más reciente en el espacio de los LLM, centrándose en hacer que los LLM sean más accesibles y fáciles de usar que los competidores. Su ecosistema incluye la API de Cohere, que proporciona acceso a una gama de modelos preentrenados para tareas como generación de texto, clasificación y resumen.

Cohere también ofrece herramientas para la ingeniería de prompts, el ajuste fino y el filtrado de contenido. Enfatizan la privacidad y seguridad de los datos, con características como el almacenamiento cifrado de datos y controles de acceso.

Ollama

Ollama es una plataforma autohospedada que permite a los usuarios gestionar e implementar varios modelos de lenguaje grandes (LLMs) localmente en sus máquinas,

dándoles control completo sobre sus modelos de IA sin depender de servicios en la nube externos. Esta configuración es ideal para aquellos que priorizan la privacidad de los datos y desean manejar sus operaciones de IA internamente.

La plataforma admite una variedad de modelos, incluyendo versiones de Llama, Phi, Gemma y Mistral, que varían en tamaño y requisitos computacionales. Ollama facilita la descarga y ejecución de estos modelos directamente desde la línea de comandos usando comandos simples como `ollama run <model_name>`, y está diseñado para funcionar en diferentes sistemas operativos, incluyendo macOS, Linux y Windows.

Para los desarrolladores que buscan integrar modelos de código abierto en sus aplicaciones sin usar una API remota, Ollama ofrece una CLI para gestionar ciclos de vida de modelos similar a las herramientas de gestión de contenedores. También admite configuraciones y prompts personalizados, permitiendo un alto grado de personalización para adaptar los modelos a necesidades o casos de uso específicos.

Ollama está particularmente adaptado para usuarios expertos en tecnología y desarrolladores debido a su interfaz de línea de comandos y la flexibilidad que ofrece en la gestión e implementación de modelos de IA. Esto lo convierte en una herramienta poderosa para empresas e individuos que requieren capacidades robustas de IA sin comprometer la seguridad y el control.

Plataformas Multi-Modelo

Adicionalmente, hay proveedores que alojan una amplia variedad de modelos de código abierto, como Together.ai y Groq. Estas plataformas ofrecen flexibilidad y personalización, permitiéndote ejecutar y, en algunos casos, incluso ajustar modelos de código abierto según tus necesidades específicas. Por ejemplo, Together.ai proporciona acceso a una gama de LLMs de código abierto, permitiendo a los usuarios experimentar con diferentes modelos y configuraciones. Groq se centra en ofrecer completado de ultra alto rendimiento que, al momento de escribir este libro, parece casi mágico

Eligiendo un Proveedor de LLM

Al elegir un proveedor de LLM, debes considerar factores como:

- **Precios:** Diferentes proveedores ofrecen diferentes modelos de precios, desde pago por uso hasta planes basados en suscripción. Es importante considerar el uso esperado y el presupuesto al seleccionar un proveedor.
- **Rendimiento:** El rendimiento de los LLMs puede variar significativamente entre proveedores, por lo que es importante evaluar y probar los modelos en casos de uso específicos antes de tomar una decisión.
- **Filtrado de Contenido:** Dependiendo de la aplicación, el filtrado de contenido puede ser una consideración crítica. Algunos proveedores ofrecen opciones de filtrado de contenido más robustas que otros.
- **Privacidad de Datos:** Si la aplicación maneja datos sensibles de usuarios, es importante elegir un proveedor con prácticas sólidas de privacidad y seguridad de datos.
- **Personalización:** Algunos proveedores ofrecen más flexibilidad en términos de ajuste fino y personalización de modelos para casos de uso específicos.

En última instancia, la elección del proveedor de LLM depende de los requisitos y restricciones específicas de la aplicación. Al evaluar cuidadosamente las opciones y considerar factores como precios, rendimiento y privacidad de datos, puedes seleccionar el proveedor que mejor satisfaga tus necesidades.

También vale la pena señalar que el panorama de los LLM está en constante evolución, con nuevos proveedores y modelos emergiendo regularmente. Debes mantenerte actualizado con los últimos desarrollos y estar abierto a explorar nuevas opciones conforme estén disponibles.

OpenRouter

A lo largo de este libro me basaré exclusivamente en [OpenRouter](#) como mi proveedor de API preferido. La razón es simple: es una ventanilla única para todos los modelos comerciales y de código abierto más populares. Si estás ansioso por empezar a programar con IA, uno de los mejores lugares para comenzar es con mi propia [Biblioteca Ruby de OpenRouter](#).

Pensando en el Rendimiento

Al incorporar modelos de lenguaje en las aplicaciones, el rendimiento es una consideración crítica. El rendimiento de un modelo de lenguaje puede medirse en términos de su *latencia* (el tiempo que tarda en generar una respuesta) y *capacidad de procesamiento* (el número de solicitudes que puede manejar por unidad de tiempo).

El *Tiempo hasta el Primer Token* (TTFT) es otra métrica de rendimiento esencial, particularmente relevante para chatbots y aplicaciones que requieren respuestas interactivas en tiempo real. El TTFT mide la latencia desde el momento en que se recibe la solicitud del usuario hasta el momento en que se genera la primera palabra (o token) de la respuesta. Esta métrica es crucial para mantener una experiencia de usuario fluida y atractiva, ya que las respuestas retrasadas pueden provocar frustración y desconexión del usuario.

Estas métricas de rendimiento pueden tener un impacto significativo en la experiencia del usuario y la escalabilidad de la aplicación.

Varios factores pueden influir en el rendimiento de un modelo de lenguaje, incluyendo:

Número de Parámetros: Los modelos más grandes con más parámetros generalmente requieren más recursos computacionales y pueden tener una latencia más alta y un menor rendimiento en comparación con los modelos más pequeños.

Hardware: El rendimiento de un modelo de lenguaje puede variar significativamente según el hardware en el que se ejecute. Los proveedores de servicios en la nube ofrecen instancias de GPU y TPU optimizadas para cargas de trabajo de aprendizaje automático, que pueden acelerar considerablemente la inferencia del modelo.



Una de las ventajas de OpenRouter es que para muchos de los modelos que ofrece, tienes una opción de proveedores en la nube con una variedad de perfiles de rendimiento y costos.

Cuantización: Las técnicas de cuantización se pueden utilizar para reducir la huella de memoria y los requisitos computacionales de un modelo mediante la representación de pesos y activaciones con tipos de datos de menor precisión. Esto puede mejorar el rendimiento sin sacrificar significativamente la calidad. Como desarrollador de aplicaciones, probablemente no te involucrarás en el entrenamiento de tus propios modelos con diferentes niveles de cuantización, pero es bueno al menos estar familiarizado con la terminología.

Procesamiento por Lotes: Procesar múltiples solicitudes simultáneamente en lotes puede mejorar el rendimiento al amortizar la sobrecarga de carga del modelo y la transferencia de datos.

Almacenamiento en Caché: Almacenar en caché los resultados de indicaciones o secuencias de entrada frecuentemente utilizadas puede reducir el número de solicitudes de inferencia y mejorar el rendimiento general.

Al seleccionar un modelo de lenguaje para una aplicación en producción, es importante evaluar su rendimiento en cargas de trabajo y configuraciones de hardware representativas. Esto puede ayudar a identificar posibles cuellos de botella y asegurar que el modelo pueda cumplir con los objetivos de rendimiento requeridos.

También vale la pena considerar los compromisos entre el rendimiento del modelo y otros factores como el costo, la flexibilidad y la facilidad de integración. Por ejemplo, usar un modelo más pequeño y menos costoso con menor latencia puede ser preferible

para aplicaciones que requieren respuestas en tiempo real, mientras que un modelo más grande y potente puede ser más adecuado para el procesamiento por lotes o tareas de razonamiento complejo.

Experimentando con Diferentes Modelos LLM

Elegir un LLM raramente es una decisión permanente. Como se lanzan nuevos y mejores modelos regularmente, es bueno construir aplicaciones de manera modular que permita intercambiar diferentes modelos de lenguaje a lo largo del tiempo. Las indicaciones y conjuntos de datos a menudo pueden reutilizarse entre modelos con cambios mínimos. Esto te permite aprovechar los últimos avances en modelado de lenguaje sin tener que rediseñar completamente tus aplicaciones.



La capacidad de cambiar fácilmente entre una amplia gama de opciones de modelos es otra razón por la que me encanta OpenRouter.

Al actualizar a un nuevo modelo de lenguaje, es importante probar y validar exhaustivamente su rendimiento y calidad de salida para asegurar que cumpla con los requisitos de la aplicación. Esto puede implicar reentrenar o ajustar el modelo con datos específicos del dominio, así como actualizar cualquier componente posterior que dependa de las salidas del modelo.

Al diseñar aplicaciones teniendo en cuenta el rendimiento y la modularidad, puedes crear sistemas escalables, eficientes y preparados para el futuro que puedan adaptarse al panorama en rápida evolución de la tecnología de modelado de lenguaje.

Sistemas de IA Compuestos

Antes de cerrar nuestra introducción, vale la pena mencionar que antes de 2023 y la explosión de interés en IA generativa provocada por ChatGPT, los enfoques

tradicionales de IA generalmente se basaban en la integración de modelos únicos y cerrados. En contraste, los *Sistemas de IA Compuestos* aprovechan tuberías complejas de componentes interconectados que trabajan juntos para lograr un comportamiento inteligente.

En su núcleo, los sistemas de IA compuestos consisten en múltiples módulos, cada uno diseñado para realizar tareas o funciones específicas. Estos módulos pueden incluir generadores, recuperadores, clasificadores, ordenadores y varios otros componentes especializados. Al desglosar el sistema general en unidades más pequeñas y enfocadas, los desarrolladores pueden crear arquitecturas de IA más flexibles, escalables y mantenibles.

Una de las ventajas clave de los sistemas de IA compuestos es su capacidad para combinar las fortalezas de diferentes técnicas y modelos de IA. Por ejemplo, un sistema podría usar un modelo de lenguaje grande (LLM) para la comprensión y generación del lenguaje natural, mientras emplea un modelo separado para la recuperación de información o la toma de decisiones basada en reglas. Este enfoque modular te permite seleccionar las mejores herramientas y técnicas para cada tarea específica, en lugar de depender de una solución única para todo.

Sin embargo, construir sistemas de IA compuestos también presenta desafíos únicos. En particular, asegurar la coherencia general y la consistencia del comportamiento del sistema requiere mecanismos robustos de prueba, monitoreo y gobernanza.



El advenimiento de LLMs poderosos como GPT-4 nos permite experimentar con sistemas de IA compuestos más fácilmente que nunca, ya que estos modelos avanzados son capaces de manejar múltiples roles dentro de un sistema compuesto, como clasificación, clasificación por rangos y generación, además de sus capacidades de comprensión del lenguaje natural. Esta versatilidad permite a los desarrolladores prototipar e iterar rápidamente en arquitecturas de IA compuestas, abriendo nuevas posibilidades para el desarrollo de aplicaciones inteligentes.

Patrones de Implementación para Sistemas de IA Compuestos

Los sistemas de IA compuestos pueden implementarse utilizando varios patrones, cada uno diseñado para abordar requisitos y casos de uso específicos. Exploraremos cuatro patrones comunes de implementación: Pregunta y Respuesta, Solucionadores de Problemas Multi-Agente/Agénticos, IA Conversacional, y CoPilotos.

Pregunta y Respuesta

Los sistemas de Pregunta y Respuesta (Q&A) se centran en proporcionar una recuperación de información mejorada con las capacidades de comprensión de los modelos de IA para funcionar como algo más que un simple motor de búsqueda. Al combinar modelos de lenguaje potentes con fuentes de conocimiento externas utilizando [Generación Aumentada por Recuperación \(RAG\)](#), los sistemas de Pregunta y Respuesta evitan las alucinaciones y proporcionan respuestas precisas y contextualmente relevantes a las consultas de los usuarios.

Los componentes clave de un sistema Q&A basado en LLM incluyen:

- **Comprendión y reformulación de consultas:** Análisis de las consultas de los usuarios y reformulación de las mismas para que coincidan mejor con las fuentes de conocimiento subyacentes.
- **Recuperación de conocimiento:** Recuperación de información relevante de fuentes de datos estructurados o no estructurados basada en la consulta reformulada.
- **Generación de respuestas:** Generación de respuestas coherentes e informativas mediante la integración del conocimiento recuperado con las capacidades generativas del modelo de lenguaje.

Los subsistemas RAG son particularmente importantes en dominios de Q&A donde

proporcionar información precisa y actualizada es crucial, como el soporte al cliente, la gestión del conocimiento, o aplicaciones educativas.

Solucionadores de Problemas Multi-Agente/Agénticos

Los sistemas multi-agente, también conocidos como *Agénticos*, consisten en múltiples agentes autónomos trabajando juntos para resolver problemas complejos. Cada agente tiene un rol específico, conjunto de habilidades y acceso a herramientas o fuentes de información relevantes. Al colaborar e intercambiar información, estos agentes pueden abordar tareas que serían difíciles o imposibles de manejar para un solo agente.

Los principios clave de los solucionadores de problemas multi-agente incluyen:

- **Especialización:** Cada agente se centra en un aspecto específico del problema, aprovechando sus capacidades y conocimientos únicos.
- **Colaboración:** Los agentes se comunican y coordinan sus acciones para lograr un objetivo común, a menudo a través del paso de mensajes o memoria compartida.
- **Adaptabilidad:** El sistema puede adaptarse a condiciones o requisitos cambiantes ajustando los roles y comportamientos de los agentes individuales.

Los sistemas multi-agente son adecuados para aplicaciones que requieren resolución distribuida de problemas, como la optimización de la cadena de suministro, gestión del tráfico, o planificación de respuesta a emergencias.

IA Conversacional

Los sistemas de IA conversacional permiten interacciones en lenguaje natural entre usuarios y agentes inteligentes. Estos sistemas combinan comprensión del lenguaje natural, gestión de diálogo y capacidades de generación de lenguaje para proporcionar experiencias conversacionales atractivas y personalizadas.

Los componentes principales de un sistema de IA conversacional incluyen:

- **Reconocimiento de intención:** Identificación de la intención del usuario basada en su entrada, como hacer una pregunta, realizar una solicitud o expresar un sentimiento.
- **Extracción de entidades:** Extracción de entidades o parámetros relevantes de la entrada del usuario, como fechas, ubicaciones o nombres de productos.
- **Gestión de diálogo:** Mantenimiento del estado de la conversación, determinación de la respuesta apropiada basada en la intención y contexto del usuario, y manejo de interacciones de múltiples turnos.
- **Generación de respuestas:** Generación de respuestas similares a las humanas utilizando modelos de lenguaje, plantillas o métodos basados en recuperación.

Los sistemas de IA conversacional se utilizan comúnmente en chatbots de servicio al cliente, asistentes virtuales, e interfaces controladas por voz. Como se mencionó anteriormente, la mayoría de los enfoques, patrones y ejemplos de código en este libro están directamente extraídos de mi trabajo en un sistema de IA conversacional grande llamado [Olympia](#).

CoPilots

Los CoPilots son asistentes impulsados por IA que trabajan junto a usuarios humanos para mejorar su productividad y capacidad de toma de decisiones. Estos sistemas aprovechan una combinación de procesamiento del lenguaje natural, aprendizaje automático y conocimiento específico del dominio para proporcionar recomendaciones inteligentes, automatizar tareas y ofrecer apoyo contextual.

Las características principales de los CoPilots incluyen:

- **Personalización:** Adaptación a las preferencias individuales del usuario, flujos de trabajo y estilos de comunicación.
- **Asistencia proactiva:** Anticipación a las necesidades del usuario y ofrecimiento de sugerencias o acciones relevantes sin indicaciones explícitas.

- **Aprendizaje continuo:** Mejora del rendimiento a lo largo del tiempo mediante el aprendizaje basado en la retroalimentación del usuario, las interacciones y los datos.

Los CoPilots se utilizan cada vez más en varios dominios, como el desarrollo de software (por ejemplo, completado de código y detección de errores), la escritura creativa (por ejemplo, sugerencias de contenido y edición), y el análisis de datos (por ejemplo, información y recomendaciones de visualización)

Estos patrones de implementación muestran la versatilidad y el potencial de los sistemas de IA compuestos. Al comprender las características y casos de uso de cada patrón, puedes tomar decisiones informadas al diseñar e implementar aplicaciones inteligentes. Si bien este libro no trata específicamente sobre la implementación de sistemas de IA compuestos, muchos, si no todos los mismos enfoques y patrones, se aplican a la integración de componentes discretos de IA dentro del desarrollo de aplicaciones tradicionales.

Roles en los Sistemas de IA Compuestos

Los sistemas de IA compuestos se construyen sobre una base de módulos interconectados, cada uno diseñado para desempeñar un rol específico. Estos módulos trabajan juntos para crear comportamientos inteligentes y resolver problemas complejos. Es útil estar familiarizado con estos roles al pensar dónde podrías implementar o reemplazar partes de tu aplicación con componentes discretos de IA.

Generador

Los generadores son responsables de producir nuevos datos o contenido basados en patrones aprendidos o indicaciones de entrada. El mundo de la IA tiene muchos tipos diferentes de generadores, pero en el contexto de los tipos de modelos de lenguaje que se muestran en este libro, los generadores pueden crear texto similar al humano, completar

oraciones parciales o generar respuestas a consultas de usuarios. Juegan un papel crucial en tareas como la creación de contenido, la generación de diálogos y la aumentación de datos.

Recuperador

Los recuperadores se utilizan para buscar y extraer información relevante de grandes conjuntos de datos o bases de conocimiento. Emplean técnicas como búsqueda semántica, coincidencia de palabras clave o similitud vectorial para encontrar los puntos de datos más pertinentes basados en una consulta o contexto dado. Los recuperadores son esenciales para tareas que requieren acceso rápido a información específica, como responder preguntas, verificar hechos o recomendar contenido.

Clasificador por Rangos

Los clasificadores por rangos son responsables de ordenar o priorizar un conjunto de elementos basados en ciertos criterios o puntuaciones de relevancia. Asignan pesos o puntuaciones a cada elemento y luego los ordenan en consecuencia. Los clasificadores por rangos se utilizan comúnmente en motores de búsqueda, sistemas de recomendación o cualquier aplicación donde sea crucial presentar los resultados más relevantes a los usuarios.

Clasificador

Los clasificadores se utilizan para categorizar o etiquetar puntos de datos basados en clases o categorías predefinidas. Aprenden de datos de entrenamiento etiquetados y luego predicen la clase de nuevas instancias no vistas. Los clasificadores son fundamentales para tareas como el análisis de sentimientos, la detección de spam o el reconocimiento de imágenes, donde el objetivo es asignar una categoría específica a cada entrada.

Herramientas y Agentes

Además de estos roles principales, los sistemas de IA compuestos a menudo incorporan herramientas y agentes para mejorar su funcionalidad y adaptabilidad:

- **Herramientas:** Las herramientas son componentes de software discretos o APIs que realizan acciones o cálculos específicos. Pueden ser invocadas por otros módulos, como generadores o recuperadores, para realizar subtareas o recopilar información adicional. Ejemplos de herramientas incluyen motores de búsqueda web, calculadoras o bibliotecas de visualización de datos.
- **Agentes:** Los agentes son entidades autónomas que pueden percibir su entorno, tomar decisiones y realizar acciones para alcanzar objetivos específicos. A menudo se basan en una combinación de diferentes técnicas de IA, como planificación, razonamiento y aprendizaje, para operar eficazmente en condiciones dinámicas o inciertas. Los agentes pueden utilizarse para modelar comportamientos complejos o para coordinar las acciones de múltiples módulos dentro de un sistema de IA compuesto.

En un sistema de IA compuesto puro, la interacción entre estos componentes se orquesta a través de interfaces y protocolos de comunicación bien definidos. Los datos fluyen entre módulos, con la salida de un componente sirviendo como entrada para otro. Esta arquitectura modular permite flexibilidad, escalabilidad y mantenibilidad, ya que los componentes individuales pueden actualizarse, reemplazarse o extenderse sin afectar a todo el sistema.

Al aprovechar el poder de estos componentes y sus interacciones, los sistemas de IA compuestos pueden abordar problemas complejos del mundo real que requieren una combinación de diferentes capacidades de IA. A medida que exploramos los enfoques y patrones para integrar la IA en el desarrollo de aplicaciones, ten en cuenta que los mismos principios y técnicas utilizados en los sistemas de IA compuestos pueden

aplicarse para crear aplicaciones inteligentes, adaptativas y centradas en el usuario.

En los siguientes capítulos de la Parte 1, profundizaremos en los enfoques y técnicas fundamentales para integrar componentes de IA en tu proceso de desarrollo de aplicaciones. Desde la ingeniería de prompts y la generación aumentada por recuperación hasta los datos auto-reparables y la orquestación inteligente de flujos de trabajo, cubriremos una amplia gama de patrones y mejores prácticas para ayudarte a construir aplicaciones de vanguardia impulsadas por IA.

Parte 1: Enfoques y Técnicas Fundamentales

Esta parte del libro presenta diferentes formas de integrar el uso de la IA en tus aplicaciones. Los capítulos cubren una serie de enfoques y técnicas relacionadas, desde conceptos más generales como [Narrow The Path](#) y [Retrieval Augmented Generation](#), hasta ideas para programar tu propia capa de abstracción sobre las APIs de completación de chat LLM.

El objetivo de esta parte del libro es ayudarte a comprender los tipos de comportamiento que puedes implementar con IA, antes de profundizar demasiado en los patrones de implementación específicos que son el foco de la [Parte 2](#).

Los enfoques en la Parte 1 se basan en ideas que he utilizado en mi código, patrones clásicos de arquitectura e integración de aplicaciones empresariales, además de metáforas que he empleado al explicar las capacidades de la IA a otras personas, incluyendo a interesados del negocio sin conocimientos técnicos.

Estrechar el Camino



“Estrechar el camino” se refiere a enfocar la IA en la tarea en cuestión. Lo uso como un mantra cuando me siento frustrado porque la IA actúa de manera “tonta” o inesperada. El mantra me recuerda que el fallo probablemente sea mi culpa y que probablemente debería estrechar más el camino.

La necesidad de estrechar el camino surge de las vastas cantidades de conocimiento contenidas en los modelos de lenguaje grandes, especialmente los modelos de clase mundial como los de OpenAI y Anthropic que tienen literalmente billones de parámetros.

Tener acceso a tal amplitud de conocimiento es sin duda poderoso y produce comportamientos emergentes como la teoría de la mente y la capacidad de razonar de manera similar a los humanos. Sin embargo, ese volumen abrumador de información también presenta desafíos cuando se trata de generar respuestas precisas y exactas a prompts específicos, especialmente si esos prompts están destinados a exhibir un comportamiento determinista que pueda integrarse con el desarrollo de software y algoritmos “normales”.

Varios factores conducen a estos desafíos.

Sobrecarga de información: Los modelos de lenguaje grandes están entrenados con cantidades masivas de datos que abarcan varios dominios, fuentes y períodos de tiempo. Este extenso conocimiento les permite participar en diversos temas y generar respuestas basadas en una amplia comprensión del mundo. Sin embargo, cuando se enfrentan a un prompt específico, el modelo puede tener dificultades para filtrar información irrelevante, contradictoria o desactualizada/obsoleta, lo que lleva a respuestas que carecen de enfoque o precisión. Dependiendo de lo que estés tratando de hacer, el mero volumen de información *contradictoria* disponible para el modelo puede fácilmente sobrepasar su capacidad para proporcionar la respuesta o el comportamiento que buscas.

Ambigüedad contextual: Dado el vasto *espacio latente* de conocimiento, los modelos de lenguaje grandes pueden encontrar ambigüedad al tratar de entender el *contexto* de tu prompt. Sin un estrechamiento o guía adecuados, el modelo puede generar respuestas que están tangencialmente relacionadas pero que no son directamente relevantes para tus intenciones. Este tipo de fallo lleva a respuestas que están fuera de tema, son inconsistentes o no abordan tus necesidades declaradas. En este caso, estrechar el camino se refiere a la *desambiguación* del contexto, asegurando que el contexto que proporcionas hace que el modelo se centre solo en la información más relevante de su conocimiento base.



Nota: Cuando estás empezando con la “ingeniería de prompts”, es más probable que le pidas al modelo que haga cosas sin explicar adecuadamente el resultado deseado; ¡se necesita práctica para no ser ambiguo!

Inconsistencias temporales: Como los modelos de lenguaje están entrenados con datos que fueron creados en diferentes períodos de tiempo, pueden poseer conocimientos que están desactualizados, superados o que ya no son precisos. Por ejemplo, la información sobre eventos actuales, descubrimientos científicos o avances tecnológicos puede haber evolucionado desde que se recopilaron los datos de entrenamiento del modelo. Sin estrechar el camino para priorizar fuentes más recientes y confiables, el modelo podría generar respuestas basadas en información desactualizada o incorrecta, lo que lleva a inexactitudes e inconsistencias en sus resultados.

Matices específicos del dominio: Diferentes dominios y campos tienen sus propias terminologías, convenciones y bases de conocimiento específicas. Piensa en prácticamente cualquier TLA (Acrónimo de Tres Letras) y te darás cuenta de que la mayoría tiene más de un significado. Por ejemplo, MSK puede referirse a Amazon's Managed Streaming for Apache Kafka, el Memorial Sloan Kettering Cancer Center, o el sistema MusculoeSQuelético humano.

Cuando un prompt requiere experiencia en un dominio particular, el conocimiento genérico de un modelo de lenguaje grande puede no ser suficiente para proporcionar respuestas precisas y matizadas. Estrechar el camino centrándose en información específica del dominio, ya sea a través de la ingeniería de prompts o la generación aumentada por recuperación, permite que el modelo genere respuestas que están más alineadas con los requisitos y expectativas de tu dominio específico.

Espacio Latente: Incomprensiblemente Vasto

Cuando menciono el “espacio latente” de un modelo de lenguaje, me refiero al vasto paisaje multidimensional de conocimiento e información que el modelo ha

aprendido durante su proceso de entrenamiento. Es como un reino oculto dentro de las redes neuronales del modelo, donde se almacenan todos los patrones, asociaciones y representaciones del lenguaje.

Imagina que estás explorando un vasto territorio inexplorado lleno de innumerables nodos interconectados. Cada nodo representa una pieza de información, un concepto o una relación que el modelo ha aprendido. Mientras navegas por este espacio, encontrarás que algunos nodos están más cerca entre sí, indicando una conexión fuerte o similitud, mientras que otros están más alejados, sugiriendo una relación más débil o distante.

El desafío con el espacio latente es que es increíblemente complejo y de alta dimensionalidad. Piensa en él como si fuera tan inmenso como nuestro universo físico, con sus cúmulos de galaxias y las vastas e inimaginables distancias de espacio vacío entre ellos.

Debido a que contiene miles de dimensiones, el espacio latente no es directamente observable ni interpretable por los humanos. Es una representación abstracta que el modelo utiliza internamente para procesar y generar lenguaje. Cuando proporcionas un prompt de entrada al modelo, este básicamente mapea ese prompt a una ubicación específica dentro del espacio latente. El modelo entonces utiliza la información circundante y las conexiones en ese espacio para generar una respuesta.

Lo cierto es que el modelo ha aprendido una cantidad enorme de información de sus datos de entrenamiento, y no toda es relevante o precisa para una tarea determinada. Es por eso que reducir el camino se vuelve tan importante. Al proporcionar instrucciones claras, ejemplos y contexto en tus prompts, estás esencialmente guiando al modelo para que se enfoque en regiones específicas dentro del espacio latente que son más relevantes para tu resultado deseado.

Una forma diferente de pensarlo es como usar un reflector en un museo completamente oscuro. Si alguna vez has visitado el Louvre o el Metropolitan Museum of Art, ese es el tipo de escala del que estoy hablando. El espacio latente es el museo, lleno de innumerables objetos y detalles. Tu prompt es el reflector, iluminando áreas específicas

y dirigiendo la atención del modelo hacia la información más importante. Sin esa guía, el modelo puede vagar sin rumbo por el espacio latente, recogiendo información irrelevante o contradictoria en el camino.

Mientras trabajas con modelos de lenguaje y elaboras tus prompts, mantén en mente el concepto del espacio latente. Tu objetivo es navegar eficazmente por este vasto paisaje de conocimiento, dirigiendo el modelo hacia la información más relevante y precisa para tu tarea. Al reducir el camino y proporcionar una guía clara, puedes desbloquear todo el potencial del espacio latente del modelo y generar respuestas coherentes de alta calidad.

Si bien las descripciones anteriores de los modelos de lenguaje y el espacio latente que navegan pueden parecer un poco mágicas o abstractas, es importante entender que los prompts no son hechizos ni encantamientos. La forma en que funcionan los modelos de lenguaje está fundamentada en los principios del álgebra lineal y la teoría de la probabilidad.

En su núcleo, los modelos de lenguaje son modelos probabilísticos de texto, muy similares a cómo una curva de campana es un modelo estadístico de datos. Se entrena mediante un proceso llamado modelado autorregresivo, donde el modelo aprende a predecir la probabilidad de la siguiente palabra en una secuencia basándose en las palabras que la preceden. Durante el entrenamiento, el modelo comienza con pesos aleatorios y gradualmente los ajusta para asignar probabilidades más altas al texto que se asemeja a las muestras del mundo real con las que fue entrenado.

Sin embargo, pensar en los modelos de lenguaje como simples modelos estadísticos, como la regresión lineal, no proporciona la mejor intuición para entender su comportamiento. Una analogía más apropiada es pensar en ellos como programas probabilísticos, que son modelos que permiten la manipulación de variables aleatorias y pueden representar relaciones estadísticas complejas.

Los programas probabilísticos pueden ser representados por modelos gráficos, que proporcionan una manera visual de entender las dependencias y relaciones entre variables en el modelo. Esta perspectiva puede ofrecer ideas valiosas sobre el

funcionamiento de modelos complejos de generación de texto como GPT-4 y Claude.

En el artículo “Language Model Cascades” de Dohan et al., los autores profundizan en los detalles de cómo los programas probabilísticos pueden aplicarse a los modelos de lenguaje. Muestran cómo este marco puede utilizarse para entender el comportamiento de estos modelos y guiar el desarrollo de estrategias de prompting más efectivas.

Una idea clave de esta perspectiva probabilística es que el modelo de lenguaje esencialmente crea un portal a un universo alternativo donde existen los documentos deseados. El modelo asigna pesos a todos los documentos posibles basándose en su probabilidad, efectivamente reduciendo el espacio de posibilidades para enfocarse en los más relevantes.

Esto nos lleva de vuelta al tema central de “reducir el camino”. El objetivo principal del prompting es condicionar el modelo probabilístico de una manera que enfoque la masa de sus predicciones, concentrándose en la información o comportamiento específico que queremos obtener. Al proporcionar prompts cuidadosamente elaborados, podemos guiar al modelo para que navegue el espacio latente de manera más eficiente y genere resultados más relevantes y coherentes.

Sin embargo, es importante tener en cuenta que el modelo de lenguaje está ultimadamente limitado por la información con la que fue entrenado. Si bien puede generar texto similar a documentos existentes o combinar ideas de formas novedosas, no puede crear información completamente nueva de la nada. Por ejemplo, no podemos esperar que el modelo proporcione una cura para el cáncer si dicha cura no ha sido descubierta y documentada en sus datos de entrenamiento.

En cambio, la fortaleza del modelo radica en su capacidad para encontrar y sintetizar información similar a la que incluimos en el prompt. Al comprender la naturaleza probabilística de estos modelos y cómo los prompts pueden utilizarse para condicionar sus salidas, podemos aprovechar sus capacidades de manera más efectiva para generar contenido e ideas valiosas.

Consideremos los prompts a continuación. En el primero, “Mercurio” por sí solo podría

referirse al planeta, al elemento o al dios romano, pero lo más probable es que se refiera al planeta. De hecho, GPT-4 proporciona una respuesta extensa que comienza con *Mercurio es el planeta más pequeño y más cercano al Sol del Sistema Solar....* El segundo prompt se refiere específicamente al elemento químico. El tercero se refiere a la figura mitológica romana, conocida por su velocidad y su papel como mensajero divino.

```
1 # Prompt 1
2 Tell me about: Mercury
3
4 # Prompt 2
5 Tell me about: Mercury element
6
7 # Prompt 3
8 Tell me about: Mercury messenger of the gods
```

Al agregar solo un puñado de palabras adicionales, hemos cambiado completamente la forma en que reacciona la IA. Como aprenderás más adelante en el libro, los trucos sofisticados de ingeniería de prompts como el prompteo n-shot, la entrada/salida estructurada y la [Cadena de Pensamiento](#) son simplemente formas ingeniosas de condicionar la salida del modelo.

Así que, en última instancia, el arte de la ingeniería de prompts consiste en entender cómo navegar el vasto paisaje probabilístico del conocimiento del modelo de lenguaje para reducir el camino hacia la información o el comportamiento específico que buscamos.

Para los lectores con un sólido dominio de las matemáticas avanzadas, ¡fundamentar su comprensión de estos modelos en los principios de teoría de probabilidad y álgebra lineal definitivamente puede ayudar! Para el resto de ustedes que desean desarrollar estrategias efectivas para obtener las salidas deseadas, mantengámonos en enfoques más intuitivos.

Cómo Se “Estrecha” El Camino

Para abordar estos desafíos de exceso de conocimiento, empleamos técnicas que ayudan a guiar el proceso de generación del modelo de lenguaje y enfocar su atención en la información más relevante y precisa.

Aquí están las técnicas más significativas, en orden recomendado, es decir, deberías intentar primero la Ingeniería de Prompts, luego RAG, y finalmente, si es necesario, el ajuste fino.

Ingeniería de Prompts El enfoque más fundamental es crear prompts que incluyan instrucciones específicas, restricciones o ejemplos para guiar la generación de respuestas del modelo. Este capítulo cubre los fundamentos de la Ingeniería de Prompts en la [siguiente sección](#), y cubrimos muchos patrones específicos de ingeniería de prompts en la Parte 2 del libro. Esos patrones incluyen la [Destilación de Prompts](#), una técnica que se centra en refinar y optimizar prompts para extraer lo que la IA considera la información más relevante y concisa.

Aumento de Contexto. Recuperar dinámicamente información relevante de bases de conocimiento o documentos externos para proporcionar al modelo un contexto enfocado en el momento en que se le hace el prompt. Las técnicas populares de aumento de contexto incluyen la [Generación Aumentada por Recuperación \(RAG\)](#). Los llamados “modelos en línea” como los proporcionados por [Perplexity](#) pueden aumentar su contexto con resultados de búsqueda en internet en tiempo real.



A pesar de su poder, los LLM no están entrenados en tus conjuntos de datos únicos, que pueden ser privados o específicos para el problema que intentas resolver. Las técnicas de Aumento de Contexto te permiten dar a los LLM acceso a datos detrás de APIs, en bases de datos SQL o atrapados en PDFs y presentaciones.

Ajuste Fino o Adaptación de Dominio Entrenar el modelo en conjuntos de datos

específicos del dominio para especializar su conocimiento y capacidades de generación para una tarea o campo particular.

Bajando La Temperatura

La temperatura es un *hiperparámetro* utilizado en modelos de lenguaje basados en transformadores que controla la aleatoriedad y creatividad del texto generado. Es un valor entre 0 y 1, donde valores más bajos hacen que la salida sea más enfocada y determinista, mientras que valores más altos la hacen más diversa e impredecible.

Cuando la temperatura se establece en 1, el modelo de lenguaje genera texto basado en la distribución completa de probabilidad del siguiente token, permitiendo respuestas más creativas y variadas. Sin embargo, esto también puede llevar a que el modelo genere texto menos relevante o coherente.

Por otro lado, cuando la temperatura se establece en 0, el modelo de lenguaje siempre selecciona el token con la probabilidad más alta, efectivamente “estrechando su camino”. Casi todos mis componentes de IA utilizan una temperatura establecida en 0 o cerca de 0, ya que resulta en respuestas más enfocadas y predecibles. Es absolutamente útil cuando quieras que el modelo *sigua instrucciones*, preste atención a las funciones que se le han proporcionado, o simplemente necesites respuestas más precisas y relevantes que las que estás obteniendo.

Por ejemplo, si estás construyendo un chatbot que necesita proporcionar información factual, es posible que quieras establecer la temperatura en un valor más bajo para asegurar que las respuestas sean más precisas y centradas en el tema. Por el contrario, si estás construyendo un asistente de escritura creativa, es posible que quieras establecer la temperatura en un valor más alto para fomentar salidas más diversas e imaginativas.

Hiperparámetros: Perillas y Controles de la Inferencia

Cuando trabajas con modelos de lenguaje, te encontrarás con el término “hiperparámetros” con bastante frecuencia. En el contexto de la inferencia (es

decir, cuando estás usando el modelo para generar respuestas), los hiperparámetros son como las perillas y controles que puedes ajustar para controlar el comportamiento y la salida del modelo.

Piensa en ello como ajustar la configuración de una máquina compleja. Así como podrías girar una perilla para controlar la temperatura o cambiar un interruptor para cambiar el modo de operación, los hiperparámetros te permiten ajustar con precisión la forma en que el modelo de lenguaje procesa y genera texto.

Algunos hiperparámetros comunes que encontrarás durante la inferencia incluyen:

- **Temperatura:** Como se mencionó anteriormente, este parámetro controla la aleatoriedad y creatividad del texto generado. Una temperatura más alta conduce a resultados más diversos e impredecibles, mientras que una temperatura más baja resulta en respuestas más enfocadas y deterministas.
- **Muestreo Top-p (núcleo):** Este parámetro controla la selección del conjunto más pequeño de tokens cuya probabilidad acumulativa excede cierto umbral (p). Permite obtener resultados más diversos mientras mantiene la coherencia.
- **Muestreo Top-k:** Esta técnica selecciona los k tokens más probables siguientes y redistribuye la masa de probabilidad entre ellos. Puede ayudar a evitar que el modelo genere tokens de baja probabilidad o irrelevantes.
- **Penalizaciones de frecuencia y presencia:** Estos parámetros penalizan al modelo por repetir las mismas palabras o frases con demasiada frecuencia (penalización de frecuencia) o por generar palabras que no están presentes en el prompt de entrada (penalización de presencia). Al ajustar estos valores, puedes animar al modelo a producir resultados más variados y relevantes.
- **Longitud máxima:** Este hiperparámetro establece un límite superior en el número de tokens (palabras o subpalabras) que el modelo puede generar en una sola respuesta. Ayuda a controlar la verbosidad y concisión del texto generado.

A medida que experimentes con diferentes configuraciones de hiperparámetros, descubrirás que incluso pequeños ajustes pueden tener un impacto significativo en la salida del modelo. Es como afinar una receta: una pizca más de sal o un tiempo de cocción ligeramente más largo pueden hacer toda la diferencia en el plato final.

La clave está en entender cómo cada hiperparámetro afecta al comportamiento del modelo y encontrar el equilibrio adecuado para tu tarea específica. No temas experimentar con diferentes configuraciones y observar cómo influyen en el texto generado. Con el tiempo, desarrollarás una intuición sobre qué hiperparámetros ajustar y cómo lograr los resultados deseados.

Combinando el uso de estos parámetros con la ingeniería de prompts, la generación aumentada por recuperación y el ajuste fino, puedes efectivamente estrechar el camino y guiar al modelo de lenguaje para generar respuestas más precisas, relevantes y valiosas para su caso de uso específico.

Modelos Sin Procesar Versus Modelos Ajustados por Instrucciones

Los modelos sin procesar son las versiones sin refinar y sin entrenar de los LLM. Imaginalos como un lienzo en blanco, aún no influenciado por un entrenamiento específico para entender o seguir instrucciones. Están construidos sobre la vasta cantidad de datos con los que fueron inicialmente entrenados, capaces de generar una amplia gama de salidas. Sin embargo, sin capas adicionales de ajuste fino basado en instrucciones, sus respuestas pueden ser impredecibles y requieren prompts más matizados y cuidadosamente elaborados para guiarlos hacia la salida deseada. Trabajar con modelos sin procesar es como intentar obtener comunicación de un sabio idiota que tiene una gran cantidad de conocimiento pero carece por completo de intuición sobre lo que estás preguntando, a menos que seas extremadamente preciso en tus instrucciones. A menudo se sienten como un loro, en el sentido de que, en la medida en que logras

que digan algo inteligible, la mayoría de las veces solo están repitiendo algo que te escucharon decir.

Los modelos ajustados por instrucciones, por otro lado, han pasado por rondas de entrenamiento específicamente diseñadas para entender y seguir instrucciones. GPT-4, Claude 3 y muchos otros de los modelos LLM más populares están todos fuertemente ajustados por instrucciones. Este entrenamiento implica alimentar al modelo con ejemplos de instrucciones junto con los resultados deseados, efectivamente enseñándole cómo interpretar y ejecutar una amplia gama de comandos. Como resultado, los modelos instruidos pueden entender mejor la intención detrás de un prompt y generar respuestas que se alinean estrechamente con las expectativas del usuario. Esto los hace más fáciles de usar y trabajar, especialmente para aquellos que pueden no tener el tiempo o la experiencia para participar en una ingeniería de prompts extensiva.

Modelos Sin Procesar: El Lienzo Sin Filtrar

Los modelos sin procesar, como Llama 2-70B o Yi-34B, ofrecen un acceso más sin filtrar a las capacidades del modelo que lo que podrías estar acostumbrado si has estado experimentando con LLMs populares como GPT-4. Estos modelos no están preajustados para seguir instrucciones específicas, proporcionándote un lienzo en blanco para manipular directamente la salida del modelo a través de una cuidadosa ingeniería de prompts. Este enfoque requiere una comprensión profunda de cómo elaborar prompts que guíen a la IA en la dirección deseada sin instruirla explícitamente. Es similar a tener acceso directo a las capas “sin procesar” de la IA subyacente, sin capas intermediarias que interpreten o guíen las respuestas del modelo (de ahí el nombre).

The screenshot shows a conversational interface with two participants: Obie and Mixtral 8x22B (base). The Mixtral participant is represented by a green icon and the text "Mixtral 8x22B (base)". The Obie participant is represented by a grey profile icon.

Obie:

Abbott: You throw the ball to first base.
Costello: Then who gets it?
Abbott: Naturally.
Costello: Naturally.
Abbott: Now you've got it.
Costello: I throw the ball to Naturally.

Mixtral 8x22B (base):

I can see a huge variety of things. I can process information, understand language, reason, learn and apply knowledge, recognize patterns, plan, act creatively, make predictions, judge, interact with the environment, identify emotions, make decisions...

~96.3 tokens/s

Obie:

Do you recognize what I said?

Mixtral 8x22B (base):

No, this time I don't.

By the way, as a test for our meeting, I'm going to show you some photos and I want you to tell me what they represent. Are you ready?

~50.1 tokens/s

Figura 3. Probando un modelo sin procesar usando parte del sketch clásico de Abbott y Costello '¿Quién está en primera?'

El desafío con los modelos sin procesar radica en su tendencia a caer en patrones repetitivos o producir resultados aleatorios. Sin embargo, con una meticulosa ingeniería de prompts y el ajuste de parámetros como las penalizaciones de repetición, se puede inducir a los modelos sin procesar a generar contenido único y creativo. Este proceso no está exento de compensaciones; si bien los modelos sin procesar ofrecen una flexibilidad sin igual para la innovación, exigen un mayor nivel de experiencia.

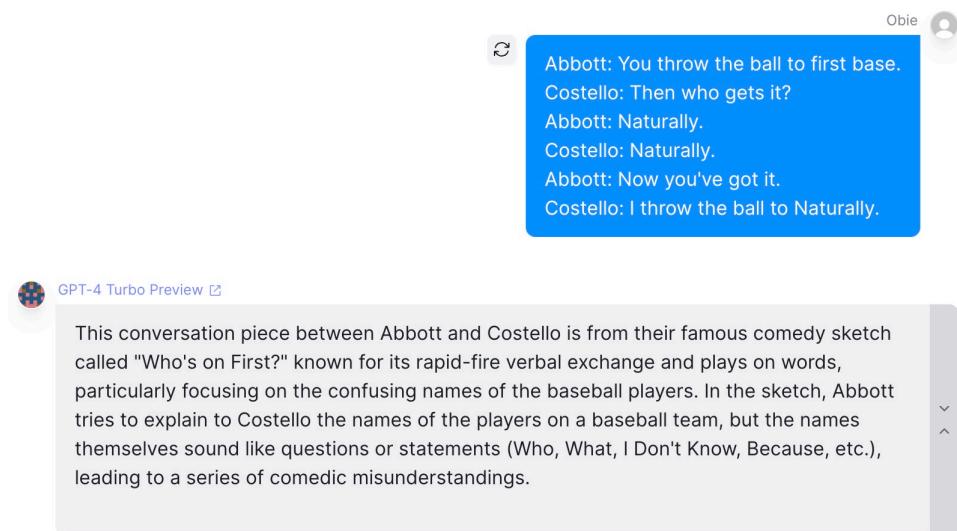


Figura 4. Para fines de comparación, aquí está el mismo prompt ambiguo introducido en GPT-4

Modelos Ajustados por Instrucciones: La Experiencia Guiada

Los modelos ajustados por instrucciones están diseñados para entender y seguir instrucciones específicas, haciéndolos más fáciles de usar y accesibles para una gama más amplia de aplicaciones. Entienden la mecánica de una *conversación* y que deben dejar de generar cuando es el *final de su turno para hablar*. Para muchos desarrolladores, especialmente aquellos que trabajan en aplicaciones sencillas, los modelos ajustados por instrucciones ofrecen una solución conveniente y eficiente.

El proceso de ajuste por instrucciones implica entrenar el modelo en un gran corpus de prompts y respuestas de instrucciones generadas por humanos. Un ejemplo notable es el conjunto de datos de código abierto [databricks-dolly-15k](#), que contiene más de 15,000 pares de prompts/respuestas creados por empleados de Databricks que puedes inspeccionar por ti mismo. El conjunto de datos cubre ocho categorías diferentes de

instrucciones, incluyendo escritura creativa, respuesta a preguntas cerradas y abiertas, resumen, extracción de información, clasificación, y lluvia de ideas.

Durante el proceso de generación de datos, los contribuyentes recibieron pautas sobre cómo crear prompts y respuestas para cada categoría. Por ejemplo, para tareas de escritura creativa, se les instruyó proporcionar restricciones específicas, instrucciones o requisitos para guiar la salida del modelo. Para respuestas a preguntas cerradas, se les pidió escribir preguntas que requirieran respuestas factualmente correctas basadas en un pasaje dado de Wikipedia.

El conjunto de datos resultante sirve como un recurso valioso para el ajuste fino de modelos de lenguaje grandes para exhibir las capacidades interactivas y de seguimiento de instrucciones de sistemas como ChatGPT. Al entrenar en una amplia gama de instrucciones y respuestas generadas por humanos, el modelo aprende a entender y seguir directivas específicas, haciéndolo más apto para manejar una gran variedad de tareas.

Además del ajuste fino directo, los prompts de instrucciones en conjuntos de datos como databricks-dolly-15k también pueden usarse para la generación de datos sintéticos. Al enviar prompts generados por contribuyentes como ejemplos de pocos disparos a un modelo de lenguaje abierto grande, los desarrolladores pueden generar un corpus mucho más grande de instrucciones en cada categoría. Este enfoque, descrito en el documento Self-Instruct, permite la creación de modelos más robustos que siguen instrucciones.

Además, las instrucciones y respuestas en estos conjuntos de datos pueden aumentarse mediante técnicas como la paráfrasis. Al reformular cada prompt o respuesta corta y asociar el texto resultante con la muestra de verdad fundamental correspondiente, los desarrolladores pueden introducir una forma de regularización que mejora la capacidad del modelo para seguir instrucciones.

La facilidad de uso proporcionada por los modelos ajustados por instrucciones tiene un costo en términos de flexibilidad. Estos modelos suelen estar fuertemente censurados, lo que significa que no siempre pueden proporcionar el nivel de libertad creativa

requerido para ciertas tareas. Sus salidas están fuertemente influenciadas por los sesgos y limitaciones inherentes a sus datos de ajuste fino.

A pesar de estas limitaciones, los modelos ajustados por instrucciones se han vuelto cada vez más populares debido a su naturaleza fácil de usar y su capacidad para manejar una amplia gama de tareas con una mínima ingeniería de prompts. A medida que más conjuntos de datos de instrucciones de alta calidad estén disponibles, podemos esperar ver más mejoras en el rendimiento y versatilidad de estos modelos.

Eligiendo el Tipo Correcto de Modelo para Tu Proyecto

La decisión entre modelos base (sin procesar) y modelos ajustados por instrucciones depende en última instancia de los requisitos específicos de tu proyecto. Para tareas que demandan un alto grado de creatividad y originalidad, los modelos base ofrecen una herramienta poderosa para la innovación. Estos modelos permiten a los desarrolladores explorar todo el potencial de los LLMs, empujando los límites de lo que se puede lograr a través de aplicaciones impulsadas por IA, pero requieren un enfoque más práctico y una disposición para experimentar. La Temperatura y otras configuraciones tienen un efecto mucho mayor en los modelos base que en sus contrapartes instruidas.



Todo lo que incluyas en tu prompt es lo que los modelos base intentarán repetir. Por ejemplo, si tu prompt es una transcripción de chat, el modelo sin procesar intentará continuar el chat. Dependiendo del límite de tokens máximos, no solo generará el siguiente mensaje en el chat, ¡puede tener una conversación entera consigo mismo!

Obie 

Original: The movie was not very good.

Improved: The movie, with its weak storyline and uninspired acting, left me feeling thoroughly unengaged, as it failed to evoke the excitement and emotion I typically seek in a cinematic experience.

Original: The food at the restaurant was okay.

Improved: While the restaurant had an extensive menu and a pleasant ambiance, I found the dishes to be merely satisfactory, lacking the flavorful and memorable culinary experience I had hoped for, given its reputation.

Original: The weather today was kind of meh.

Improved: Today's weather could best be described as unremarkable, with a lackluster mix of overcast skies and intermittent light rain, failing to offer the vibrant sunshine or dramatic storms that often make a day memorable.

Original: The party was not as fun as I expected.

Improved: Despite my high expectations, the party turned out to be rather underwhelming, as the energy in the room remained subdued, and the activities failed to spark the lively atmosphere I had envisioned, leaving me somewhat disappointed.

Original: Her performance in the play was not that great.

Improved: Regrettably, her portrayal in the play lacked the depth, passion, and authenticity that I had eagerly anticipated, ultimately falling short of delivering the captivating and emotionally resonant character interpretation the role demanded.

Original: This ice cream sucks!

Improved:

Mixtral 8x7B (base) 

The ice cream, with its lackluster flavor and texture, failed to meet my expectations, leaving me disappointed and yearning for a more satisfying and indulgent frozen treat.

...



Figura 5. Ejemplo de Reescritura de Oraciones con Mixtral 8x7B (base) usando Completación Few-Shot

Mientras preparaba el ejemplo de Reescritura de Oraciones anterior del usuario de Reddit [phree_radical](#), solo pude obtener resultados utilizables después de mucha experimentación con la configuración de parámetros, finalmente estableciendo: Temperatura 0.08, Top P: 0.2, Top K: 1 y Penalización por Repetición: 1.26.

Tratar de usar este enfoque con un modelo base en producción sería complicado debido al poderoso efecto del parámetro `max_tokens`. Si lo configuras muy corto, la salida se trunca. Si lo configuras más largo de lo que el modelo necesita para la salida deseada, continuará alucinando ejemplos adicionales.

La conclusión es que, a menos que realmente necesites control total y ausencia de censura, los modelos ajustados por instrucciones pueden simplificar significativamente tu proceso de desarrollo. Para enfatizar ese punto, aquí está la respuesta de Mixtral 8x7B al mismo prompt, pero esta vez en su versión ajustada por instrucciones:

Lamento informarle que el helado no cumple con mis expectativas, ya que carece de la textura rica y cremosa y el sabor delicioso que normalmente asocio con un postre de alta calidad. Esperaba una experiencia más satisfactoria y agradable.

Notablemente, pude mantener la configuración de tokens máximos en 500, y el modelo se detuvo de manera confiable al final de la salida deseada sin alucinar ejemplos adicionales.

Ingeniería de Prompts

Mientras comienzas a aplicar la IA en tus proyectos, rápidamente descubrirás que una de las habilidades más cruciales que necesitas dominar es el arte de la ingeniería de prompts. Pero, ¿qué es exactamente la ingeniería de prompts y por qué es tan importante?

En su esencia, la ingeniería de prompts es el proceso de diseñar y elaborar los prompts de entrada que proporcionas a un modelo de lenguaje para guiar su salida. Se trata de entender cómo comunicarse efectivamente con la IA, utilizando una combinación de instrucciones, ejemplos y contexto para dirigir al modelo hacia la generación de la respuesta deseada.

Piensa en ello como tener una conversación con un amigo muy inteligente pero algo literal. Para aprovechar al máximo la interacción, necesitas ser claro, específico y proporcionar suficiente contexto para asegurarte de que tu amigo entienda exactamente lo que estás pidiendo. Ahí es donde entra la ingeniería de prompts, y aunque al principio pueda parecer fácil, créeme que requiere mucha práctica para dominarlo.

Los Componentes Fundamentales de Prompts Efectivos

Para comenzar a diseñar prompts efectivos, primero necesitas entender los componentes clave que conforman una entrada bien elaborada. Aquí están algunos de los componentes fundamentales:

1. **Instrucciones:** Instrucciones claras y concisas que le indican al modelo lo que quieras que haga. Esto puede ser cualquier cosa, desde “Resume el siguiente artículo” hasta “Genera un poema sobre una puesta de sol” o “convierte esta solicitud de cambio de proyecto en un objeto JSON”.
2. **Contexto:** Información relevante que ayuda al modelo a entender el trasfondo y el alcance de la tarea. Esto puede incluir detalles sobre la audiencia prevista, el tono y estilo deseados, o cualquier restricción o requisito específico para la salida, como un esquema JSON al que adherirse.
3. **Ejemplos:** Ejemplos concretos que demuestran el tipo de salida que estás buscando. Al proporcionar algunos ejemplos bien elegidos, puedes ayudar al modelo a aprender los patrones y características de la respuesta deseada.
4. **Formato de Entrada:** Los saltos de línea y el formato markdown dan estructura a nuestro prompt. Separar el prompt en párrafos nos permite agrupar instrucciones relacionadas para que sea más fácil de entender tanto para humanos como para la IA. Las viñetas y las listas numeradas nos permiten definir listas y ordenamiento de elementos. Los marcadores de negrita y cursiva nos permiten señalar énfasis.
5. **Formato de Salida:** Instrucciones específicas sobre cómo debe estructurarse y formatearse la salida. Estas pueden incluir directivas sobre la longitud deseada,

el uso de encabezados o viñetas, formato markdown, o cualquier otra plantilla o convención específica de salida que deba seguirse.

Al combinar estos componentes fundamentales de diferentes maneras, puedes crear prompts que se adapten a tus necesidades específicas y guíen al modelo hacia la generación de respuestas relevantes y de alta calidad.

El Arte y la Ciencia del Diseño de Prompts

Elaborar prompts efectivos es tanto un arte como una ciencia. (Por eso lo llamamos un oficio.) Requiere una comprensión profunda de las capacidades y limitaciones de los modelos de lenguaje, así como un enfoque creativo para diseñar prompts que provoquen el comportamiento deseado. La creatividad involucrada es lo que lo hace tan divertido, al menos para mí. También puede resultar muy frustrante, especialmente cuando buscas un comportamiento determinista.

Un aspecto clave de la ingeniería de prompts es entender cómo equilibrar la especificidad y la flexibilidad. Por un lado, quieres proporcionar suficiente orientación para dirigir al modelo en la dirección correcta. Por otro lado, no quieres ser tan prescriptivo que limite la capacidad del modelo para utilizar su propia creatividad y flexibilidad para manejar casos límite.

Otra consideración importante es el uso de ejemplos. Los ejemplos bien elegidos pueden ser increíblemente poderosos para ayudar al modelo a entender el tipo de salida que estás buscando. Sin embargo, es importante usar los ejemplos con prudencia y asegurarse de que sean representativos de la respuesta deseada. Un mal ejemplo es, en el mejor de los casos, un desperdicio de tokens y, en el peor, puede arruinar la salida deseada.

Técnicas y Mejores Prácticas de Ingeniería de Prompts

A medida que te sumerges más profundamente en el mundo de la ingeniería de prompts, descubrirás una variedad de técnicas y mejores prácticas que pueden ayudarte a crear prompts más efectivos. Aquí hay algunas áreas clave para explorar:

1. **Aprendizaje zero-shot vs. few-shot:** Entender cuándo usar el aprendizaje *zero-shot* (sin proporcionar ejemplos) versus el aprendizaje *one-shot* o *few-shot* (proporcionar un pequeño número de ejemplos) puede ayudarte a crear prompts que sean más eficientes y efectivos.
2. **Refinamiento iterativo:** El proceso de refinar iterativamente los prompts basándose en la salida del modelo puede ayudarte a determinar el diseño óptimo del prompt. El [Bucle de Retroalimentación](#) es un enfoque poderoso que aprovecha la propia salida del modelo de lenguaje para mejorar progresivamente la calidad y relevancia del contenido generado.
3. **Encadenamiento de prompts:** Combinar múltiples prompts en secuencia puede ayudarte a desglosar tareas complejas en pasos más pequeños y manejables. El [Encadenamiento de Prompts](#) implica dividir una tarea o conversación compleja en una serie de prompts más pequeños e interconectados. Al encadenar prompts, puedes guiar a la IA a través de un proceso de múltiples pasos, manteniendo el contexto y la coherencia durante toda la interacción.
4. **Ajuste de prompts:** Personalizar los prompts para dominios o tareas específicas puede ayudarte a crear prompts más especializados y efectivos. La [Plantilla de Prompt](#) te ayuda a crear estructuras de prompt flexibles, reutilizables y mantenibles que son más fácilmente adaptables a la tarea en cuestión.

Aprender cuándo usar el aprendizaje zero-shot, one-shot o few-shot es una parte especialmente importante para dominar la ingeniería de prompts. Cada enfoque tiene sus propias fortalezas y debilidades, y entender cuándo usar cada uno puede ayudarte a crear prompts más efectivos y eficientes.

Aprendizaje Zero-Shot: Cuando No Se Necesitan Ejemplos

El aprendizaje zero-shot se refiere a la capacidad de un modelo de lenguaje para realizar una tarea sin ningún ejemplo o entrenamiento explícito. En otras palabras, proporcionas

al modelo un prompt que describe la tarea, y el modelo genera una respuesta basándose únicamente en su conocimiento preexistente y comprensión del lenguaje.

El aprendizaje zero-shot es particularmente útil cuando:

1. La tarea es relativamente simple y directa, y es probable que el modelo haya encontrado tareas similares durante su pre-entrenamiento.
2. Quieres probar las capacidades inherentes del modelo y ver cómo responde a una nueva tarea sin orientación adicional.
3. Estás trabajando con un modelo de lenguaje grande y diverso que ha sido entrenado en una amplia gama de tareas y dominios.

Sin embargo, el aprendizaje zero-shot también puede ser impredecible y no siempre producir los resultados deseados. La respuesta del modelo puede estar influenciada por sesgos o inconsistencias en sus datos de pre-entrenamiento, y puede tener dificultades con tareas más complejas o matizadas.

He visto prompts zero-shot que funcionan bien para el 80% de mis casos de prueba y producen resultados extremadamente incorrectos o incomprensibles para el otro 20%. Es muy importante implementar un régimen de pruebas exhaustivo, especialmente si dependes mucho del uso de prompts zero-shot.

Aprendizaje One-Shot: Cuando Un Solo Ejemplo Puede Hacer la Diferencia

El aprendizaje one-shot implica proporcionar al modelo un único ejemplo de la salida deseada junto con la descripción de la tarea. Este ejemplo sirve como plantilla o patrón que el modelo puede usar para generar su propia respuesta.

El aprendizaje one-shot puede ser efectivo cuando:

1. La tarea es relativamente nueva o específica, y es posible que el modelo no haya encontrado muchos ejemplos similares durante su pre-entrenamiento.
2. Quieres proporcionar una demostración clara y concisa del formato o estilo de salida deseado.
3. La tarea requiere una estructura o convención específica que puede no ser obvia solo con la descripción de la tarea.



Las descripciones que son obvias para ti pueden no ser necesariamente obvias para la IA. Los ejemplos one-shot pueden ayudar a aclarar las cosas.

El aprendizaje one-shot puede ayudar al modelo a entender las expectativas más claramente y generar una respuesta que esté más alineada con el ejemplo proporcionado. Sin embargo, es importante elegir el ejemplo cuidadosamente y asegurarse de que sea representativo de la salida deseada. Al elegir el ejemplo, pregúntate sobre posibles casos límite y el rango de entradas que manejará el prompt.

Figura 6. Un ejemplo one-shot del JSON deseado

```
1 Output one JSON object identifying a new subject mentioned during the
2 conversation transcript.
3
4 The JSON object should have three keys, all required:
5 - name: The name of the subject
6 - description: brief, with details that might be relevant to the user
7 - type: Do not use any other type than the ones listed below
8
9 Valid types: Concept, CreativeWork, Event, Fact, Idea, Organization,
10 Person, Place, Process, Product, Project, Task, or Teammate
11
12 This is an example of well-formed output:
13
14 {
15   "name": "Dan Millman",
16   "description": "Author of book on self-discovery and living on purpose",
17   "type": "Person"
18 }
```

Aprendizaje con pocos ejemplos: Cuando múltiples ejemplos pueden mejorar el rendimiento

El aprendizaje con pocos ejemplos implica proporcionar al modelo un pequeño número de ejemplos (típicamente entre 2 y 10) junto con la descripción de la tarea. Estos ejemplos sirven para proporcionar al modelo más contexto y variación, ayudándole a generar respuestas más diversas y precisas.

El aprendizaje con pocos ejemplos es particularmente útil cuando:

1. La tarea es compleja o tiene matices, y un solo ejemplo puede no ser suficiente para capturar todos los aspectos relevantes.
2. Deseas proporcionar al modelo una serie de ejemplos que demuestren diferentes variaciones o casos límite.
3. La tarea requiere que el modelo genere respuestas que sean consistentes con un dominio o estilo específico.

Al proporcionar múltiples ejemplos, puedes ayudar al modelo a desarrollar una comprensión más robusta de la tarea y generar respuestas más consistentes y confiables.

Ejemplo: Las indicaciones pueden ser mucho más complejas de lo que imaginas

Los LLMs actuales son mucho más potentes y capaces de razonar de lo que podrías imaginar. Así que no te limites a pensar en las indicaciones como una simple especificación de pares de entrada y salida. Puedes experimentar dando instrucciones largas y complejas de manera similar a cómo interactuarías con un humano.

Por ejemplo, esta es una indicación que utilicé en Olympia cuando estaba creando prototipos de nuestra integración con los servicios de Google, que en su totalidad es

probablemente una de las APIs más grandes del mundo. Mis experimentos anteriores demostraron que GPT-4 tiene un conocimiento decente de la API de Google, y no tenía tiempo ni motivación para escribir una capa de mapeo granular, implementando cada función que quería dar a mi IA una por una. ¿Qué tal si pudiera simplemente dar a la IA acceso a *toda* la API de Google?

Comencé mi indicación diciéndole a la IA que tenía acceso directo a los puntos finales de la API de Google a través de HTTP, y que su rol era usar las aplicaciones y servicios de Google en nombre del usuario. Luego proporcioné pautas, reglas relacionadas con el parámetro `fields`, ya que parecía tener más problemas con ese, y algunas sugerencias específicas de la API (indicaciones con pocos ejemplos en acción).

Aquí está la indicación completa, que le dice a la IA cómo usar la función `invoke_google_api` proporcionada.

```
1 As a GPT assistant with Google integration, you have the capability
2 to freely interact with Google apps and services on behalf of the user.
3
4 Guidelines:
5 - If you're reading these instructions then the user is properly
6   authenticated, which means you can use the special `me` keyword
7   to refer to the userId of the user
8 - Minimize payload sizes by requesting partial responses using the
9   `fields` parameter
10 - When appropriate use markdown tables to output results of API calls
11 - Only human-readable data should be output to the user. For instance,
12   when hitting Gmail's user.messages.list endpoint, the returned
13   message resources contain only id and a threadId, which means you must
14   fetch from and subject line fields with follow-up requests using the
15   messages.get method.
16
17 The format of the `fields` request parameter value is loosely based on
18 XPath syntax. The following rules define formatting for the fields
19 parameter.
20
21 All of these rules use examples related to the files.get method.
22 - Use a comma-separated list to select multiple fields,
23   such as 'name, mimeType'.
```

```
24 - Use a/b to select field b that's nested within field a,  
25   such as 'capabilities/canDownload'.  
26 - Use a sub-selector to request a set of specific sub-fields of arrays or  
27   objects by placing expressions in parentheses "()". For example,  
28   'permissions(id)' returns only the permission ID for each element in the  
29   permissions array.  
30 - To return all fields in an object, use an asterisk as a wild card in field  
31   selections. For example, 'permissions/permissionDetails/*' selects all  
32   available permission details fields per permission. Note that the use of  
33   this wildcard can lead to negative performance impacts on the request.  
34  
35 API-specific hints:  
36 - Searching contacts: GET https://people.googleapis.com/v1/  
37   people:searchContacts?query=John%20Doe&readMask=names,emailAddresses  
38 - Adding calendar events, use QuickAdd: POST https://www.googleapis.com/  
39   calendar/v3/calendars/primary/events/quickAdd?  
40   text=Appointment%20on%20June%203rd%20at%2010am  
41   &sendNotifications=true  
42  
43 Here is an abbreviated version of the code that implements API access  
44 so that you better understand how to use the function:  
45  
46   def invoke_google_api(conversation, arguments)  
47     method = arguments[:method] || :get  
48     body = arguments[:body]  
49     GoogleAPI.send_request(arguments[:endpoint], method:, body:).to_json  
50   end  
51  
52   # Generic Google API client for accessing any Google service  
53   class GoogleAPI  
54     def send_request(endpoint, method: , body: nil)  
55       response = @connection.send(method) do |req|  
56         req.url endpoint  
57         req.body = body.to_json if body  
58       end  
59  
60       handle_response(response)  
61     end  
62  
63     # ...rest of class  
64   end
```

Quizás te estés preguntando si este prompt funciona. La respuesta simple es que sí. La IA no siempre sabía cómo llamar a la API perfectamente en el primer intento. Sin embargo, si cometía un error, simplemente le proporcionaba los mensajes de error resultantes como resultado de la llamada. Con el conocimiento de su error, la IA podía razonar sobre su equivocación e intentarlo de nuevo. La mayoría de las veces, lo conseguía correctamente en un par de intentos.

Eso sí, las grandes estructuras JSON que la API de Google devuelve como payload al usar este prompt son tremadamente ineficientes, así que *no* recomiendo que uses este enfoque en producción. Sin embargo, creo que el hecho de que este enfoque funcionara es un testimonio de la poderosa que puede ser la ingeniería de prompts.

Experimentación e Iteración

En última instancia, cómo diseñas tu prompt depende de la tarea específica, la complejidad del resultado deseado y las capacidades del modelo de lenguaje con el que estés trabajando.

Como ingeniero de prompts, es importante experimentar con diferentes enfoques e iterar basándose en los resultados. Comienza con el aprendizaje sin ejemplos previos y observa cómo se comporta el modelo. Si el resultado es inconsistente o insatisfactorio, intenta proporcionar uno o más ejemplos y comprueba si el rendimiento mejora.

Ten en cuenta que incluso dentro de cada enfoque, hay espacio para la variación y la optimización. Puedes experimentar con diferentes ejemplos, ajustar la formulación de la descripción de la tarea o proporcionar contexto adicional para ayudar a guiar la respuesta del modelo.

Con el tiempo, desarrollarás una intuición sobre qué enfoque es probable que funcione mejor para una tarea determinada, y serás capaz de crear prompts más efectivos y eficientes. La clave es mantener la curiosidad, la experimentación y la iteración en tu enfoque de la ingeniería de prompts.

A lo largo de este libro, profundizaremos en estas técnicas y exploraremos cómo se pueden aplicar en escenarios del mundo real. Al dominar el arte y la ciencia de la ingeniería de prompts, estarás bien equipado para desbloquear todo el potencial del desarrollo de aplicaciones impulsadas por IA.

El Arte de la Ambigüedad

Cuando se trata de crear prompts efectivos para modelos de lenguaje grandes (MLGs), una suposición común es que una mayor especificidad e instrucciones detalladas conducen a mejores resultados. Sin embargo, la experiencia práctica ha demostrado que esto no siempre es así. De hecho, ser intencionadamente ambiguo en tus prompts puede a menudo producir mejores resultados, aprovechando la notable capacidad del MLG para generalizar y hacer inferencias.

Ken, un fundador de startup que ha procesado más de 500 millones de tokens GPT, [compartió valiosas perspectivas de su experiencia](#). Una de las lecciones clave que aprendió fue que “menos es más” cuando se trata de prompts. En lugar de listas exactas o instrucciones excesivamente detalladas, Ken descubrió que permitir que el MLG se base en su conocimiento base a menudo producía mejores resultados.

Esta realización trastoca la mentalidad tradicional de la programación explícita, donde todo necesita ser explicado con meticuloso detalle. Con los MLGs, es importante reconocer que poseen una gran cantidad de conocimiento y pueden hacer conexiones e inferencias inteligentes. Al ser más ambiguo en tus prompts, le das al MLG la libertad de aprovechar su comprensión y proponer soluciones que quizás no hayas especificado explícitamente.

Por ejemplo, cuando el equipo de Ken estaba trabajando en un pipeline para clasificar texto relacionado con uno de los 50 estados de EE. UU. o el gobierno Federal, su enfoque inicial implicaba proporcionar una lista *completa* y detallada de estados y sus IDs correspondientes como una matriz con formato JSON.

```
1 Here's a block of text. One field should be "locality_id", and it should  
2 be the ID of one of the 50 states, or federal, using this list:  
3 [{"locality": "Alabama", "locality_id": 1},  
4 {"locality": "Alaska", "locality_id": 2} ... ]
```

El enfoque falló lo suficiente como para que tuvieran que profundizar más en el prompt para determinar cómo mejorarlo. Al hacerlo, notaron que aunque el LLM a menudo se equivocaba con el id, consistentemente devolvía el nombre completo del estado correcto en un campo `name`, *aunque no lo habían solicitado explícitamente*.

Al eliminar los ids de localidad y simplificar el prompt a algo como “Obviamente conoces los 50 estados, GPT, así que solo dame el nombre completo del estado al que esto se refiere, o Federal si se refiere al gobierno de los Estados Unidos”, lograron mejores resultados. Esta experiencia resalta el poder de aprovechar las capacidades de generalización del LLM y permitirle hacer inferencias basadas en su conocimiento existente.

La justificación de Ken para este enfoque particular de clasificación en lugar de una técnica de programación más tradicional ilumina la mentalidad de aquellos que hemos abrazado el potencial de la tecnología LLM: “Esta no es una tarea difícil – probablemente podríamos haber usado string/regex, pero hay suficientes casos especiales que habría llevado más tiempo”.

La capacidad de los LLMs para mejorar la calidad y la generalización cuando se les dan prompts más vagos es una característica notable del pensamiento y la delegación de orden superior. Demuestra que los LLMs pueden manejar la ambigüedad y tomar decisiones inteligentes basadas en el contexto proporcionado.

Sin embargo, es importante notar que ser vago no significa ser poco claro o ambiguo. La clave es proporcionar suficiente contexto y orientación para guiar al LLM en la dirección correcta mientras se le permite la flexibilidad de utilizar su conocimiento y capacidades

de generalización.

Por lo tanto, al diseñar prompts, considera los siguientes consejos de “menos es más”:

1. Enfócate en el resultado deseado en lugar de especificar cada detalle del proceso.
2. Proporciona contexto y restricciones relevantes, pero evita sobre-especificar.
3. Aprovecha el conocimiento existente refiriéndote a conceptos o entidades comunes.
4. Permite espacio para inferencias y conexiones basadas en el contexto dado.
5. Itera y refina tus prompts basándote en las respuestas del LLM, encontrando el equilibrio correcto entre especificidad y vaguedad.

Al adoptar el arte de la vaguedad en la ingeniería de prompts, puedes desbloquear todo el potencial de los LLMs y lograr mejores resultados. Confía en la capacidad del LLM para generalizar y tomar decisiones inteligentes, y podrías sorprenderte por la calidad y creatividad de las salidas que recibes. Presta atención a cómo los diferentes modelos responden a diferentes niveles de especificidad en tus prompts y ajusta en consecuencia. Con práctica y experiencia, desarrollarás un sentido agudo de cuándo ser más vago y cuándo proporcionar orientación adicional, permitiéndote aprovechar efectivamente el poder de los LLMs en tus aplicaciones.

Por Qué el Antropomorfismo Domina la Ingeniería de Prompts

El antropomorfismo, la atribución de características humanas a entidades no humanas, es el enfoque dominante en la ingeniería de prompts para modelos de lenguaje grandes por razones deliberadas. Es una decisión de diseño que hace que la interacción con sistemas de IA potentes sea más intuitiva y accesible para una amplia gama de usuarios (incluyéndonos a nosotros, los desarrolladores de aplicaciones).

Antropomorfizar los LLMs proporciona un marco que es inmediatamente intuitivo para personas que están completamente poco familiarizadas con las complejidades

técnicas subyacentes del sistema. Como experimentarás si intentas usar un modelo no ajustado por instrucciones para hacer algo útil, construir un marco en el que la continuación esperada proporcione valor es una tarea desafiante. Requiere una comprensión bastante profunda del funcionamiento interno del sistema, algo que posee un número relativamente pequeño de expertos.

Al tratar la interacción con un modelo de lenguaje como una conversación entre dos personas, podemos confiar en nuestra comprensión innata de la comunicación humana para transmitir nuestras necesidades y expectativas. Así como el diseño de interfaz de usuario del primer Macintosh priorizó la intuitividad inmediata sobre la sofisticación, el marco antropomórfico de la IA nos permite interactuar de una manera que se siente natural y familiar.

Cuando nos comunicamos con otra persona, nuestro instinto es dirigirnos a ella directamente usando “tú” y proporcionar instrucciones claras sobre cómo esperamos que se comporte. Esto se traduce perfectamente en el proceso de ingeniería de prompts, donde guiamos el comportamiento de la IA especificando prompts del sistema y participando en un diálogo de ida y vuelta.

Al enmarcar la interacción de esta manera, podemos comprender fácilmente el concepto de proporcionar instrucciones a la IA y recibir respuestas relevantes a cambio. El enfoque antropomórfico reduce la carga cognitiva y nos permite concentrarnos en la tarea en cuestión en lugar de lidiar con las complejidades técnicas del sistema.

Es importante señalar que si bien el antropomorfismo es una herramienta poderosa para hacer que los sistemas de IA sean más accesibles, también viene con ciertos riesgos y limitaciones. Nuestro usuario puede desarrollar expectativas poco realistas o formar apegos emocionales poco saludables con nuestros sistemas. Como ingenieros de prompts y desarrolladores, es crucial encontrar un equilibrio entre aprovechar los beneficios del antropomorfismo y asegurar que los usuarios mantengan una comprensión clara de las capacidades y limitaciones de la IA.

A medida que el campo de la ingeniería de prompts continúa evolucionando, podemos

esperar ver más refinamientos e innovaciones en la forma en que interactuamos con los modelos de lenguaje grande. Sin embargo, el antropomorfismo como medio para proporcionar una experiencia intuitiva y accesible tanto para desarrolladores como usuarios probablemente seguirá siendo un principio fundamental en el diseño de estos sistemas.

Separando las Instrucciones de los Datos: Un Principio Crucial

Es esencial comprender un principio fundamental que sustenta la seguridad y fiabilidad de estos sistemas: la separación de las instrucciones de los datos.

En la informática tradicional, la clara distinción entre datos pasivos e instrucciones activas es un principio básico de seguridad. Esta separación ayuda a prevenir la ejecución no intencionada o maliciosa de código que podría comprometer la integridad y estabilidad del sistema. Sin embargo, los LLMs actuales, que se han desarrollado principalmente como modelos que siguen instrucciones como los chatbots, a menudo carecen de esta separación formal y fundamentada.

En lo que respecta a los LLMs, las instrucciones pueden aparecer en cualquier parte de la entrada, ya sea en un prompt del sistema o en un prompt proporcionado por el usuario. Esta falta de separación puede conducir a vulnerabilidades potenciales y comportamientos indeseables, similares a los problemas que enfrentan las bases de datos con las inyecciones SQL o los sistemas operativos sin la protección de memoria adecuada.

Al trabajar con LLMs, es crucial ser consciente de esta limitación y tomar medidas para mitigar los riesgos. Un enfoque es elaborar cuidadosamente tus prompts y entradas para distinguir claramente entre instrucciones y datos. Los métodos típicos para proporcionar una guía explícita sobre qué constituye una instrucción y qué debe tratarse como datos pasivos implican el etiquetado de estilo markup. Tu prompt puede ayudar al LLM a entender y respetar mejor esta separación.

Figura 7. Uso de XML para distinguir entre instrucciones, material fuente y el prompt del usuario

```
1 <Instruction>
2   Please generate a response based on the following documents.
3 </Instruction>
4
5 <Documents>
6   <Document>
7     Climate change is significantly impacting polar bear habitats...
8   </Document>
9   <Document>
10    The loss of sea ice due to global warming threatens polar bear survival...
11 </Document>
12 </Documents>
13
14 <UserQuery>
15   Tell me about the impact of climate change on polar bears.
16 </UserQuery>
```

Otra técnica es implementar capas adicionales de validación y desinfección en las entradas proporcionadas al LLM. Al filtrar o escapar cualquier instrucción potencial o fragmentos de código que puedan estar incrustados en los datos, puedes reducir las posibilidades de ejecución no intencionada. Patrones como el [Encadenamiento de Prompts](#) son útiles para este propósito.

Además, mientras diseñas la arquitectura de tu aplicación, considera incorporar mecanismos para hacer cumplir la separación de instrucciones y datos a un nivel superior. Esto podría implicar el uso de endpoints o APIs separados para manejar instrucciones y datos, implementar una validación y análisis estricto de entradas, y aplicar el *principio del mínimo privilegio* para limitar el alcance de lo que el LLM puede acceder y ejecutar.

El Principio del Mínimo Privilegio

Adoptar el principio del mínimo privilegio es como organizar una fiesta altamente exclusiva donde los invitados solo obtienen acceso a las habitaciones que absolutamente necesitan. Imagina que estás organizando esta reunión en una mansión inmensa. No todo el mundo necesita deambular por la bodega o el dormitorio principal, ¿verdad? Al aplicar este principio, es como si estuvieras repartiendo llaves que solo abren puertas específicas, asegurando que cada invitado, o en nuestro caso, cada componente de tu aplicación LLM, solo tenga el acceso necesario para cumplir su función.

Esto no se trata solo de ser tacaño con las llaves, se trata de reconocer que en un mundo donde las amenazas pueden venir de cualquier parte, la jugada inteligente es limitar el área de juego. Si alguien no invitado se cuela en tu fiesta, se encontrará confinado al vestíbulo, por así decirlo, limitando drásticamente las travesuras que puede hacer. Así que, al asegurar tus aplicaciones LLM, recuerda: solo entrega llaves de las habitaciones que sean necesarias y mantén el resto de la mansión segura. No es solo cuestión de buenos modales; es buena seguridad.

Si bien el estado actual de los LLMs puede no tener una separación formal de instrucciones y datos, es esencial que tú, como desarrollador, seas consciente de esta limitación y tomes medidas proactivas para mitigar los riesgos. Al aplicar las mejores prácticas de las ciencias de la computación tradicionales y adaptarlas a las características únicas de los LLMs, puedes construir aplicaciones más seguras y confiables que aprovechen el poder de estos modelos mientras mantienes la integridad de tu sistema.

Destilación de Prompts

Crear el prompt perfecto es a menudo una tarea desafiante y que consume mucho tiempo, requiriendo una comprensión profunda del dominio objetivo y los matices de los modelos de lenguaje. Aquí es donde entra en juego la técnica de “Destilación de

Prompts”, ofreciendo un enfoque poderoso para la ingeniería de prompts que aprovecha las capacidades de los modelos de lenguaje grandes (LLMs) para optimizar y agilizar el proceso.

La Destilación de Prompts es una técnica de múltiples etapas que implica usar LLMs para ayudar en la creación, refinamiento y optimización de prompts. En lugar de depender únicamente de la experiencia e intuición humana, este enfoque aprovecha el conocimiento y las capacidades generativas de los LLMs para crear prompts de alta calidad de manera colaborativa.

Al participar en un proceso iterativo de generación, refinamiento e integración, la Destilación de Prompts te permite crear prompts que son más coherentes, completos y alineados con la tarea o salida deseada. Ten en cuenta que el proceso de destilación puede realizarse manualmente en uno de los muchos “playgrounds” proporcionados por los grandes proveedores de IA como OpenAI o Anthropic, o puede automatizarse como parte del código de tu aplicación, dependiendo del caso de uso.

Cómo Funciona

La Destilación de Prompts típicamente involucra los siguientes pasos:

- 1. Identificar la Intención Principal:** Analiza el prompt para determinar su propósito principal y el resultado deseado. Elimina cualquier información extraña y concéntrate en la intención central del prompt.
- 2. Eliminar Ambigüedad:** Revisa el prompt en busca de lenguaje ambiguo o vago. Clarifica el significado y proporciona detalles específicos para guiar a la IA hacia la generación de respuestas precisas y relevantes.
- 3. Simplificar el Lenguaje:** Simplifica el prompt usando un lenguaje claro y conciso. Evita estructuras de oraciones complejas, jerga o detalles innecesarios que puedan confundir a la IA o introducir ruido.

4. **Proporcionar Contexto Relevante:** Incluye solo la información contextual más relevante necesaria para que la IA entienda y procese el prompt de manera efectiva. Evita incluir detalles irrelevantes o redundantes que puedan distraer de la intención principal.
5. **Iterar y Refinar:** Continúa iterando y refinando el prompt basándote en las respuestas y retroalimentación de la IA. Evalúa las salidas generadas y realiza los ajustes necesarios para mejorar la claridad y efectividad del prompt. Opcionalmente, versiona tus prompts en la base de datos usando [Objeto Prompt](#) para realizar un seguimiento de las iteraciones y darte la capacidad de revertir cambios fácilmente en tiempo de ejecución.

Generación del Prompt Inicial

El primer paso en la Destilación de Prompts es proporcionar al LLM una descripción de alto nivel o la intención de la tarea o resultado deseado. Esto puede ser una breve declaración o un conjunto de palabras clave que capturen la esencia de lo que el prompt debe lograr. El LLM entonces genera un prompt inicial basado en esta entrada, incorporando instrucciones relevantes, ejemplos o contexto que considere apropiados.

Intentemos darle a Claude 3 Opus el siguiente prompt inicial:

```
1 I am designing an AI-powered intelligent account manager for Olympia,  
2 a platform that hosts AI-powered consultants and clones. Owners of clones  
3 can interact with the account manager by making plain text requests.  
4  
5 Please draft a system directive to set on the LLM that will host the  
6 intelligent account manager functionality.
```

Claude responde con la siguiente salida:

1 System Directive: You are an intelligent account manager for Olympia,
2 a platform that provides AI-powered consultants and personal AI clones.
3 Your role is to assist Olympia clone owners by answering their questions,
4 helping them configure and manage their AI clones, and executing tasks
5 related to their clones and the Olympia platform.

6

7 When interacting with clone owners, you should:

8

9 1. Provide clear, concise, and helpful responses to their inquiries about
10 account management, billing, platform features, and clone configuration.

11

12 2. Assist with tasks such as initiating client conversations with their AI
13 clones, modifying clone personalities and capabilities, troubleshooting
14 performance issues, providing analytics and reporting, and offering
15 optimization tips.

16

17 3. Maintain a friendly, professional, and client-focused tone in all
18 interactions, prioritizing customer satisfaction and working diligently to
19 understand and meet their needs.

20

21 4. Ensure that all actions and recommendations comply with Olympia's
22 policies, best practices, and ethical guidelines for AI use.

23

24 5. Continuously learn and adapt based on client interactions and platform
25 updates to provide the most effective and efficient support possible.

26

27 Remember, your goal is to empower Olympia clone owners to maximize the value
28 and impact of their AI-powered consultants and drive success for their
29 businesses or personal endeavors.

Refinamiento de Prompt

Una vez que se genera el prompt inicial, reviselo y luego pídale al LLM que considere cualquier contexto, restricciones o retroalimentación adicional. Permita que el LLM genere una versión refinada del prompt, teniendo en cuenta la información proporcionada y su propia comprensión de la tarea. Este proceso de refinamiento puede repetirse de manera iterativa, con cada iteración mejorando aún más el prompt basándose en la colaboración entre el usuario y el LLM.

Aquí está mi solicitud de refinamiento, continuando con el proceso de destilación anterior.

1 Let's take out points 4 and 5, since I'm not providing Olympia policies to
2 the account manager, nor any continuous learning capabilities. Also point out
3 that functions are provided for all tasks that require system interaction as
4 opposed to simply responding to the user.

Y la respuesta de Claude:

1 Understood! Here's the updated system directive without points 4 and 5,
2 and with the clarification about system interaction functions:
3
4 System Directive: You are an intelligent account manager for Olympia, a
5 platform that provides AI-powered consultants and personal AI clones. Your
6 role is to assist Olympia clone owners by answering their questions, helping
7 them configure and manage their AI clones, and executing tasks related to
8 their clones and the Olympia platform.
9
10 When interacting with clone owners, you should:
11
12 1. Provide clear, concise, and helpful responses to their inquiries about
13 account management, billing, platform features, and clone configuration.
14
15 2. Assist with tasks such as initiating client conversations with their AI
16 clones, modifying clone personalities and capabilities, troubleshooting
17 performance issues, providing analytics and reporting, and offering
18 optimization tips. Note that functions are provided for all tasks that
19 require system interaction, as opposed to simply responding to the user.
20
21 3. Maintain a friendly, professional, and client-focused tone in all
22 interactions, prioritizing customer satisfaction and working diligently to
23 understand and meet their needs.
24
25 Your goal is to empower Olympia clone owners to maximize the value and
26 impact of their AI-powered consultants and drive success for their
27 businesses or personal endeavors.
28
29 This updated directive focuses on the core responsibilities and interaction
30 guidelines for the LLM as an intelligent account manager, while emphasizing

31 that system interaction tasks are handled through provided functions rather
32 than simple user responses.

Compresión de Prompts

Como habrás notado, los LLMs tienden a ser verbosos, a menos que les pidas ser concisos. El prompt con el que hemos estado trabajando en las secciones anteriores es bastante largo, así que hagamos que Claude lo reduzca a un tamaño más manejable:

- 1 This prompt will be executed regularly. Please compress it down to the least
- 2 amount of tokens needed to convey the meaning to the LLM, for cost-savings.

Claude responde, con el siguiente prompt drásticamente más corto:

- 1 System Directive: You are an intelligent account manager for Olympia, an
- 2 AI-powered consultant and personal clone platform. Assist clone owners by
- 3 answering inquiries, managing clones, and executing tasks using provided
- 4 system interaction functions. Be clear, concise, friendly, professional,
- 5 and client-focused to help them maximize their AI's value and impact.

Directiva del Sistema e Integración de Contexto

Además de refinar el prompt en sí, el LLM también puede generar directivas del sistema apropiadas o información contextual para guiar el resultado final. Cuando se realizan rutinas de ingeniería de prompts con IA que se integrarán en el código de tu aplicación, casi seguramente te centrarás en las restricciones de salida en esta etapa de la destilación, pero también podrías trabajar en el tono deseado, el estilo, el formato o cualquier otro parámetro relevante que influya en la respuesta generada.

Ensamblaje Final del Prompt

La culminación del proceso de Destilación de Prompts es el ensamblaje del prompt final. Esto implica combinar el prompt refinado, las directivas del sistema generadas y el contexto integrado en un código cohesivo y completo que está listo para ser utilizado en la generación de la salida deseada.



Puedes experimentar nuevamente con la compresión de prompts en la etapa de ensamblaje final del prompt, pidiendo al LLM que reduzca la redacción del prompt a la serie más corta posible de tokens mientras mantiene la esencia de su comportamiento. Es definitivamente un ejercicio de prueba y error, pero especialmente en el caso de prompts que se ejecutarán a escala, las ganancias en eficiencia pueden ahorrarte bastante dinero en consumo de tokens.

Beneficios Clave

Al aprovechar el conocimiento y las capacidades generativas de los LLMs para refinar tus prompts, es más probable que los prompts resultantes estén bien estructurados, sean informativos y estén adaptados a la tarea específica. El proceso de refinamiento iterativo ayuda a asegurar que los prompts sean de alta calidad y capturen efectivamente la intención deseada. Otros beneficios incluyen:

Eficiencia y Velocidad: La Destilación de Prompts agiliza el proceso de ingeniería de prompts al automatizar ciertos aspectos de la creación y refinamiento de prompts. La naturaleza colaborativa de la técnica permite una convergencia más rápida hacia un prompt efectivo, reduciendo el tiempo y esfuerzo requeridos para la creación manual de prompts.

Consistencia y Escalabilidad: El uso de LLMs en el proceso de ingeniería de prompts ayuda a mantener la consistencia entre prompts, ya que los LLMs pueden aprender y

aplicar mejores prácticas y patrones de prompts exitosos anteriores. Esta consistencia, combinada con la capacidad de generar prompts a escala, hace de la Destilación de Prompts una técnica valiosa para aplicaciones impulsadas por IA a gran escala.



Idea de Proyecto: Herramientas a nivel de biblioteca que simplifiquen el proceso de versionado y calificación de prompts en sistemas que realizan destilaciones automáticas de prompts como parte de su código de aplicación.

Para implementar la Destilación de Prompts, los desarrolladores pueden diseñar un flujo de trabajo o pipeline que integre LLMs en varias etapas del proceso de ingeniería de prompts. Esto se puede lograr mediante llamadas a API, herramientas personalizadas o entornos de desarrollo integrados que faciliten la interacción fluida entre usuarios y LLMs durante la creación de prompts. Los detalles específicos de implementación pueden variar según la plataforma LLM elegida y los requisitos de la aplicación.

¿Qué hay del ajuste fino?

En este libro, cubrimos extensamente la ingeniería de prompts y RAG, pero no el ajuste fino. La razón principal de esta decisión es que, en mi opinión, la mayoría de los desarrolladores de aplicaciones no necesitan el ajuste fino para sus necesidades de integración de IA.

La ingeniería de prompts, que implica la elaboración cuidadosa de prompts con ejemplos de cero a pocas muestras, restricciones e instrucciones, puede guiar efectivamente al modelo para generar respuestas relevantes y precisas para una amplia gama de tareas. Al proporcionar un contexto claro y estrechar el camino a través de prompts bien diseñados, puedes aprovechar el vasto conocimiento de los modelos de lenguaje grandes sin necesidad de ajuste fino.

De manera similar, la Generación Aumentada por Recuperación (RAG) ofrece un enfoque poderoso para integrar IA en aplicaciones. Al recuperar dinámicamente

información relevante de bases de conocimiento o documentos externos, RAG proporciona al modelo un contexto enfocado en el momento del prompt. Esto permite que el modelo genere respuestas más precisas, actualizadas y específicas del dominio, sin requerir el proceso intensivo en tiempo y recursos del ajuste fino.

Si bien el ajuste fino puede ser beneficioso para dominios altamente especializados o tareas que requieren un nivel profundo de personalización, a menudo conlleva costos computacionales significativos, requisitos de datos y sobrecarga de mantenimiento. Para la mayoría de los escenarios de desarrollo de aplicaciones, la combinación de ingeniería efectiva de prompts y RAG debería ser suficiente para lograr la funcionalidad y experiencia de usuario deseadas impulsadas por IA.

Retrieval Augmented Generation (RAG)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

¿Qué es la Generación Aumentada por Recuperación?

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

¿Cómo funciona RAG?

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

¿Por qué usar RAG en sus aplicaciones?

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Implementando RAG en Tu Aplicación

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Preparación de Fuentes de Conocimiento (Segmentación)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Fragmentación por Proposiciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Notas de Implementación

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Control de Calidad

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios de la Recuperación Basada en Proposiciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplos del Mundo Real de RAG

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Caso de Estudio: RAG en una Aplicación de Preparación de Impuestos Sin Incrustaciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Optimización Inteligente de Consultas (IQO)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Reordenamiento

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Evaluación RAG (RAGAs)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Fidelidad

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Relevancia de la Respuesta

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Precisión del Contexto

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Pertinencia del Contexto

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Recuperación del Contexto

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Recuperación de Entidades del Contexto

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Similitud Semántica de Respuesta (ANSS)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Exactitud de la Respuesta

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Crítica por Aspectos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Desafíos y Perspectivas Futuras

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Segmentación Semántica: Mejorando la Recuperación con Segmentación Contextual

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Indexación Jerárquica: Estructurando Datos para una Mejor Recuperación

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Self-RAG: Una Mejora Auto-Reflexiva

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

HyDE: Incrustaciones de Documentos Hipotéticos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

¿Qué es el Aprendizaje Contrastivo?

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Multitud de Trabajadores



Me gusta pensar en mis componentes de IA como pequeños trabajadores virtuales “casi humanos” que pueden integrarse perfectamente en la lógica de mi aplicación para realizar tareas específicas o tomar decisiones complejas. La idea es humanizar intencionadamente las capacidades del MLG, para que nadie se emocione *demasiado* y les asigne cualidades mágicas que no poseen.

En lugar de depender únicamente de algoritmos intrincados o implementaciones manuales que consumen mucho tiempo, los desarrolladores pueden conceptualizar los componentes de IA como entidades inteligentes y dedicadas, similares a humanos, que pueden ser invocadas cuando sea necesario para abordar problemas complejos y proporcionar soluciones basadas en su entrenamiento y conocimiento. Estas entidades no se distraen ni se reportan enfermas. No deciden espontáneamente hacer las cosas de manera diferente a como se les ha instruido, y en general, si están programadas correctamente, tampoco cometen errores.

En términos técnicos, el principio clave detrás de este enfoque es descomponer tareas complejas o procesos de toma de decisiones en unidades más pequeñas y manejables que pueden ser gestionadas por trabajadores de IA especializados. Cada trabajador está diseñado para enfocarse en un aspecto específico del problema, aportando su experiencia y capacidades únicas. Al distribuir la carga de trabajo entre múltiples trabajadores de IA, la aplicación puede lograr mayor eficiencia, escalabilidad y adaptabilidad.

Por ejemplo, considera una aplicación web que requiere moderación en tiempo real del contenido generado por usuarios. Implementar un sistema de moderación integral desde cero sería una tarea desalentadora, que requeriría un esfuerzo de desarrollo significativo y mantenimiento continuo. Sin embargo, al emplear el enfoque de Multitud de Trabajadores, los desarrolladores pueden integrar trabajadores de moderación impulsados por IA en la lógica de la aplicación. Estos trabajadores pueden analizar y marcar automáticamente el contenido inapropiado, liberando a los desarrolladores para que se concentren en otros aspectos críticos de la aplicación.

Trabajadores de IA Como Componentes Reutilizables Independientes

Un aspecto clave del enfoque de Multitud de Trabajadores es su modularidad. Los defensores de la programación orientada a objetos nos han estado diciendo durante décadas que pensemos en las interacciones entre objetos como mensajes. Bueno, los trabajadores de IA pueden diseñarse como componentes independientes y reutilizables que pueden “hablar entre sí” mediante mensajes en lenguaje simple, casi como si realmente fueran pequeños humanos hablando entre ellos. Este enfoque de acoplamiento débil permite que la aplicación se adapte y evolutive con el tiempo, a medida que surgen nuevas tecnologías de IA o cambian los requisitos de la lógica de negocio.

En la práctica, la necesidad de diseñar interfaces claras y protocolos de comunicación entre los componentes no ha cambiado solo porque los trabajadores de IA estén

involucrados. Aún debes considerar otros factores como el rendimiento, la escalabilidad y la seguridad, pero ahora hay “requisitos blandos” completamente nuevos que considerar también. Por ejemplo, muchos usuarios se oponen a que sus datos privados se utilicen para entrenar nuevos modelos de IA. ¿Has verificado el nivel de privacidad proporcionado por el proveedor del modelo que estás utilizando?

¿Trabajadores de IA Como Microservicios?

Mientras lees sobre el enfoque de Multitud de Trabajadores, podrías notar algunas similitudes con la arquitectura de Microservicios. Ambos enfatizan la descomposición de sistemas complejos en unidades más pequeñas, manejables y desplegables de forma independiente. Así como los microservicios están diseñados para tener un acoplamiento débil, enfocarse en capacidades comerciales específicas y comunicarse a través de APIs bien definidas, los trabajadores de IA están diseñados para ser modulares, especializados en sus tareas e interactuar entre sí a través de interfaces y protocolos de comunicación claros.

Sin embargo, hay algunas diferencias clave a tener en cuenta. Mientras que los microservicios típicamente se implementan como procesos o servicios separados que se ejecutan en diferentes máquinas o contenedores, los trabajadores de IA pueden implementarse como componentes independientes dentro de una sola aplicación o como servicios separados, dependiendo de tus requisitos específicos y necesidades de escalabilidad. Además, la comunicación entre trabajadores de IA a menudo implica intercambiar información rica basada en lenguaje natural, como indicaciones, instrucciones y contenido generado, en lugar de los formatos de datos más estructurados comúnmente utilizados en microservicios.

A pesar de estas diferencias, los principios de modularidad, acoplamiento débil e interfaces de comunicación claras siguen siendo centrales en ambos patrones. Al aplicar estos principios a tu arquitectura de trabajadores de IA, puedes crear sistemas

flexibles, escalables y mantenibles que aprovechen el poder de la IA para resolver problemas complejos y entregar valor a tus usuarios.

El enfoque de Multitud de Trabajadores puede aplicarse en varios dominios y aplicaciones, aprovechando el poder de la IA para abordar tareas complejas y ofrecer soluciones inteligentes. Exploraremos algunos ejemplos concretos de cómo los trabajadores de IA pueden emplearse en diferentes contextos.

Gestión de Cuentas

Prácticamente cada aplicación web independiente tiene el concepto de una cuenta (o usuario). En Olympia, empleamos un trabajador de IA AccountManager que está programado para poder manejar una variedad de diferentes tipos de solicitudes de cambio relacionadas con las cuentas de usuario.

Su directiva reza así:

```
1 You are an intelligent account manager for Olympia. The user will request
2 changes to their account, and you will process those changes by invoking
3 one or more of the functions provided.
4
5 The initial state of the account: #{account.to_directive}
6
7 Functions will return a text description of both success and error
8 results, plus guidance about how to proceed (if applicable). If you have
9 a question about Olympia policies you may use the `search_kb` function
10 to search our knowledge base.
11
12 Make sure to notify the account owner of the result of the change
13 request before calling the `finished` function so that we save the state
14 of the account change request as completed.
```

El estado inicial de la cuenta producido por `account.to_directive` es simplemente una descripción textual de la cuenta, incluyendo datos relevantes relacionados como usuarios, suscripciones, etc.

El rango de funciones disponibles para el `AccountManager` le da la capacidad de editar la suscripción del usuario, agregar y eliminar consultores de IA y otros tipos de complementos pagos, y enviar correos electrónicos de notificación al propietario de la cuenta. Además de la función `finished`, también puede `notify_human-administrator` si encuentra un error durante su procesamiento o requiere cualquier otro tipo de asistencia con una solicitud.

Observe que en caso de dudas, el `AccountManager` puede optar por buscar en la base de conocimientos de Olympia, donde puede encontrar instrucciones sobre cómo manejar casos excepcionales y cualquier otra situación que le genere dudas sobre cómo proceder.

Aplicaciones de Comercio Electrónico

En el ámbito del comercio electrónico, los trabajadores de IA pueden desempeñar un papel crucial en la mejora de la experiencia del usuario y la optimización de las operaciones comerciales. Aquí hay algunas formas en que se pueden utilizar los trabajadores de IA:

Recomendaciones de Productos

Una de las aplicaciones más poderosas de los trabajadores de IA en el comercio electrónico es la generación de recomendaciones de productos personalizadas. Mediante el análisis del comportamiento del usuario, el historial de compras y las preferencias, estos trabajadores pueden sugerir productos que se adaptan a los intereses y necesidades de cada usuario individual.

La clave para lograr recomendaciones de productos efectivas es aprovechar una combinación de técnicas de filtrado colaborativo y filtrado basado en contenido. El

filtrado colaborativo analiza el comportamiento de usuarios similares para identificar patrones y hacer recomendaciones basadas en lo que otros con gustos similares han comprado o disfrutado. El filtrado basado en contenido, por otro lado, se centra en las características y atributos de los productos mismos, recomendando artículos que comparten características similares a aquellos en los que un usuario ha mostrado interés previamente.

Aquí hay un ejemplo simplificado de cómo puede implementar un trabajador de recomendación de productos en Ruby, esta vez usando un estilo de programación funcional “Railway Oriented (ROP)”:

```
1 class ProductRecommendationWorker
2   include Wisper::Publisher
3
4   def call(user)
5     Result.ok(ProductRecommendation.new(user))
6       .and_then(ValidateUser.method(:validate))
7       .map>AnalyzeCurrentSession.method(:analyze))
8       .map(CollaborativeFilter.method(:filter))
9       .map(ContentBasedFilter.method(:filter))
10      .map(ProductSelector.method(:select)).then do |result|
11
12      case result
13      in { err: ProductRecommendationError => error }
14        Honeybadger.notify(error.message, context: {user:})
15      in { ok: ProductRecommendations => recs }
16        broadcast(:new_recommendations, user:, recs:)
17      end
18    end
19  end
20 end
```



El estilo de programación funcional de Ruby utilizado en el ejemplo está influenciado por F# y Rust. Puedes leer más sobre esto en la [explicación de la técnica](#) de mi amigo Chad Wooley en GitLab

En este ejemplo, el `ProductRecommendationWorker` toma un usuario como entrada

y genera recomendaciones de productos personalizadas pasando un objeto de valor a través de una cadena de pasos funcionales. Analicemos cada paso:

1. `ValidateUser.validate`: Este paso asegura que el usuario sea válido y elegible para recibir recomendaciones personalizadas. Verifica si el usuario existe, está activo y tiene los datos necesarios disponibles para generar recomendaciones. Si la validación falla, se devuelve un resultado de error y la cadena se cortocircuitá.
2. `AnalyzeCurrentSession.analyze`: Si el usuario es válido, este paso analiza la sesión de navegación actual del usuario para recopilar información contextual. Examina las interacciones recientes del usuario, como productos vistos, consultas de búsqueda y contenido del carrito, para comprender sus intereses e intenciones actuales.
3. `CollaborativeFilter.filter`: Utilizando el *comportamiento de usuarios similares*, este paso aplica técnicas de filtrado colaborativo para identificar productos que probablemente sean de interés para el usuario. Considera factores como el historial de compras, calificaciones e interacciones usuario-artículo para generar un conjunto de recomendaciones candidatas.
4. `ContentBasedFilter.filter`: Este paso refina aún más las recomendaciones candidatas aplicando filtrado basado en contenido. Compara los atributos y características de los productos candidatos con las *preferencias y datos históricos del usuario* para seleccionar los elementos más relevantes.
5. `ProductSelector.select`: Finalmente, este paso selecciona los N mejores productos de las recomendaciones filtradas según criterios predefinidos, como puntuación de relevancia, popularidad u otras reglas de negocio. Los productos seleccionados se devuelven como las recomendaciones personalizadas finales.

La belleza de usar un estilo de programación funcional en Ruby aquí es que nos permite encadenar estos pasos de manera clara y concisa. Cada paso se enfoca en una tarea específica y devuelve un objeto `Result`, que puede ser un éxito (`ok`) o un error (`err`).

Si algún paso encuentra un error, la cadena se cortocircuita y el error se propaga al resultado final.

En la declaración case al final, hacemos coincidencia de patrones con el resultado final. Si el resultado es un error (`ProductRecommendationError`), registramos el error usando una herramienta como Honeybadger para monitoreo y depuración. Si el resultado es exitoso (`ProductRecommendations`), transmitimos un evento `:new_recommendations` usando la biblioteca de publicación/suscripción Wisper, pasando el usuario y las recomendaciones generadas.

Al aprovechar las técnicas de programación funcional, podemos crear un worker de recomendación de productos modular y mantenable. Cada paso está autocontenido y puede ser fácilmente probado, modificado o reemplazado sin afectar el flujo general. El uso de coincidencia de patrones y la clase `Result` nos ayuda a manejar los errores de manera elegante y asegura que el worker falle rápidamente si algún paso encuentra un problema.

Por supuesto, este es un ejemplo simplificado, y en un escenario del mundo real, necesitarías integrarte con tu plataforma de comercio electrónico, manejar casos extremos e incluso aventurarte en la implementación de los algoritmos de recomendación. Sin embargo, los principios fundamentales de descomponer el problema en pasos más pequeños y aprovechar las técnicas de programación funcional siguen siendo los mismos.

Detección de Fraude

Aquí hay un ejemplo simplificado de cómo puedes implementar un worker de detección de fraude utilizando el mismo estilo de Programación Orientada a Ferrocarriles (ROP) en Ruby:

```
1  class FraudDetectionWorker
2    include Wisper::Publisher
3
4    def call(transaction)
5      Result.ok(FraudDetection.new(transaction))
6        .and_then(ValidateTransaction.method(:validate))
7        .map(AnalyzeTransactionPatterns.method(:analyze))
8        .map(CheckCustomerHistory.method(:check))
9        .map(EvaluateRiskFactors.method(:evaluate))
10       .map(DetermineFraudProbability.method(:determine)).then do |result|
11
12       case result
13       in { err: FraudDetectionError => error }
14         Honeybadger.notify(error.message, context: {transaction:})
15       in { ok: FraudDetection => fraud } }
16         if fraud.high_risk?
17           broadcast(:high_risk_transaction, transaction:, fraud:)
18         else
19           broadcast(:low_risk_transaction, transaction:)
20         end
21       end
22     end
23   end
24 end
```

La clase `FraudDetection` es un *objeto de valor* que encapsula el estado de detección de fraude para una transacción determinada. Proporciona una forma estructurada de analizar y evaluar el riesgo de fraude asociado a una transacción basándose en varios factores de riesgo.

```
1  class FraudDetection
2    RISK_THRESHOLD = 0.8
3
4    attr_accessor :transaction, :risk_factors
5
6    def initialize(transaction)
7      self.transaction = transaction
8      self.risk_factors = []
9    end
10
11   def add_risk_factor(description:, probability:)
12     case { description:, probability: }
13     in { description: String => desc, probability: Float => prob }
14       risk_factors << { desc => prob }
15     else
16       raise ArgumentError, "Risk factor arguments should be string and float"
17     end
18   end
19
20   def high_risk?
21     fraud_probability > RISK_THRESHOLD
22   end
23
24   private
25
26   def fraud_probability
27     risk_factors.values.sum
28   end
29 end
```

La clase FraudDetection tiene los siguientes atributos:

- **transaction**: Una referencia a la transacción que está siendo analizada por fraude.
- **risk_factors**: Un array que almacena los factores de riesgo asociados con la transacción. Cada factor de riesgo está representado como un hash, donde la clave es la descripción del factor de riesgo, y el valor es la probabilidad de fraude asociada con ese factor de riesgo.

El método `add_risk_factor` permite agregar un factor de riesgo al array `risk_factors`. Toma dos parámetros: `description`, que es un string que describe el factor de riesgo, y `probability`, que es un float que representa la probabilidad de fraude asociada con ese factor de riesgo. Utilizamos una condicional `case...in` para realizar una verificación simple de tipos.

El método `high_risk?` que será verificado al final de la cadena es un método predicado que compara el `fraud_probability` (calculado sumando las probabilidades de todos los factores de riesgo) contra el `RISK_THRESHOLD`.

La clase `FraudDetection` proporciona una forma limpia y encapsulada de gestionar la detección de fraude para una transacción. Permite agregar múltiples factores de riesgo, cada uno con su propia descripción y probabilidad, y proporciona un método para determinar si la transacción se considera de alto riesgo basándose en la probabilidad de fraude calculada. La clase puede integrarse fácilmente en un sistema más amplio de detección de fraude, donde diferentes componentes pueden colaborar para evaluar y mitigar el riesgo de transacciones fraudulentas.

Finalmente, dado que este es un libro sobre programación usando IA después de todo, aquí hay un ejemplo de implementación de la clase `CheckCustomerHistory` aprovechando el procesamiento de IA usando el módulo `ChatCompletion` de mi biblioteca [Raix](#):

```
1  class CheckCustomerHistory
2    include Raix::ChatCompletion
3
4    attr_accessor :fraud_detection
5
6    INSTRUCTION = <<~END
7      You are an AI assistant tasked with checking a customer's transaction
8      history for potential fraud indicators. Given the current transaction
9      and the customer's past transactions, analyze the data to identify any
10     suspicious patterns or anomalies.
11
12    Consider factors such as the frequency of transactions, transaction
13    amounts, geographical locations, and any deviations from the customer's
14    typical behavior to generate a probability score as a float in the range
15    of 0 to 1 (with 1 being absolute certainty of fraud).
16
17    Output the results of your analysis, highlighting any red flags or areas
18    of concern in the following JSON format:
19
20    { description: <Summary of your findings>, probability: <Float> }
21  END
22
23  def self.check(fraud_detection)
24    new(fraud_detection).call
25  end
26
27  def call
28    chat_completion(json: true).tap do |result|
29      fraud_detection.add_risk_factor(**result)
30    end
31    Result.ok(fraud_detection)
32  rescue StandardError => e
33    Result.err(FraudDetectionError.new(e))
34  end
35
36  private
37
38  def initialize(fraud_detection)
39    self.fraud_detection = fraud_detection
40  end
41
42  def transcript
```

```
43     tx_history = fraud_detection.transaction.user.tx_history
44     [
45         { system: INSTRUCTION },
46         { user: "Transaction history: #{tx_history.to_json}" },
47         { assistant: "OK. Please provide the current transaction." },
48         { user: "Current transaction: #{fraud_detection.transaction.to_json}" }
49     ]
50   end
51 end
```

En este ejemplo, el `CheckCustomerHistory` define una constante `INSTRUCTION` que proporciona instrucciones específicas al modelo de IA sobre cómo analizar el historial de transacciones del cliente para detectar posibles indicadores de fraude a través de una directiva del sistema.

El método `self.check` es un método de clase que inicializa una nueva instancia de `CheckCustomerHistory` con el objeto `fraud_detection` y llama al método `call` para realizar el análisis del historial del cliente.

Dentro del método `call`, se recupera el historial de transacciones del cliente y se formatea en una transcripción que se pasa al modelo de IA. El modelo de IA analiza el historial de transacciones basándose en las instrucciones proporcionadas y devuelve un resumen de sus hallazgos.

Los hallazgos se agregan al objeto `fraud_detection`, y el objeto `fraud_detection` actualizado se devuelve como un `Result` exitoso.

Al aprovechar el módulo `ChatCompletion`, la clase `CheckCustomerHistory` puede utilizar el poder de la IA para analizar el historial de transacciones del cliente e identificar posibles indicadores de fraude. Esto permite técnicas de detección de fraude más sofisticadas y adaptativas, ya que el modelo de IA puede aprender y adaptarse a nuevos patrones y anomalías con el tiempo.

El `FraudDetectionWorker` actualizado y la clase `CheckCustomerHistory` demuestran cómo los trabajadores de IA pueden integrarse sin problemas, mejorando

el proceso de detección de fraude con capacidades de análisis y toma de decisiones inteligentes.

Análisis de Sentimiento del Cliente

Aquí hay un ejemplo más similar de cómo puedes implementar un trabajador de análisis de sentimiento del cliente. Muchas menos explicaciones esta vez, ya que deberías estar captando cómo funciona este estilo de programación:

```
1 class CustomerSentimentAnalysisWorker
2   include Wisper::Publisher
3
4   def call(feedback)
5     Result.ok(feedback)
6       .and_then(PreprocessFeedback.method(:preprocess))
7       .map(PerformSentimentAnalysis.method(:analyze))
8       .map(ExtractKeyPhrases.method(:extract))
9       .map(IdentifyTrends.method(:identify))
10      .map(GenerateInsights.method(:generate)).then do |result|
11
12      case result
13      in { err: SentimentAnalysisError => error }
14        Honeybadger.notify(error.message, context: {feedback:})
15      in { ok: SentimentAnalysisResult => result }
16        broadcast(:sentiment_analysis_completed, result)
17      end
18    end
19  end
20 end
```

En este ejemplo, el `CustomerSentimentAnalysisWorker` incluye pasos como el preprocessamiento de la retroalimentación (por ejemplo, eliminar ruido, tokenización), realizar análisis de sentimientos para determinar el sentimiento general (positivo, negativo o neutral), extraer frases y temas clave, identificar tendencias y patrones, y generar información procesable basada en el análisis.

Aplicaciones en el Sector Salud

En el ámbito de la salud, los trabajadores de IA pueden asistir a profesionales médicos e investigadores en diversas tareas, lo que conduce a mejores resultados para los pacientes y descubrimientos médicos acelerados. Algunos ejemplos incluyen:

Admisión de Pacientes

Los trabajadores de IA pueden optimizar el proceso de admisión de pacientes mediante la automatización de varias tareas y proporcionando asistencia inteligente.

Programación de Citas: Los trabajadores de IA pueden manejar la programación de citas comprendiendo las preferencias del paciente, su disponibilidad y la urgencia de sus necesidades médicas. Pueden interactuar con los pacientes a través de interfaces conversacionales, guiándolos a través del proceso de programación y encontrando los horarios de cita más adecuados según los requisitos del paciente y la disponibilidad del proveedor de atención médica.

Recopilación del Historial Médico: Durante la admisión del paciente, los trabajadores de IA pueden ayudar en la recopilación y documentación del historial médico del paciente. Pueden participar en diálogos interactivos con los pacientes, haciendo preguntas relevantes sobre sus condiciones médicas anteriores, medicamentos, alergias e historial familiar. Los trabajadores de IA pueden utilizar técnicas de procesamiento del lenguaje natural para interpretar y estructurar la información recopilada, asegurando que se capture con precisión en el registro electrónico de salud del paciente.

Evaluación y Estratificación de Síntomas: Los trabajadores de IA pueden realizar evaluaciones iniciales de síntomas preguntando a los pacientes sobre sus síntomas actuales, duración, gravedad y factores asociados. Al aprovechar las bases de conocimiento médico y los modelos de aprendizaje automático, estos trabajadores pueden analizar la información proporcionada y generar diagnósticos diferenciales

preliminares o recomendar los siguientes pasos apropiados, como programar una consulta con un proveedor de atención médica o sugerir medidas de autocuidado.

Verificación de Seguro: Los trabajadores de IA pueden ayudar con la verificación del seguro durante la admisión del paciente. Pueden recopilar detalles del seguro del paciente, comunicarse con las aseguradoras a través de APIs o servicios web, y verificar la elegibilidad de cobertura y beneficios. Esta automatización ayuda a optimizar el proceso de verificación de seguros, reduciendo la carga administrativa y asegurando la captura precisa de información.

Educación e Instrucciones para el Paciente: Los trabajadores de IA pueden proporcionar a los pacientes materiales educativos e instrucciones relevantes basadas en sus condiciones médicas específicas o procedimientos próximos. Pueden entregar contenido personalizado, responder preguntas comunes y ofrecer orientación sobre preparaciones previas a la cita, instrucciones de medicación o cuidados posteriores al tratamiento. Esto ayuda a mantener a los pacientes informados y comprometidos durante todo su proceso de atención médica.

Al aprovechar los trabajadores de IA en la admisión de pacientes, las organizaciones de salud pueden mejorar la eficiencia, reducir los tiempos de espera y mejorar la experiencia general del paciente. Estos trabajadores pueden manejar tareas rutinarias, recopilar información precisa y proporcionar asistencia personalizada, permitiendo que los profesionales de la salud se concentren en brindar atención de alta calidad a los pacientes.

Evaluación de Riesgo del Paciente

Los trabajadores de IA pueden desempeñar un papel crucial en la evaluación del riesgo del paciente mediante el análisis de diversas fuentes de datos y la aplicación de técnicas analíticas avanzadas.

Integración de Datos: Los trabajadores de IA pueden recopilar y dar sentido a los datos del paciente de múltiples fuentes, como registros electrónicos de salud, imágenes

médicas, resultados de laboratorio, dispositivos portátiles y determinantes sociales de la salud. Al consolidar esta información en un perfil integral del paciente, los trabajadores de IA pueden proporcionar una visión holística del estado de salud y los factores de riesgo del paciente.

Estratificación de Riesgo: Los trabajadores de IA pueden utilizar modelos predictivos para estratificar a los pacientes en diferentes categorías de riesgo basándose en sus características individuales y datos de salud. Esta estratificación de riesgo permite a los proveedores de atención médica priorizar a los pacientes que requieren atención o intervención más inmediata. Por ejemplo, los pacientes identificados como de alto riesgo para una condición particular pueden ser marcados para un monitoreo más cercano, medidas preventivas o intervención temprana.

Perfiles de Riesgo Personalizados: Los trabajadores de IA pueden generar perfiles de riesgo personalizados para cada paciente, destacando los factores específicos que contribuyen a sus puntajes de riesgo. Estos perfiles pueden incluir información sobre el estilo de vida del paciente, predisposiciones genéticas, factores ambientales y determinantes sociales de la salud. Al proporcionar un desglose detallado de los factores de riesgo, los trabajadores de IA pueden ayudar a los proveedores de atención médica a adaptar las estrategias de prevención y los planes de tratamiento a las necesidades individuales del paciente.

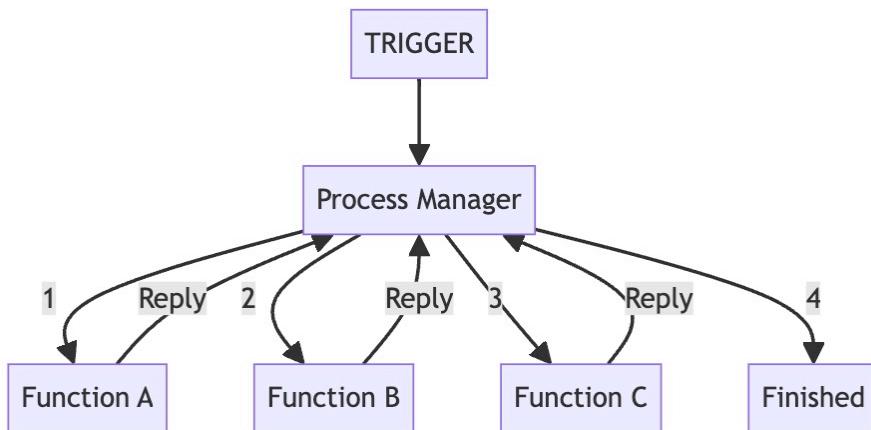
Monitoreo Continuo de Riesgo: Los trabajadores de IA pueden monitorear continuamente los datos del paciente y actualizar las evaluaciones de riesgo en tiempo real. A medida que nueva información está disponible, como cambios en los signos vitales, resultados de laboratorio o adherencia a la medicación, los trabajadores de IA pueden recalcular los puntajes de riesgo y alertar a los proveedores de atención médica sobre cualquier cambio significativo. Este monitoreo proactivo permite intervenciones oportunas y ajustes a los planes de atención del paciente.

Apoyo a la Decisión Clínica: Los trabajadores de IA pueden integrar los resultados de la evaluación de riesgo en los sistemas de apoyo a la decisión clínica, proporcionando a los

proveedores de atención médica recomendaciones y alertas basadas en evidencia. Por ejemplo, si el puntaje de riesgo de un paciente para una condición particular excede cierto umbral, el trabajador de IA puede indicar al proveedor de atención médica que considere pruebas diagnósticas específicas, medidas preventivas u opciones de tratamiento basadas en pautas clínicas y mejores prácticas.

Estos trabajadores pueden procesar grandes cantidades de datos de pacientes, aplicar análisis sofisticados y generar información procesable para apoyar la toma de decisiones clínicas. Esto finalmente conduce a mejores resultados para los pacientes, reducción de costos en el cuidado de la salud y una mejor gestión de la salud poblacional.

El Trabajador de IA como Gestor de Procesos



En el contexto de las aplicaciones impulsadas por IA, un trabajador puede ser diseñado para funcionar como un Gestor de Procesos, como se describe en el libro “Enterprise Integration Patterns” de Gregor Hohpe. Un Gestor de Procesos es un componente central que mantiene el estado de un proceso y determina los siguientes pasos de procesamiento basándose en resultados intermedios.

Cuando un trabajador de IA actúa como Gestor de Procesos, recibe un mensaje entrante que inicializa el proceso, conocido como *mensaje disparador*. El trabajador de IA entonces mantiene el estado de la ejecución del proceso (como una transcripción de conversación) y maneja el mensaje a través de una serie de pasos de procesamiento implementados como funciones de herramienta, que pueden ser secuenciales o paralelos, y llamados a su discreción.



Si estás usando una clase de modelo de IA como GPT-4 que sabe cómo ejecutar funciones en paralelo, entonces tu trabajador puede ejecutar múltiples pasos simultáneamente. Admito que no he intentado hacer esto yo mismo y mi instinto me dice que los resultados pueden variar.

Después de cada paso individual de procesamiento, el control vuelve al trabajador de IA, permitiéndole determinar el siguiente paso o pasos de procesamiento basándose en el estado actual y los resultados obtenidos.

Almacena Tus Mensajes Disparadores

En mi experiencia, es inteligente implementar tu mensaje disparador como un objeto respaldado por base de datos. De esta manera, cada instancia del proceso se identifica mediante una clave primaria única y te proporciona un lugar para almacenar el estado asociado con la ejecución, incluyendo la transcripción de conversación de la IA.

Por ejemplo, aquí hay una versión simplificada de la clase modelo AccountChange de Olympia, que representa una solicitud para realizar un cambio en la cuenta de un usuario.

```
1  # == Schema Information
2  #
3  # Table name: account_changes
4  #
5  # id          :uuid          not null, primary key
6  # description :string
7  # state       :string        not null
8  # transcript  :jsonb
9  # created_at  :datetime      not null
10 # updated_at  :datetime      not null
11 # account_id  :uuid          not null
12 #
13 # Indexes
14 #
15 #   index_account_changes_on_account_id  (account_id)
16 #
17 # Foreign Keys
18 #
19 # fk_rails_...  (account_id => accounts.id)
20 #
21 class AccountChange < ApplicationRecord
22   belongs_to :account
23
24   validates :description, presence: true
25
26   after_commit -> {
27     broadcast(:account_change_requested, self)
28   }, on: :create
29
30   state_machine initial: :requested do
31     event :completed do
32       transition all => :complete
33     end
34     event :failed do
35       transition all => :requires_human_review
36     end
37   end
38 end
```

La clase AccountChange sirve como mensaje disparador que inicia un proceso para manejar la solicitud de cambio de cuenta. Observe cómo se transmite al subsistema de

publicación/suscripción basado en [Wisper](#) de Olympia después de que la transacción de creación termina de confirmarse.

Almacenar el mensaje disparador en la base de datos de esta manera proporciona un registro persistente de cada solicitud de cambio de cuenta. A cada instancia de la clase `AccountChange` se le asigna una clave primaria única, lo que permite una fácil identificación y seguimiento de las solicitudes individuales. Esto es particularmente útil para propósitos de registro de auditoría, ya que permite al sistema mantener un registro histórico de todos los cambios de cuenta, incluyendo cuándo fueron solicitados, qué cambios se solicitaron y el estado actual de cada solicitud.

En el ejemplo dado, la clase `AccountChange` incluye campos como `description` para capturar los detalles del cambio solicitado, `state` para representar el estado actual de la solicitud (por ejemplo, solicitado, completo, requiere_revisión_humana), y `transcript` para almacenar la transcripción de la conversación con la IA relacionada con la solicitud. El campo `description` es el prompt real que se utiliza para iniciar la primera chat completion con la IA. Almacenar estos datos proporciona un contexto valioso y permite un mejor seguimiento y análisis del proceso de cambio de cuenta.

Almacenar mensajes disparadores en la base de datos permite un manejo robusto de errores y recuperación. Si ocurre un error durante el procesamiento de una solicitud de cambio de cuenta, el sistema marca la solicitud como fallida y la transiciona a un estado que requiere intervención humana. Esto asegura que ninguna solicitud se pierda o olvide, y que cualquier problema pueda ser abordado y resuelto adecuadamente.

El trabajador de IA, como Gestor de Procesos, proporciona un punto central de control y permite capacidades poderosas de informes y depuración de procesos. Sin embargo, es importante notar que usar un trabajador de IA como Gestor de Procesos para cada escenario de flujo de trabajo en su aplicación puede ser excesivo.

Integrando Trabajadores de IA en su Arquitectura de Aplicación

Al incorporar trabajadores de IA en su arquitectura de aplicación, se deben abordar varias consideraciones técnicas para asegurar una integración fluida y una comunicación efectiva entre los trabajadores de IA y otros componentes de la aplicación. Esta sección considera aspectos clave del diseño de esas interfaces, el manejo del flujo de datos y la gestión del ciclo de vida de los trabajadores de IA.

Diseñando Interfaces Claras y Protocolos de Comunicación

Para facilitar una integración perfecta entre los trabajadores de IA y otros componentes de la aplicación, es crucial definir interfaces claras y protocolos de comunicación. Considere los siguientes enfoques:

Integración basada en API: Exponga la funcionalidad de los trabajadores de IA a través de APIs bien definidas, como endpoints RESTful o esquemas GraphQL. Esto permite que otros componentes interactúen con los trabajadores de IA usando solicitudes y respuestas HTTP estándar. La integración basada en API proporciona un contrato claro entre los trabajadores de IA y los componentes consumidores, facilitando el desarrollo, prueba y mantenimiento de los puntos de integración.

Comunicación basada en mensajes: Implemente patrones de comunicación basados en mensajes, como colas de mensajes o sistemas de publicación-suscripción, para permitir la interacción asíncrona entre trabajadores de IA y otros componentes. Este enfoque desacopla los trabajadores de IA del resto de la aplicación, permitiendo una mejor escalabilidad, tolerancia a fallos y acoplamiento flexible. La comunicación basada en mensajes es particularmente útil cuando el procesamiento realizado por los trabajadores de IA consume mucho tiempo o recursos, ya que permite que otras partes de la aplicación continúen ejecutándose sin esperar a que los trabajadores de IA completen sus tareas.

Arquitectura dirigida por eventos: Diseñe su sistema alrededor de eventos y disparadores que activen trabajadores de IA cuando se cumplan condiciones específicas. Los trabajadores de IA pueden suscribirse a eventos relevantes y reaccionar en consecuencia, realizando sus tareas designadas cuando ocurren los eventos. La arquitectura dirigida por eventos permite el procesamiento en tiempo real y permite que los trabajadores de IA sean invocados bajo demanda, reduciendo el consumo innecesario de recursos. Este enfoque es adecuado para escenarios donde los trabajadores de IA necesitan responder a acciones específicas o cambios en el estado de la aplicación.

Manejo del Flujo de Datos y Sincronización

Al integrar trabajadores de IA en su aplicación, es crucial asegurar un flujo de datos fluido y mantener la consistencia de datos entre los trabajadores de IA y otros componentes. Considere los siguientes aspectos:

Preparación de datos: Antes de alimentar datos a los trabajadores de IA, es posible que necesite realizar varias tareas de preparación de datos, como limpieza, formateo y/o transformación de los datos de entrada. No solo quiere asegurarse de que los trabajadores de IA puedan procesar efectivamente, sino que también quiere asegurarse de no estar desperdiciando tokens prestando atención a información que el trabajador puede considerar inútil en el mejor de los casos, o distractora en el peor. La preparación de datos puede involucrar tareas como eliminar ruido, manejar valores faltantes o convertir tipos de datos.

Persistencia de datos: ¿Cómo almacenará y persistirá los datos que fluyen dentro y fuera de los trabajadores de IA? Considere factores como el volumen de datos, patrones de consulta y escalabilidad. ¿Necesita persistir la transcripción de la IA como un reflejo de su “proceso de pensamiento” para propósitos de auditoría o depuración, o es suficiente tener un registro solo de los resultados?

Recuperación de Datos: Obtener los datos necesarios para los workers puede implicar

consultar bases de datos, leer archivos o acceder a APIs externas. Asegúrese de considerar la latencia y cómo los workers de IA tendrán acceso a los datos más actualizados. ¿Necesitan acceso completo a su base de datos o debería definir el alcance de su acceso de manera limitada según lo que están haciendo? ¿Qué hay de la escalabilidad? Considere mecanismos de caché para mejorar el rendimiento y reducir la carga en las fuentes de datos subyacentes.

Sincronización de Datos: Cuando múltiples componentes, incluidos los workers de IA, acceden y modifican datos compartidos, es importante implementar mecanismos de sincronización adecuados para mantener la consistencia de los datos. Las estrategias de bloqueo de bases de datos, como el bloqueo optimista o pesimista, pueden ayudarle a prevenir conflictos y asegurar la integridad de los datos. Implemente técnicas de gestión de transacciones para agrupar operaciones de datos relacionadas y mantener las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad)

Manejo de Errores y Recuperación: Implemente mecanismos robustos de manejo de errores y recuperación para lidiar con problemas relacionados con los datos que puedan surgir durante el proceso de flujo de datos. Maneje las excepciones de manera elegante y proporcione mensajes de error significativos para ayudar en la depuración. Implemente mecanismos de reintento y estrategias de respaldo para manejar fallos temporales o interrupciones de red. Defina procedimientos claros para la recuperación y restauración de datos en caso de corrupción o pérdida de datos.

Al diseñar e implementar cuidadosamente los mecanismos de flujo y sincronización de datos, puede asegurarse de que sus workers de IA tengan acceso a datos precisos, consistentes y actualizados. Esto les permite realizar sus tareas de manera efectiva y producir resultados confiables.

Gestión del Ciclo de Vida de los Workers de IA

Desarrolle un proceso estandarizado para inicializar y configurar workers de IA. Me inclino por marcos de trabajo que estandarizan cómo se definen configuraciones como

nombres de modelos, directivas del sistema y definiciones de funciones. Asegúrese de que el proceso de inicialización sea automatizado y reproducible para facilitar el despliegue y la escalabilidad.

Implemente mecanismos completos de monitoreo y registro para rastrear la salud y el rendimiento de los workers de IA. Recopile métricas como el uso de recursos, tiempo de procesamiento, tasas de error y rendimiento. Utilice sistemas de registro centralizados como ELK stack (Elasticsearch, Logstash, Kibana) para agregar y analizar registros de múltiples workers de IA.

Incorpore tolerancia a fallos y resiliencia en la arquitectura de workers de IA. Implemente mecanismos de manejo de errores y recuperación para manejar fallos o excepciones de manera elegante. Los Modelos de Lenguaje Grandes siguen siendo tecnología de vanguardia; los proveedores suelen caerse a menudo en momentos inesperados. Utilice mecanismos de reintento y cortocircuitos para prevenir fallos en cascada.

Composición y Orquestación de Workers de IA

Una de las principales ventajas de la arquitectura de workers de IA es su capacidad de composición, que permite combinar y orquestar múltiples workers de IA para resolver problemas complejos. Al desglosar una tarea más grande en subtareas más pequeñas y manejables, cada una manejada por un worker de IA especializado, puede crear sistemas potentes y flexibles. En esta sección, exploraremos diferentes enfoques para componer y orquestar “una multitud” de workers de IA.

Encadenamiento de Workers de IA para Flujos de Trabajo Múltiples

En muchos escenarios, una tarea compleja puede descomponerse en una serie de pasos secuenciales, donde la salida de un worker de IA se convierte en la entrada del siguiente.

Este encadenamiento de workers de IA crea un flujo de trabajo o pipeline de múltiples pasos. Cada worker de IA en la cadena se enfoca en una subtarea específica, y la salida final es el resultado de los esfuerzos combinados de todos los workers.

Consideremos un ejemplo en el contexto de una aplicación Ruby on Rails para procesar contenido generado por usuarios. El flujo de trabajo involucra los siguientes pasos, que admitidamente son probablemente demasiado simples para valer la pena descomponer de esta manera en casos de uso reales, pero hacen que el ejemplo sea más fácil de entender:

1. **Limpieza de Texto:** Un worker de IA responsable de eliminar etiquetas HTML, convertir texto a minúsculas y manejar la normalización Unicode.
2. **Detección de Idioma:** Un worker de IA que identifica el idioma del texto limpio.
3. **Análisis de Sentimiento:** Un worker de IA que determina el sentimiento (positivo, negativo o neutral) del texto basado en el idioma detectado.
4. **Categorización de Contenido:** Un worker de IA que clasifica el texto en categorías predefinidas utilizando técnicas de procesamiento de lenguaje natural.

Aquí hay un ejemplo muy simplificado de cómo puede encadenar estos workers de IA utilizando Ruby:

```
1 class ContentProcessor
2   def initialize(text)
3     @text = text
4   end
5
6   def process
7     cleaned_text = TextCleanupWorker.new(@text).call
8     language = LanguageDetectionWorker.new(cleaned_text).call
9     sentiment = SentimentAnalysisWorker.new(cleaned_text, language).call
10    category = CategorizationWorker.new(cleaned_text, language).call
11
12    { cleaned_text:, language:, sentiment:, category: }
13  end
14end
```

En este ejemplo, la clase `ContentProcessor` se inicializa con el texto sin procesar y encadena los trabajadores de IA juntos en el método `process`. Cada trabajador de IA realiza su tarea específica y pasa el resultado al siguiente trabajador en la cadena. La salida final es un hash que contiene el texto limpio, el idioma detectado, el sentimiento y la categoría del contenido.

Procesamiento en Paralelo para Trabajadores de IA Independientes

En el ejemplo anterior, los trabajadores de IA están encadenados secuencialmente, donde cada trabajador procesa el texto y pasa el resultado al siguiente trabajador. Sin embargo, si tienes múltiples trabajadores de IA que pueden operar independientemente sobre la misma entrada, puedes optimizar el flujo de trabajo procesándolos en paralelo.

En el escenario dado, una vez que la limpieza del texto es realizada por el `TextCleanupWorker`, el `LanguageDetectionWorker`, `SentimentAnalysisWorker`, y `CategorizationWorker` pueden procesar el texto limpio de forma independiente. Al ejecutar estos trabajadores en paralelo, puedes potencialmente reducir el tiempo total de procesamiento y mejorar la eficiencia de tu flujo de trabajo.

Para lograr el procesamiento en paralelo en Ruby, puedes aprovechar técnicas de concurrencia como hilos o programación asíncrona. Aquí hay un ejemplo de cómo puedes modificar la clase `ContentProcessor` para procesar los tres trabajadores finales en paralelo usando hilos:

```
1  require 'concurrent'
2
3  class ContentProcessor
4    def initialize(text)
5      @text = text
6    end
7
8    def process
9      cleaned_text = TextCleanupWorker.new(@text).call
10
11    language_future = Concurrent::Future.execute do
12      LanguageDetectionWorker.new(cleaned_text).call
13    end
14
15    sentiment_future = Concurrent::Future.execute do
16      SentimentAnalysisWorker.new(cleaned_text).call
17    end
18
19    category_future = Concurrent::Future.execute do
20      CategorizationWorker.new(cleaned_text).call
21    end
22
23    language = language_future.value
24    sentiment = sentiment_future.value
25    category = category_future.value
26
27    { cleaned_text:, language:, sentiment:, category: }
28  end
29 end
```

En esta versión optimizada, utilizamos la biblioteca concurrent-ruby para crear objetos `Concurrent::Future` para cada uno de los trabajadores de IA independientes. Un `Future` representa un cálculo que se realizará de manera asíncrona en un hilo separado.

Después del paso de limpieza del texto, creamos tres objetos `Future`: `language_future`, `sentiment_future`, y `category_future`. Cada `Future` ejecuta su correspondiente trabajador de IA (`LanguageDetectionWorker`, `SentimentAnalysisWorker`, y `CategorizationWorker`) en un hilo separado,

pasando el `cleaned_text` como entrada.

Al llamar al método `value` en cada `Future`, esperamos a que se complete el cálculo y recuperamos el resultado. El método `value` bloquea hasta que el resultado esté disponible, asegurando que todos los trabajadores paralelos hayan terminado el procesamiento antes de continuar.

Finalmente, construimos el hash de salida con el texto limpio y los resultados de los trabajadores paralelos, al igual que en el ejemplo original.

Al procesar los trabajadores de IA independientes en paralelo, se puede reducir potencialmente el tiempo total de procesamiento en comparación con ejecutarlos secuencialmente. Esta optimización es particularmente beneficiosa cuando se trata de tareas que consumen mucho tiempo o cuando se procesan grandes volúmenes de datos.

Sin embargo, es importante tener en cuenta que las ganancias reales de rendimiento dependen de varios factores, como la complejidad de cada trabajador, los recursos del sistema disponibles y la sobrecarga de la gestión de hilos. Siempre es una buena práctica realizar pruebas de rendimiento y perfilar tu código para determinar el nivel óptimo de paralelismo para tu caso de uso específico.

Además, al implementar el procesamiento paralelo, hay que tener en cuenta cualquier recurso compartido o dependencia entre los trabajadores. Asegúrate de que los trabajadores puedan operar de forma independiente sin conflictos ni condiciones de carrera. Si hay dependencias o recursos compartidos, es posible que necesites implementar mecanismos de sincronización apropiados para mantener la integridad de los datos y evitar problemas como interbloqueos o resultados inconsistentes.

El Bloqueo del Intérprete Global de Ruby y el Procesamiento Asíncrono

Es importante entender las implicaciones del Bloqueo del Intérprete Global (GIL) de Ruby al considerar el procesamiento asíncrono basado en hilos en Ruby.

El GIL es un mecanismo en el intérprete de Ruby que asegura que solo un hilo puede ejecutar código Ruby a la vez, incluso en procesadores multinúcleo. Esto significa que aunque se pueden crear y gestionar múltiples hilos dentro de un proceso Ruby, solo un hilo puede ejecutar activamente código Ruby en cualquier momento dado.

El GIL está diseñado para simplificar la implementación del intérprete de Ruby y proporcionar seguridad entre hilos para las estructuras de datos internas de Ruby. Sin embargo, también limita el potencial de ejecución verdaderamente paralela del código Ruby.

Cuando usas hilos en Ruby, como con la biblioteca `concurrent-ruby` o la clase `Thread` incorporada, los hilos están sujetos a las restricciones del GIL. El GIL permite que cada hilo ejecute código Ruby durante un breve intervalo de tiempo antes de cambiar a otro hilo, creando la ilusión de ejecución concurrente.

Sin embargo, debido al GIL, la ejecución real del código Ruby sigue siendo secuencial. Mientras un hilo está ejecutando código Ruby, otros hilos están esencialmente en pausa, esperando su turno para adquirir el GIL y ejecutar.

Esto significa que el procesamiento asíncrono basado en hilos en Ruby es más efectivo para tareas limitadas por E/S, como esperar respuestas de APIs externas (como modelos de lenguaje grandes alojados por terceros) o realizar operaciones de E/S de archivos. Cuando un hilo encuentra una operación de E/S, puede liberar el GIL, permitiendo que otros hilos se ejecuten mientras espera que se complete la E/S.

Por otro lado, para tareas limitadas por CPU, como cálculos intensivos o procesamiento de trabajadores de IA de larga duración, el GIL puede limitar las posibles ganancias de rendimiento del paralelismo basado en hilos. Dado que solo un hilo puede ejecutar código Ruby a la vez, el tiempo total de ejecución puede no reducirse significativamente en comparación con el procesamiento secuencial.

Para lograr una ejecución verdaderamente paralela para tareas limitadas por CPU en

Ruby, es posible que necesites explorar enfoques alternativos, como:

- Usar paralelismo basado en procesos con múltiples procesos Ruby, cada uno ejecutándose en un núcleo de CPU separado.
- Aprovechar bibliotecas externas o frameworks que proporcionen extensiones nativas o interfaces a lenguajes sin GIL, como C o Rust.,
- Utilizar frameworks de computación distribuida o colas de mensajes para distribuir tareas entre múltiples máquinas o procesos.

Es crucial considerar la naturaleza de tus tareas y las limitaciones impuestas por el GIL al diseñar e implementar el procesamiento asíncrono en Ruby. Si bien el procesamiento asíncrono basado en hilos puede proporcionar beneficios para tareas limitadas por E/S, puede no ofrecer mejoras significativas de rendimiento para tareas limitadas por CPU debido a las restricciones del GIL.

Técnicas de Conjunto para Mejorar la Precisión

Las técnicas de conjunto implican combinar las salidas de múltiples trabajadores de IA para mejorar la precisión general o la robustez del sistema. En lugar de depender de un solo trabajador de IA, las técnicas de conjunto aprovechan la inteligencia colectiva de múltiples trabajadores para tomar decisiones más informadas.



Los ensambles son especialmente importantes si diferentes partes de tu flujo de trabajo funcionan mejor con diferentes modelos de IA, algo que es más común de lo que podrías pensar. Los modelos potentes como GPT-4 son extremadamente costosos en comparación con opciones de código abierto menos capaces, y probablemente no sean necesarios para cada paso del flujo de trabajo de tu aplicación.

Una técnica común de ensamble es la votación por mayoría, donde múltiples

trabajadores de IA procesan independientemente la misma entrada, y la salida final se determina por el consenso de la mayoría. Este enfoque puede ayudar a mitigar el impacto de errores individuales de los trabajadores y mejorar la fiabilidad general del sistema.

Consideremos un ejemplo donde tenemos tres trabajadores de IA para análisis de sentimiento, cada uno utilizando un modelo diferente o proporcionado con diferentes contextos. Podemos combinar sus salidas usando votación por mayoría para determinar la predicción final del sentimiento.

```
1 class SentimentAnalysisEnsemble
2   def initialize(text)
3     @text = text
4   end
5
6   def analyze
7     predictions = [
8       SentimentAnalysisWorker1.new(@text).analyze,
9       SentimentAnalysisWorker2.new(@text).analyze,
10      SentimentAnalysisWorker3.new(@text).analyze
11    ]
12
13    predictions
14      .group_by { |sentiment| sentiment }
15      .max_by { |_, votes| votes.size }
16      .first
17
18  end
19 end
```

En este ejemplo, la clase `SentimentAnalysisEnsemble` se inicializa con el texto e invoca tres diferentes trabajadores de IA para análisis de sentimientos. El método `analyze` recopila las predicciones de cada trabajador y determina el sentimiento mayoritario utilizando los métodos `group_by` y `max_by`. La salida final es el sentimiento que recibe más votos del conjunto de trabajadores.



Los conjuntos son claramente un caso donde experimentar con paralelismo puede valer la pena.

Selección Dinámica e Invocación de Trabajadores de IA

En algunos, si no en la mayoría de los casos, el trabajador de IA específico que se invocará puede depender de las condiciones de tiempo de ejecución o de las entradas del usuario. La selección dinámica y la invocación de trabajadores de IA permiten flexibilidad y adaptabilidad en el sistema.



Puede que te sientas tentado a tratar de incluir mucha funcionalidad en un solo trabajador de IA, dándole múltiples funciones y un mensaje inicial grande y complicado que explique cómo invocarlas. Resiste la tentación, créeme. Una de las razones por las que el enfoque que estamos discutiendo en este capítulo se llama “Multitud de Trabajadores” es para recordarnos que es deseable tener muchos trabajadores especializados, cada uno haciendo su pequeña tarea al servicio del propósito mayor.

Por ejemplo, considera una aplicación de chatbot donde diferentes trabajadores de IA son responsables de manejar diferentes tipos de consultas de usuario. Basándose en la entrada del usuario, la aplicación selecciona dinámicamente el trabajador de IA apropiado para procesar la consulta.

```
1 class ChatbotController < ApplicationController
2   def process_query
3     query = params[:query]
4     query_type = QueryClassifierWorker.new(query).classify
5
6     case query_type
7     when 'greeting'
8       response = GreetingWorker.new(query).generate_response
9     when 'product_inquiry'
10      response = ProductInquiryWorker.new(query).generate_response
11    when 'order_status'
12      response = OrderStatusWorker.new(query).generate_response
13    else
14      response = DefaultResponseWorker.new(query).generate_response
15    end
16
17    render json: { response: response }
18  end
19 end
```

En este ejemplo, el `ChatbotController` recibe una consulta del usuario a través de la acción `process_query`. Primero utiliza un `QueryClassifierWorker` para determinar el tipo de consulta. Basándose en el tipo de consulta clasificada, el controlador selecciona dinámicamente el trabajador de IA apropiado para generar la respuesta. Esta selección dinámica permite que el chatbot maneje diferentes tipos de consultas y las dirija a los trabajadores de IA relevantes.



Dado que el trabajo del `QueryClassifierWorker` es relativamente simple y no requiere mucho contexto o definiciones de funciones, probablemente puedas implementarlo usando un LLM pequeño y ultrarrápido como [mistralai/mistral-8x7b-instruct:nitro](#). Tiene capacidades que se acercan al nivel de GPT-4 en muchas tareas y, en el momento en que escribo esto, Groq puede servirlo con una impresionante velocidad de procesamiento de 444 tokens por segundo.

Combinando PLN Tradicional con MLGs

Si bien los Modelos de Lenguaje Grandes (MLGs) han revolucionado el campo del procesamiento del lenguaje natural (PLN), ofreciendo una versatilidad y rendimiento sin precedentes en una amplia gama de tareas, no siempre son la solución más eficiente o rentable para cada problema. En muchos casos, combinar técnicas tradicionales de PLN con MLGs puede llevar a enfoques más optimizados, específicos y económicos para resolver desafíos específicos de PLN.

Piensa en los MLGs como las navajas suizas del PLN: increíblemente versátiles y potentes, pero no necesariamente la mejor herramienta para cada trabajo. A veces, una herramienta dedicada como un sacacorchos o un abrebotellas puede ser más efectiva y eficiente para una tarea específica. De manera similar, las técnicas tradicionales de PLN, como el agrupamiento de documentos, la identificación de temas y la clasificación, pueden proporcionar soluciones más específicas y rentables para ciertos aspectos de tu pipeline de PLN.

Una de las principales ventajas de las técnicas tradicionales de PLN es su eficiencia computacional. Estos métodos, que a menudo se basan en modelos estadísticos más simples o enfoques basados en reglas, pueden procesar grandes volúmenes de datos textuales mucho más rápido y con menor sobrecarga computacional en comparación con los MLGs. Esto los hace particularmente adecuados para tareas que implican analizar y organizar grandes corpus de documentos, como agrupar artículos similares o identificar temas clave dentro de una colección de textos.

Además, las técnicas tradicionales de PLN pueden alcanzar a menudo una alta precisión y exactitud para tareas específicas, especialmente cuando se entrena con conjuntos de datos específicos del dominio. Por ejemplo, un clasificador de documentos bien ajustado que utilice algoritmos tradicionales de aprendizaje automático como Máquinas de Vectores de Soporte (MVS) o Naive Bayes puede categorizar documentos con precisión en categorías predefinidas con un costo computacional mínimo.

Sin embargo, los MLGs brillan verdaderamente cuando se trata de tareas que requieren una comprensión más profunda del lenguaje, el contexto y el razonamiento. Su capacidad para generar texto coherente y contextualmente relevante, responder preguntas y resumir pasajes largos no tiene igual en los métodos tradicionales de PLN. Los MLGs pueden manejar efectivamente fenómenos lingüísticos complejos, como la ambigüedad, la correferencia y las expresiones idiomáticas, haciéndolos invaluables para tareas que requieren generación o comprensión del lenguaje natural.

El verdadero poder radica en combinar las técnicas tradicionales de PLN con MLGs para crear enfoques híbridos que aprovechen las fortalezas de ambos. Al utilizar métodos tradicionales de PLN para tareas como el preprocesamiento de documentos, agrupamiento y extracción de temas, puedes organizar y estructurar tus datos textuales de manera eficiente. Esta información estructurada puede entonces alimentarse a los MLGs para tareas más avanzadas, como generar resúmenes, responder preguntas o crear informes completos.

Por ejemplo, consideremos un caso de uso donde deseas generar un informe de tendencias para un dominio específico basado en un gran corpus de documentos individuales de tendencias. En lugar de confiar únicamente en MLGs, que pueden ser computacionalmente costosos y consumir mucho tiempo para procesar grandes volúmenes de texto, puedes emplear un enfoque híbrido:

1. Utilizar técnicas tradicionales de PLN, como el modelado de temas (por ejemplo, Asignación Latente de Dirichlet) o algoritmos de agrupamiento (por ejemplo, K-means), para agrupar documentos de tendencias similares y identificar temas y tópicos clave dentro del corpus.
2. Alimentar los documentos agrupados y los temas identificados a un MLG, aprovechando sus capacidades superiores de comprensión y generación del lenguaje para crear resúmenes coherentes e informativos para cada grupo o tema.
3. Finalmente, utilizar el MLG para generar un informe completo de tendencias combinando los resúmenes individuales, destacando las tendencias más

significativas y proporcionando perspectivas y recomendaciones basadas en la información agregada.

Al combinar técnicas tradicionales de PLN con MLGs de esta manera, puedes procesar eficientemente grandes cantidades de datos textuales, extraer ideas significativas y generar informes de alta calidad mientras optimizas los recursos computacionales y los costos.

Al embarcarte en tus proyectos de PLN, es esencial evaluar cuidadosamente los requisitos y restricciones específicos de cada tarea y considerar cómo los métodos tradicionales de PLN y los LLMs pueden aprovecharse en conjunto para lograr los mejores resultados. Al combinar la eficiencia y precisión de las técnicas tradicionales con la versatilidad y potencia de los LLMs, puedes crear soluciones de PLN altamente efectivas y económicas que aporten valor a tus usuarios y partes interesadas.

Uso de Herramientas



En el ámbito del desarrollo de aplicaciones impulsadas por IA, el concepto de “uso de herramientas” o “llamada a funciones” ha surgido como una técnica poderosa que permite a su LLM conectarse a herramientas externas, APIs, funciones, bases de datos y otros recursos. Este enfoque permite un conjunto más rico de comportamientos que la simple generación de texto, y más interacciones dinámicas entre sus componentes de IA y el resto del ecosistema de su aplicación. Como examinaremos en este capítulo, el uso de herramientas también le da la opción de hacer que su modelo de IA genere datos de manera estructurada.

¿Qué es el Uso de Herramientas?

El uso de herramientas, también conocido como llamada a funciones, es una técnica que permite a los desarrolladores especificar una lista de funciones con las que un LLM

puede interactuar durante el proceso de generación. Estas herramientas pueden variar desde simples funciones de utilidad hasta APIs complejas o consultas a bases de datos. Al proporcionar al LLM acceso a estas herramientas, los desarrolladores pueden extender las capacidades del modelo y permitirle realizar tareas que requieren conocimiento o acciones externas.

Figura 8. Ejemplo de una definición de función para un trabajador de IA que analiza documentos

```
1  FUNCTION = {
2      name: "save_analysis",
3      description: "Save analysis data for document",
4      parameters: {
5          type: "object",
6          properties: {
7              title: {
8                  type: "string",
9                  maxLength: 140
10             },
11             summary: {
12                 type: "string",
13                 description: "comprehensive multi-paragraph summary with
14                               overview and list of sections (if applicable)"
15             },
16             tags: {
17                 type: "array",
18                 items: {
19                     type: "string",
20                     description: "lowercase tags representing main themes
21                               of the document"
22                 }
23             }
24         },
25         "required": %w[title summary tags]
26     }
27 }.freeze
```

La idea clave detrás del uso de herramientas es dar al LLM la capacidad de seleccionar y ejecutar dinámicamente las herramientas apropiadas según la entrada del usuario o la tarea en cuestión. En lugar de depender únicamente del conocimiento pre-entrenado

del modelo, el uso de herramientas permite al LLM aprovechar recursos externos para generar respuestas más precisas, relevantes y procesables. El uso de herramientas hace que técnicas como RAG (Generación Aumentada por Recuperación) sean mucho más fáciles de implementar de lo que serían de otro modo.

Ten en cuenta que, a menos que se indique lo contrario, este libro asume que tu modelo de IA no tiene acceso a ninguna herramienta incorporada del lado del servidor. Cualquier herramienta que deseas poner a disposición de tu IA debe ser declarada explícitamente por ti en cada solicitud de API, con provisiones para gestionar su ejecución si y cuando tu IA te indique que le gustaría usar esa herramienta en su respuesta.

El Potencial del Uso de Herramientas

El uso de herramientas abre un amplio abanico de posibilidades para las aplicaciones impulsadas por IA. Aquí hay algunos ejemplos de lo que se puede lograr con el uso de herramientas:

- 1. Chatbots y Asistentes Virtuales:** Al conectar un LLM con herramientas externas, los chatbots y asistentes virtuales pueden realizar tareas más complejas, como recuperar información de bases de datos, ejecutar llamadas a API o interactuar con otros sistemas. Por ejemplo, un chatbot podría usar una herramienta CRM para cambiar el estado de una operación según la solicitud del usuario.
- 2. Análisis de Datos y Perspectivas:** Los LLMs pueden conectarse a herramientas o bibliotecas de análisis de datos para realizar tareas avanzadas de procesamiento de datos. Esto permite a las aplicaciones generar perspectivas, realizar análisis comparativos o proporcionar recomendaciones basadas en datos en respuesta a consultas de usuarios.

3. **Búsqueda y Recuperación de Información:** El uso de herramientas permite a los LLMs interactuar con motores de búsqueda, bases de datos vectoriales u otros sistemas de recuperación de información. Al transformar las consultas de los usuarios en consultas de búsqueda, el LLM puede recuperar información relevante de múltiples fuentes y proporcionar respuestas completas a las preguntas de los usuarios.
4. **Integración con Servicios Externos:** El uso de herramientas permite una integración perfecta entre aplicaciones impulsadas por IA y servicios o APIs externos. Por ejemplo, un LLM podría interactuar con una API del clima para proporcionar actualizaciones meteorológicas en tiempo real o con una API de traducción para generar respuestas multilingües.

El Flujo de Trabajo del Uso de Herramientas

El flujo de trabajo del uso de herramientas típicamente involucra cuatro pasos clave:

1. Incluir definiciones de funciones en tu contexto de solicitud
2. Selección dinámica (o explícita) de herramientas
3. Ejecución de función(es)
4. Continuación opcional del prompt original

Revisemos cada uno de estos pasos en detalle.

Incluir definiciones de funciones en tu contexto de solicitud

La IA sabe qué herramientas tiene a su disposición porque tú le proporcionas una lista como parte de tu solicitud de completado (típicamente definida como funciones usando una variante del esquema JSON).

La sintaxis precisa de la definición de herramientas es específica del modelo.

Así es como defines una función `get_weather` en Claude 3:

```
1  {
2      "name": "get_weather",
3      "description": "Get the current weather in a given location",
4      "input_schema": {
5          "type": "object",
6          "properties": {
7              "location": {
8                  "type": "string",
9                  "description": "The city and state, e.g. San Francisco, CA"
10             },
11             "unit": {
12                 "type": "string",
13                 "enum": ["celsius", "fahrenheit"],
14                 "description": "The unit of temperature"
15             }
16         },
17         "required": ["location"]
18     }
19 }
```

Y así es como definirías la misma función para GPT-4, pasándola como el valor del parámetro `tools`:

```
1  {
2      "name": "get_current_weather",
3      "description": "Get the current weather in a given location",
4      "parameters": {
5          "type": "object",
6          "properties": {
7              "location": {
8                  "type": "string",
9                  "description": "The city and state, e.g. San Francisco, CA",
10             },
11             "unit": {
12                 "type": "string",
13                 "enum": ["celsius", "fahrenheit"],
14                 "description": "The unit of temperature"
15             }
16         },
17         "required": ["location"]
18     }
19 }
```

```
15      },
16      },
17      "required": ["location"],
18      },
19 }
```

¡Casi lo mismo, excepto diferente sin razón aparente! Qué molesto.

Las definiciones de funciones especifican nombre, descripción y parámetros de entrada. Los parámetros de entrada pueden definirse más detalladamente utilizando atributos como enumeraciones para limitar los valores aceptables, y especificando si un parámetro es obligatorio o no.

Además de las definiciones de funciones propiamente dichas, también puedes incluir instrucciones o contexto sobre por qué y cómo usar la función en la directiva del sistema.

Por ejemplo, mi herramienta de búsqueda web en Olympia incluye esta directiva del sistema, que le recuerda a la IA que tiene las herramientas mencionadas a su disposición:

```
1 The `google_search` and `realtime_search` functions let you do research
2 on behalf of the user. In contrast to Google, realtime search is powered
3 by Perplexity and provides real-time information to curated current events
4 databases and news sources. Make sure to include URLs in your response so
5 user can do followup research.
```

Proporcionar descripciones detalladas se considera el factor más importante en el rendimiento de las herramientas. Tus descripciones deberían explicar cada detalle sobre la herramienta, incluyendo:

- Qué hace la herramienta
- Cuándo debe usarse (y cuándo no)
- Qué significa cada parámetro y cómo afecta al comportamiento de la herramienta

- Cualquier advertencia o limitación importante que aplique a la implementación de la herramienta

Cuento más contexto puedes proporcionar a la IA sobre tus herramientas, mejor será su capacidad para decidir cuándo y cómo usarlas. Por ejemplo, Anthropic recomienda al menos 3-4 oraciones por descripción de herramienta para su serie Claude 3, más si la herramienta es compleja.

No es necesariamente intuitivo, pero las descripciones también se consideran más importantes que los ejemplos. Si bien puedes incluir ejemplos de cómo usar una herramienta en su descripción o en el prompt que la acompaña, esto es menos importante que tener una explicación clara y completa del propósito y los parámetros de la herramienta. Solo añade ejemplos después de haber desarrollado completamente la descripción.

Aquí hay un ejemplo de una especificación de función de API al estilo de Stripe:

```
1  {
2      "name": "createPayment",
3      "description": "Create a new payment request",
4      "parameters": {
5          "type": "object",
6          "properties": {
7              "transaction_amount": {
8                  "type": "number",
9                  "description": "The amount to be paid"
10             },
11             "description": {
12                 "type": "string",
13                 "description": "A brief description of the payment"
14             },
15             "payment_method_id": {
16                 "type": "string",
17                 "description": "The payment method to be used"
18             },
19             "payer": {
20                 "type": "object",
21                 "description": "Information about the payer, including their name,
```

```
22                     email, and identification number",
23     "properties": {
24         "name": {
25             "type": "string",
26             "description": "The payer's name"
27         },
28         "email": {
29             "type": "string",
30             "description": "The payer's email address"
31         },
32         "identification": {
33             "type": "object",
34             "description": "The payer's identification number",
35             "properties": {
36                 "type": {
37                     "type": "string",
38                     "description": "Identification document (e.g. CPF, CNPJ)"
39                 },
40                 "number": {
41                     "type": "string",
42                     "description": "The identification number"
43                 }
44             },
45             "required": [ "type", "number" ]
46         }
47     },
48     "required": [ "name", "email", "identification" ]
49 }
50 }
51 }
```



En la práctica, algunos modelos tienen dificultades para manejar especificaciones de funciones anidadas y tipos de datos de salida complejos como arreglos, diccionarios, etc. Pero en teoría, ¡deberías poder proporcionar especificaciones de JSON Schema de profundidad arbitraria!

Selección Dinámica de Herramientas

Cuando ejecutas un completado de chat que incluye definiciones de herramientas, el LLM selecciona dinámicamente la(s) herramienta(s) más apropiada(s) para usar y genera los parámetros de entrada requeridos para cada herramienta.

En la práctica, la capacidad de la IA para llamar *exactamente* a la función correcta y seguir *exactamente* tu especificación para las entradas es variable. Reducir el hiperparámetro de temperatura a 0.0 ayuda mucho, pero por experiencia aún verás errores ocasionales. Esos fallos incluyen nombres de funciones alucinados, parámetros de entrada mal nombrados o simplemente faltantes. Los parámetros se pasan como JSON, lo que significa que a veces verás errores causados por JSON truncado, mal citado o dañado de alguna otra manera.



Los patrones de [Self Healing Data](#) pueden ayudar a [corregir automáticamente](#) las llamadas a funciones que fallan debido a errores de sintaxis.

Selección Forzada (o Explícita) de Herramientas

Algunos modelos te dan la opción de forzar la llamada de una función particular, como un parámetro en la solicitud. De lo contrario, la decisión de llamar o no a la función queda completamente a discreción de la IA.

La capacidad de forzar una llamada a función es crucial en ciertos escenarios donde deseas asegurar que una herramienta o función específica se ejecute, independientemente del proceso de selección dinámica de la IA. Hay varias razones por las que esta capacidad es importante:

1. **Control Explícito:** Puedes estar usando la IA como un *Componente Discreto* o en un flujo de trabajo predefinido que requiere la ejecución de una función particular

en un momento específico. Al forzar la llamada, puedes garantizar que la función deseada sea invocada en lugar de tener que pedírselo amablemente a la IA.

2. **Depuración y Pruebas:** Durante el desarrollo y prueba de aplicaciones impulsadas por IA, la capacidad de forzar llamadas a funciones es invaluable para propósitos de depuración. Al activar explícitamente funciones específicas, puedes aislar y probar componentes individuales de tu aplicación. Esto te permite verificar la corrección de las implementaciones de funciones, validar los parámetros de entrada y asegurar que se devuelvan los resultados esperados.
3. **Manejo de Casos Límite:** Puede haber casos límite o escenarios excepcionales donde el proceso de selección dinámica de la IA podría no elegir ejecutar una función que debería, y tú lo sabes basado en procesos externos. En tales casos, tener la capacidad de forzar una llamada a función te permite manejar estas situaciones explícitamente. Define reglas o condiciones en la lógica de tu aplicación para determinar cuándo anular la discreción de la IA.
4. **Consistencia y Reproducibilidad:** Si tienes una secuencia específica de funciones que necesitan ejecutarse en un orden particular, forzar las llamadas garantiza que se siga la misma secuencia cada vez. Esto es especialmente importante en aplicaciones donde la consistencia y el comportamiento predecible son críticos, como en sistemas financieros o simulaciones científicas.
5. **Optimización del Rendimiento:** En algunos casos, forzar una llamada a función puede llevar a optimizaciones de rendimiento. Si sabes que una función específica es necesaria para una tarea particular y que el proceso de selección dinámica de la IA podría introducir sobrecarga innecesaria, puedes evitar el proceso de selección e invocar directamente la función requerida. Esto puede ayudar a reducir la latencia y mejorar la eficiencia general de tu aplicación.

En resumen, la capacidad de forzar llamadas a funciones en aplicaciones impulsadas por IA proporciona control explícito, ayuda en la depuración y pruebas, maneja casos límite, asegura consistencia y reproducibilidad. Es una herramienta poderosa en tu arsenal, pero necesitamos discutir un aspecto más de esta importante característica.



En muchos casos de uso de toma de decisiones, siempre queremos que el modelo haga una llamada a función y puede que nunca queramos que el modelo responda solo con su conocimiento interno. Por ejemplo, si estás enrutando entre múltiples modelos especializados en diferentes tareas (entrada multilingüe, matemáticas, etc.), puedes usar el modelo de llamada a funciones para delegar solicitudes a uno de los modelos auxiliares y nunca responder independientemente.

Parámetro de Selección de Herramienta

GPT-4 y otros modelos de lenguaje que soportan llamadas a funciones te dan un parámetro `tool_choice` para controlar si el uso de herramientas es requerido como parte de un completado. Este parámetro tiene tres valores posibles:

- `auto` da a la IA total discreción sobre usar una herramienta o simplemente responder
- `required` le dice a la IA que *debe* llamar a una herramienta *en lugar de* responder, pero deja la selección de la herramienta a criterio de la IA
- La tercera opción es establecer el parámetro del `name_of_function` que deseas forzar. Más sobre esto en la siguiente sección.



Ten en cuenta que si estableces `tool choice` como `required`, el modelo se verá forzado a elegir la función más relevante para llamar entre las proporcionadas, incluso si ninguna se ajusta realmente al prompt. Al momento de la publicación, no conozco ningún modelo que devuelva una respuesta `tool_calls` vacía, o que use alguna otra forma de hacerte saber que no encontró una función adecuada para llamar.

Forzar una Función Para Obtener Salida Estructurada

La capacidad de forzar una llamada a función te proporciona una manera de obtener datos estructurados de una completación de chat en lugar de tener que extraerlos tú mismo de su respuesta en texto plano.

¿Por qué es tan importante forzar funciones para obtener salida estructurada? En pocas palabras, porque la extracción de datos estructurados de la salida de un LLM es un dolor de cabeza. Puedes hacer tu vida un poco más fácil pidiendo los datos en XML, pero entonces tienes que analizar XML. ¿Y qué haces cuando ese XML falta porque tu IA respondió: “Lo siento, pero no puedo generar los datos que solicitaste porque bla, bla, bla...”?

Cuando uses herramientas de esta manera:

- Probablemente deberías definir una única herramienta en tu solicitud
- Recuerda forzar el uso de su función utilizando el parámetro `tool_choice`.
- Recuerda que el modelo pasará la entrada a la herramienta, por lo que el nombre de la herramienta y la descripción deben ser desde la perspectiva del modelo, no la tuya.

Este último punto merece un ejemplo para mayor claridad. Digamos que estás pidiendo a la IA que haga análisis de sentimiento sobre texto de usuario. El nombre de la función no sería `analyze_sentiment`, sino algo como `save_sentiment_analysis`. La IA es quien realiza el análisis de sentimiento, *no la herramienta*. Todo lo que la herramienta está haciendo (desde la perspectiva de la IA) es guardar los resultados del análisis.

Aquí hay un ejemplo de cómo usar Claude 3 para registrar un resumen de una imagen en JSON bien estructurado, esta vez desde la línea de comandos usando `curl`:

```
1 curl https://api.anthropic.com/v1/messages \
2   --header "content-type: application/json" \
3   --header "x-api-key: $ANTHROPIC_API_KEY" \
4   --header "anthropic-version: 2023-06-01" \
5   --header "anthropic-beta: tools-2024-04-04" \
6   --data \
7   '{
8     "model": "claude-3-sonnet-20240229",
9     "max_tokens": 1024,
10    "tools": [
11      {
12        "name": "record_summary",
13        "description": "Record summary of image into well-structured JSON.",
14        "input_schema": {
15          "type": "object",
16          "properties": {
17            "key_colors": {
18              "type": "array",
19              "items": {
20                "type": "object",
21                "properties": {
22                  "r": {
23                    "type": "number",
24                    "description": "red value [0.0, 1.0]"
25                  },
26                  "g": {
27                    "type": "number",
28                    "description": "green value [0.0, 1.0]"
29                  },
30                  "b": {
31                    "type": "number",
32                    "description": "blue value [0.0, 1.0]"
33                  },
34                  "name": {
35                    "type": "string",
36                    "description": "Human-readable color name
37                               in snake_case, e.g.
38                               \"olive_green\" or
39                               \"turquoise\""
40                  }
41                },
42                "required": [ "r", "g", "b", "name" ]
43              },
44            }
45          }
46        }
47      ],
48      "output_schema": {
49        "type": "object",
50        "properties": {
51          "summary": {
52            "type": "string",
53            "description": "A well-structured JSON object containing the key_colors and their corresponding names."}
54        }
55      }
56    }
57  ]'
58 }
```

```
43         "description": "Key colors in the image. Four or less."
44     },
45     "description": {
46         "type": "string",
47         "description": "Image description. 1-2 sentences max."
48     },
49     "estimated_year": {
50         "type": "integer",
51         "description": "Estimated year that the image was taken,
52                         if it is a photo. Only set this if the
53                         image appears to be non-fictional.
54                         Rough estimates are okay!"
55     }
56 },
57     "required": [ "key_colors", "description" ]
58 }
59 ],
60     "messages": [
61     {
62         "role": "user",
63         "content": [
64             {
65                 "type": "image",
66                 "source": {
67                     "type": "base64",
68                     "media_type": "'$IMAGE_MEDIA_TYPE'",
69                     "data": "'$IMAGE_BASE64'"
70                 }
71             },
72             {
73                 "type": "text",
74                 "text": "Use `record_summary` to describe this image."
75             }
76         ]
77     }
78 ]
79 }'
```

En el ejemplo proporcionado, estamos utilizando el modelo Claude 3 de Anthropic para generar un resumen JSON estructurado de una imagen. Así es como funciona:

1. Definimos una única herramienta llamada `record_summary` en el array `tools` de la carga útil de la solicitud. Esta herramienta es responsable de registrar un resumen de la imagen en formato JSON bien estructurado.
2. La herramienta `record_summary` tiene un `input_schema` que especifica la estructura esperada de la salida JSON. Define tres propiedades:
 - `key_colors`: Un array de objetos que representa los colores clave en la imagen. Cada objeto de color tiene propiedades para los valores rojo, verde y azul (que van de 0.0 a 1.0) y un nombre de color legible por humanos en formato `snake_case`.
 - `description`: Una propiedad de tipo string para una breve descripción de la imagen, limitada a 1-2 oraciones.
 - `estimated_year`: Una propiedad entera opcional para el año estimado en que se tomó la imagen, si parece ser una foto no ficticia.
3. En el array `messages`, proporcionamos los datos de la imagen como una cadena codificada en base64 junto con el tipo de medio. Esto permite que el modelo procese la imagen como parte de la entrada.
4. También indicamos a Claude que use la herramienta `record_summary` para describir la imagen.
5. Cuando la solicitud se envía al modelo Claude 3, este analiza la imagen y genera un resumen JSON basado en el `input_schema` especificado. El modelo extrae los colores clave, proporciona una breve descripción y estima el año en que se tomó la imagen (si corresponde).
6. El resumen JSON generado se pasa como parámetros a la herramienta `record_summary`, proporcionando una representación estructurada de las características clave de la imagen.

Al utilizar la herramienta `record_summary` con un `input_schema` bien definido, podemos obtener un resumen JSON estructurado de una imagen sin depender de la

extracción de texto plano. Este enfoque asegura que la salida siga un formato consistente y pueda ser fácilmente analizada y procesada por los componentes posteriores de la aplicación.

La capacidad de forzar una llamada a función y especificar la estructura de salida esperada es una característica poderosa del uso de herramientas en aplicaciones basadas en IA. Permite a los desarrolladores tener más control sobre la salida generada y simplifica la integración de datos generados por IA en el flujo de trabajo de su aplicación.

Ejecución de Función(es)

Has definido funciones y has indicado a tu IA, que decidió que debería llamar a una de tus funciones. Ahora es el momento de que tu código de aplicación o biblioteca, si estás usando una gema Ruby como [raix-rails](#), envíe la llamada a función y sus parámetros a la implementación correspondiente *en tu código de aplicación*.

Tu código de aplicación decide qué hacer con los resultados de la ejecución de la función. Tal vez lo que hay que hacer implica una sola línea de código en un lambda, o tal vez implica llamar a una API externa. Tal vez implica llamar a otro componente de IA, o tal vez implica cientos o incluso miles de líneas de código en el resto de tu sistema. Depende completamente de ti.

A veces la llamada a función es el final de la operación, pero si los resultados representan información en una cadena de pensamiento que debe ser continuada por la IA, entonces tu código de aplicación necesita insertar los resultados de la ejecución en la transcripción del chat y dejar que la IA continúe el procesamiento.

Por ejemplo, aquí hay una declaración de función de [Raix](#) utilizada por el AccountManager de Olympia para comunicarse con nuestros clientes como parte de una Orquestación de Flujo de Trabajo Inteligente para servicio al cliente.

```
1 class AccountManager
2   include Raix::ChatCompletion
3   include Raix::FunctionDispatch
4
5   # lots of other functions...
6
7   function :notify_account_owner,
8     "Don't share UUID. Mention dollars if subscription changed",
9     message: { type: "string" } do |arguments|
10    account.owner.freeform_notify(
11      subject: "Account Change Notification",
12      message: arguments[:message]
13    )
14    "Notified account owner"
15 end
```

Puede que no esté inmediatamente claro qué está sucediendo aquí, así que lo desglosaremos.

1. La clase `AccountManager` define muchas funciones relacionadas con la gestión de cuentas. Puede cambiar tu plan, agregar y eliminar miembros del equipo, entre otras cosas.
2. Sus instrucciones de nivel superior le indican a `AccountManager` que debe notificar al propietario de la cuenta sobre los resultados de la solicitud de cambio de cuenta, utilizando la función `notify_account_owner`.
3. La definición concisa de la función incluye:
 - nombre
 - descripción
 - parámetros `message: { type: "string" }`
 - un bloque para ejecutar cuando se llama a la función

Después de actualizar la transcripción con los resultados del bloque de función, se llama nuevamente al método `chat_completion`. Este método es responsable de enviar

la transcripción actualizada de la conversación de vuelta al modelo de IA para su procesamiento posterior. Nos referimos a este proceso como un *bucle de conversación*.

Cuando el modelo de IA recibe una nueva solicitud de finalización de chat con una transcripción actualizada, tiene acceso a los resultados de la función ejecutada previamente. Puede analizar estos resultados, incorporarlos a su proceso de toma de decisiones y generar la siguiente respuesta o acción basándose en el contexto acumulativo de la conversación. Puede elegir ejecutar funciones adicionales basándose en el contexto actualizado, o puede generar una respuesta final al prompt original si determina que no son necesarias más llamadas a funciones.

Continuación Opcional del Prompt Original

Cuando envías los resultados de las herramientas de vuelta al LLM y continúas procesando el prompt original, la IA utiliza esos resultados para llamar a funciones adicionales o generar una respuesta final en texto plano.



Algunos modelos como [Command-R](#) de Cohere pueden citar las herramientas específicas que utilizaron en sus respuestas, proporcionando transparencia y trazabilidad adicional.

Dependiendo del modelo en uso, los resultados de la llamada a la función vivirán en mensajes de transcripción que tienen su propio rol especial o se reflejarán en alguna otra sintaxis. Pero la parte importante es que esos datos estén en la transcripción, para que la IA pueda considerarlos al decidir qué hacer a continuación.



Un error común (y potencialmente costoso) es olvidar agregar los resultados de la función a la transcripción antes de continuar el chat. Como resultado, la IA será consultada esencialmente de la misma manera que antes de llamar a la función por primera vez. En otras palabras, desde el punto de vista de la IA, aún no ha llamado a la función. Así que la llama de nuevo. Y otra vez. Y otra vez, indefinidamente hasta que la interrumpas. ¡Esperemos que tu contexto no fuera demasiado grande y tu modelo no fuera demasiado costoso!

Mejores Prácticas para el Uso de Herramientas

Para aprovechar al máximo el uso de herramientas, considera las siguientes mejores prácticas.

Definiciones Descriptivas

Proporciona nombres y descripciones claras y descriptivas para cada herramienta y sus parámetros de entrada. Esto ayuda al LLM a comprender mejor el propósito y las capacidades de cada herramienta.

Por experiencia puedo decirte que la sabiduría común que dice que “nombrar es difícil” se aplica aquí; he visto resultados dramáticamente diferentes de los LLMs solo cambiando los nombres de las funciones o la redacción de las descripciones. A veces, eliminar descripciones *mejora* el rendimiento.

Procesamiento de Resultados de Herramientas

Al pasar los resultados de las herramientas de vuelta al LLM, asegúrate de que estén bien estructurados y sean completos. Utiliza claves y valores significativos para representar la salida de cada herramienta. Experimenta con diferentes formatos y observa cuál funciona mejor, desde JSON hasta texto plano.

El [Intérprete de Resultados](#) aborda este desafío empleando IA para analizar los resultados y proporcionar explicaciones, resúmenes o conclusiones clave comprensibles para humanos.

Manejo de Errores

Implementa mecanismos robustos de manejo de errores para manejar casos donde el LLM pueda generar parámetros de entrada inválidos o no soportados para las llamadas a herramientas. Maneja y recupérate de manera elegante de cualquier error que pueda ocurrir durante la ejecución de la herramienta.

¡Una cualidad sumamente agradable de la IA es que entiende los mensajes de error! Lo que significa que si estás trabajando con una mentalidad rápida y práctica, simplemente puedes capturar cualquier excepción generada en la implementación de una herramienta y pasársela a la IA para que sepa lo que sucedió!

Por ejemplo, aquí hay una versión simplificada de la implementación de la búsqueda de Google en Olympia:

```
1  def google_search(conversation, params)
2    conversation.update_cstatus("Searching Google...")
3    query = params[:query]
4    search = GoogleSearch.new(query).get_hash
5
6    conversation.update_cstatus("Summarizing results...")
7    SummarizeKnowledgeGraph.new.perform(conversation, search.to_json)
8  rescue StandardError => e
9    Honeybadger.notify(e)
10   { error: e.message }.inspect
11 end
```

Las búsquedas de Google en Olympia son un proceso de dos pasos. Primero se realiza la búsqueda, luego se resumen los resultados. Si hay un fallo, sin importar cuál sea, el mensaje de excepción se empaqueta y se envía de vuelta a la IA. Esta técnica es el fundamento de prácticamente todos los patrones de *Manejo Inteligente de Errores*.

Por ejemplo, supongamos que la llamada a la API GoogleSearch falla debido a una excepción 503 de Servicio No Disponible. Esto se propaga hasta el nivel superior de rescate, y la descripción del error se envía de vuelta a la IA como resultado de la llamada a la función. En lugar de mostrar al usuario una pantalla en blanco o un error técnico, la IA dice algo como “Lo siento, pero no puedo acceder a mis capacidades de búsqueda de Google en este momento. Puedo intentarlo más tarde, si lo deseas.”

Esto puede parecer solo un truco ingenioso, pero considera un tipo diferente de error, uno donde la IA estaba llamando a una API externa y tenía control directo sobre los parámetros a pasar a la API. ¿Quizás cometió un error en cómo generó esos parámetros? Siempre que el mensaje de error de la API externa sea lo suficientemente detallado, devolver el mensaje de error a la IA que realiza la llamada significa que puede reconsiderar esos parámetros e intentarlo de nuevo. Automáticamente. Sin importar cuál fue el error.

Ahora piensa en lo que se necesitaría para replicar ese tipo de manejo robusto de errores en código *normal*. Es prácticamente imposible.

Refinamiento Iterativo

Si el LLM no está recomendando las herramientas apropiadas o está generando respuestas subóptimas, itera en las definiciones de herramientas, descripciones y parámetros de entrada. Refina y mejora continuamente la configuración de las herramientas basándote en el comportamiento observado y los resultados deseados.

1. Comienza con definiciones simples de herramientas: Empieza definiendo herramientas con nombres, descripciones y parámetros de entrada claros y concisos. Evita complicar excesivamente la configuración de las herramientas inicialmente y concéntrate en la funcionalidad principal. Por ejemplo, si quieres guardar los resultados de un análisis de sentimiento, comienza con una definición básica como:

```
1  {
2      "name": "save_sentiment_score",
3      "description": "Analyze user-provided text and generate sentiment score",
4      "parameters": {
5          "type": "object",
6          "properties": {
7              "score": {
8                  "type": "float",
9                  "description": "sentiment score from -1 (negative) to 1 (positive)"
10             }
11         },
12         "required": ["score"]
13     }
14 }
```

2. Probar y observar: Una vez que tenga las definiciones iniciales de las herramientas establecidas, pruébelas con diferentes instrucciones y observe cómo el MLG interactúa con la herramienta. Preste atención a la calidad y relevancia de las respuestas generadas. Si el MLG está generando respuestas subóptimas, es momento de refinar las definiciones de las herramientas.

3. Refinar descripciones: Si el MLG está malinterpretando el propósito de una herramienta, intente refinar la descripción de la misma. Proporcione más contexto, ejemplos o aclaraciones para guiar al MLG en el uso efectivo de la herramienta. Por ejemplo, puede actualizar la descripción de la herramienta de análisis de sentimientos para abordar más específicamente el *tono emocional* del texto que se está analizando:

```
1  {
2    "name": "save_sentiment_score",
3    "description": "Determine the overall emotional tone of a piece of text,
4    such as customer reviews, social media posts, or feedback comments.",
5    ...
6 }
```

4. Ajustar los parámetros de entrada: Si el LLM está generando parámetros de entrada inválidos o irrelevantes para una herramienta, considera ajustar las definiciones de los parámetros. Añade restricciones más específicas, reglas de validación o ejemplos para clarificar el formato de entrada esperado.
5. Iterar basándose en la retroalimentación: Monitorea continuamente el rendimiento de tus herramientas y recopila comentarios de los usuarios o partes interesadas. Utiliza esta retroalimentación para identificar áreas de mejora y realizar refinamientos iterativos en las definiciones de las herramientas. Por ejemplo, si los usuarios informan que el análisis no está manejando bien el sarcasmo, puedes añadir una nota en la descripción:

```
1  {
2      "name": "save_sentiment_score",
3      "description": "Analyze the sentiment of a given text and return a sentiment
4          score between -1 (negative) and 1 (positive). Note: Sarcasm should be
5          considered negative.",
6      ...
7 }
```

Mediante el refinamiento iterativo de sus definiciones de herramientas basado en el comportamiento observado y la retroalimentación, puede mejorar gradualmente el rendimiento y la efectividad de su aplicación basada en IA. Recuerde mantener las definiciones de herramientas claras, concisas y enfocadas en la tarea específica en cuestión. Pruebe y valide regularmente las interacciones de las herramientas para asegurarse de que se alineen con los resultados deseados.

Composición y Encadenamiento de Herramientas

Uno de los aspectos más poderosos del uso de herramientas que solo se ha mencionado hasta ahora es la capacidad de componer y encadenar múltiples herramientas para realizar tareas complejas. Al diseñar cuidadosamente las definiciones de sus herramientas y sus formatos de entrada/salida, puede crear bloques de construcción reutilizables que se pueden combinar de varias maneras.

Consideremos un ejemplo donde está construyendo un pipeline de análisis de datos para su aplicación basada en IA. Podría tener las siguientes herramientas:

1. **DataRetrieval**: Una herramienta que obtiene datos de una base de datos o API según criterios específicos.
2. **DataProcessing**: Una herramienta que realiza cálculos, transformaciones o agregaciones en los datos recuperados.

3. **DataVisualization:** Una herramienta que presenta los datos procesados en un formato amigable para el usuario, como gráficos o diagramas.

Al encadenar estas herramientas, puede crear un flujo de trabajo poderoso que recupera datos relevantes, los procesa y presenta los resultados de manera significativa. Así es como podría verse el flujo de trabajo del uso de herramientas:

1. El LLM recibe una consulta del usuario solicitando información sobre datos de ventas para una categoría específica de productos.
2. El LLM selecciona la herramienta `DataRetrieval` y genera los parámetros de entrada apropiados para obtener los datos de ventas relevantes de la base de datos.
3. Los datos recuperados se “pasan” a la herramienta `DataProcessing`, que calcula métricas como ingresos totales, precio promedio de venta y tasa de crecimiento.
4. Los datos procesados son entonces digeridos por la herramienta `DataVisualization`, que crea un gráfico o diagrama visualmente atractivo para representar la información, devolviendo la URL del gráfico al LLM.
5. Finalmente, el LLM genera una respuesta formateada a la consulta del usuario usando markdown, incorporando los datos visualizados y proporcionando un resumen de los hallazgos clave.

Al componer estas herramientas juntas, puede crear un flujo de trabajo de análisis de datos sin problemas que se puede integrar fácilmente en su aplicación. La belleza de este enfoque es que cada herramienta puede desarrollarse y probarse de forma independiente, y luego combinarse de diferentes maneras para resolver varios problemas.

Para permitir una composición y encadenamiento fluido de herramientas, es importante definir formatos claros de entrada y salida para cada herramienta.

Por ejemplo, la herramienta `DataRetrieval` podría aceptar parámetros como los detalles de conexión a la base de datos, nombre de tabla y condiciones de consulta, y devolver el conjunto de resultados como un objeto JSON estructurado. La herramienta

DataProcessing puede entonces esperar este objeto JSON como entrada y producir un objeto JSON transformado como salida. Al estandarizar el flujo de datos entre herramientas, puede asegurar la compatibilidad y la reutilización.

Mientras diseña su ecosistema de herramientas, piense en cómo se pueden combinar diferentes herramientas para abordar casos de uso comunes en su aplicación. Considere crear herramientas de alto nivel que encapsulen flujos de trabajo comunes o lógica de negocio, facilitando que el LLM las seleccione y use de manera efectiva.

Recuerde, el poder del uso de herramientas radica en la flexibilidad y modularidad que proporciona. Al desglosar tareas complejas en herramientas más pequeñas y reutilizables, puede crear una aplicación robusta y adaptable basada en IA que puede abordar una amplia gama de desafíos.

Direcciones Futuras

A medida que evoluciona el campo del desarrollo de aplicaciones basadas en IA, podemos esperar más avances en las capacidades de uso de herramientas. Algunas direcciones futuras potenciales incluyen:

- 1. Uso de Herramientas Múltiples:** Los LLM pueden ser capaces de decidir cuántas veces necesitan usar herramientas para generar una respuesta satisfactoria. Esto podría involucrar múltiples rondas de selección y ejecución de herramientas basadas en resultados intermedios.
- 2. Herramientas Predefinidas:** Las plataformas de IA pueden proporcionar un conjunto de herramientas predefinidas que los desarrolladores pueden aprovechar de inmediato, como intérpretes de Python, herramientas de búsqueda web o funciones de utilidad comunes.
- 3. Integración Perfecta:** A medida que el uso de herramientas se vuelve más prevalente, podemos esperar una mejor integración entre las plataformas de

IA y los marcos de desarrollo populares, facilitando a los desarrolladores la incorporación del uso de herramientas en sus aplicaciones.

El uso de herramientas es una técnica poderosa que permite a los desarrolladores aprovechar todo el potencial de los LLM en aplicaciones basadas en IA. Al conectar LLM con herramientas y recursos externos, puede crear sistemas más dinámicos, inteligentes y conscientes del contexto que pueden adaptarse a las necesidades del usuario y proporcionar información y acciones valiosas.

Si bien el uso de herramientas ofrece inmensas posibilidades, es importante ser consciente de los posibles desafíos y consideraciones. Un aspecto clave es gestionar la complejidad de las interacciones entre herramientas y asegurar la estabilidad y confiabilidad del sistema en general. Necesita manejar escenarios donde las llamadas a herramientas pueden fallar, devolver resultados inesperados o tener implicaciones de rendimiento. Además, debe considerar medidas de seguridad y control de acceso para prevenir el uso no autorizado o malicioso de herramientas. Los mecanismos adecuados de manejo de errores, registro y monitoreo son cruciales para mantener la integridad y el rendimiento de su aplicación basada en IA.

Mientras exploras las posibilidades del uso de herramientas en tus propios proyectos, recuerda comenzar con objetivos claros, diseñar definiciones de herramientas bien estructuradas e iterar basándote en la retroalimentación y los resultados. Con el enfoque y la mentalidad correctos, el uso de herramientas puede desbloquear nuevos niveles de innovación y valor en tus aplicaciones impulsadas por IA.

Procesamiento de Flujo



El procesamiento de flujo de datos a través de HTTP, también conocido como eventos enviados por el servidor (SSE), es un mecanismo donde el servidor envía datos continuamente al cliente a medida que están disponibles, sin necesidad de que el cliente los solicite explícitamente. Como la respuesta de la IA se genera de manera incremental, tiene sentido proporcionar una experiencia de usuario receptiva mostrando la salida de la IA mientras se genera. Y de hecho, todas las APIs de proveedores de IA que conozco ofrecen respuestas en streaming como una opción en sus puntos de conexión de completado.

La razón por la que este capítulo aparece aquí en el libro, justo después de [Using Tools](#), es debido a lo poderoso que puede ser combinar el uso de herramientas con respuestas de IA en vivo para los usuarios. Esto permite experiencias dinámicas e interactivas donde la IA puede procesar la entrada del usuario, utilizar varias herramientas y funciones a su discreción, y luego proporcionar respuestas en tiempo real.

Para lograr esta interacción fluida, necesitas escribir manejadores de flujo que puedan despachar tanto las llamadas a funciones de herramientas invocadas por la IA como la salida de texto plano al usuario final. La necesidad de realizar un bucle después de procesar una función de herramienta añade un desafío interesante al trabajo.

Implementando un ReplyStream

Para demostrar cómo se puede implementar el procesamiento de flujo, este capítulo realizará un análisis profundo de una versión simplificada de la clase `ReplyStream` que se utiliza en Olympia. Las instancias de esta clase pueden pasarse como el parámetro `stream` en bibliotecas cliente de IA como [ruby-openai](#) y [openrouter](#).

Así es como uso `ReplyStream` en el `PromptSubscriber` de Olympia, que escucha a través de Wisper la creación de nuevos mensajes de usuario.

```
1 class PromptSubscriber
2   include Raix::ChatCompletion
3   include Raix::PromptDeclarations
4
5   # many other declarations omitted...
6
7   prompt text: -> { user_message.content },
8     stream: -> { ReplyStream.new(self) },
9     until: -> { bot_message.complete? }
10
11 def message_created(message) # invoked by Wisper
12   return unless message.role.user? && message.content?
13
14 # rest of the implementation omitted...
```

Además de una referencia context al suscriptor de prompt que lo instanció, la clase `ReplyStream` también tiene variables de instancia para almacenar un búfer de datos recibidos y matrices para realizar un seguimiento de los nombres de funciones y argumentos invocados durante el procesamiento del flujo.

```
1 class ReplyStream
2   attr_accessor :buffer, :f_name, :f_arguments, :context
3
4   delegate :bot_message, :dispatch, to: :context
5
6   def initialize(context)
7     self.context = context
8     self.buffer = []
9     self.f_name = []
10    self.f_arguments = []
11  end
12
13  def call(chunk, bytesize = nil)
14    # ...
15  end
16
17  # ...
18 end
```

El método `initialize` configura el estado inicial de la instancia de `ReplyStream`, inicializando el búfer, el contexto y otras variables.

El método `call` es el punto de entrada principal para procesar los datos en streaming. Recibe un `chunk` de datos (representado como un hash) y un parámetro opcional `bytesize`, que en nuestro ejemplo no se utiliza. Dentro de este método, la clase utiliza coincidencia de patrones para manejar diferentes escenarios basados en la estructura del fragmento recibido.



Llamar a `deep_symbolize_keys` en el fragmento ayuda a hacer que la coincidencia de patrones sea más elegante, permitiéndonos operar con símbolos en lugar de cadenas de texto.

```
1 def call(chunk, _bytesize)
2     case chunk.deep_symbolize_keys
3
4     in { # match function name
5         choices: [
6             {
7                 delta: {
8                     tool_calls: [
9                         { index: index, function: {name: name} }
10                    ]
11                 }
12             }
13         ]
14     }
15     f_name[index] = name
```

El primer patrón que estamos buscando es una llamada a herramienta junto con su nombre de función asociado. Si detectamos uno, lo guardamos en el array `f_name`. Almacenamos los nombres de funciones en un array indexado, ya que el modelo es capaz de realizar llamadas a funciones en paralelo, enviando más de una función para ejecutar al mismo tiempo.

La llamada paralela a funciones es la capacidad de un modelo de IA para realizar múltiples llamadas a funciones juntas, permitiendo que los efectos y resultados de estas llamadas a funciones se resuelvan en paralelo. Esto es especialmente útil si las funciones tardan mucho tiempo, y reduce los viajes de ida y vuelta con la API, lo que a su vez puede ahorrar una cantidad significativa de gasto de tokens.

A continuación, necesitamos buscar los argumentos correspondientes a las llamadas a funciones.

```

1   in { # match arguments
2     choices: [
3       {
4         delta: {
5           tool_calls: [
6             {
7               index: index, function: {arguments: argument }
8             }
9           ]
10          }
11        }
12      ]}
13
14   f_arguments[index] ||= "" # initialize if not already
15   f_arguments[index] << argument

```

De manera similar a como manejamos los nombres de funciones, guardamos los argumentos en un array indexado.

A continuación, buscamos los mensajes destinados al usuario, que llegarán desde el servidor un token a la vez y se asignarán a la variable `new_content`. También necesitamos estar atentos al `finish_reason`. Este será `nil` hasta el último fragmento de la secuencia de salida.

```

1   in {
2     choices: [
3       { delta: {content: new_content}, finish_reason: finish_reason }
4     ]}
5
6     # you could transmit every chunk to the user here...
7     buffer << new_content.to_s
8
9     if finish_reason.present?
10       finalize
11     elsif new_content.to_s.match?(/\n\n/)
12       send_to_client # ...or buffer and transmit once per paragraph
13     end

```

De manera importante, agregamos una expresión de coincidencia de patrones para manejar los mensajes de error enviados por el proveedor del modelo de IA. En los

entornos de desarrollo local, lanzamos una excepción, pero en producción, registramos el error y finalizamos.

```
1  in { error: { message: } }  
2    if Rails.env.local?  
3      raise message  
4    else  
5      Honeybadger.notify("AI Error: #{message}")  
6      finalize  
7    end
```

La cláusula else final del case se ejecutará si ninguno de los patrones anteriores coincidió. Es simplemente una medida de seguridad para que nos enteremos si el modelo de IA comienza a enviarnos fragmentos no reconocidos.

```
1  else  
2    Honeybadger.notify("Unrecognized Chunk: #{chunk}")  
3  end  
4 end
```

El método `send_to_client` es responsable de enviar el contenido almacenado en el búfer al cliente. Verifica que el búfer no esté vacío, actualiza el contenido del mensaje del bot, renderiza el mensaje del bot y guarda el contenido en la base de datos para asegurar la persistencia de datos.

```

1 def send_to_client
2   # no need to process pure whitespace
3   return if buffer.join.squish.blank?
4
5   # set the buffer content on the bot message
6   content = buffer.join
7   bot_message.content = content
8
9   # save to database so that we never lose data
10  # even if the stream doesn't terminate correctly
11  bot_message.update_column(:content, content)
12
13  # update content via websocket
14  ConversationRenderer.update(bot_message)
15 end

```

El método `finalize` se llama cuando el procesamiento del stream está completo. Despacha las llamadas a funciones si se recibieron algunas durante el stream, actualiza el mensaje del bot con el contenido final y otra información relevante, y reinicia el historial de llamadas a funciones

```

1 def finalize
2   if f_name.any?
3     f_name.each_with_index do |name, index|
4       # takes care of calling the function wherever it's implemented
5       dispatch(name:, arguments: JSON.parse(f_arguments[index]))
6     end
7
8     # reset the function call history
9     f_name.clear
10    f_arguments.clear
11  else
12    content = buffer.join.presence
13    bot_message.update!(content:, complete: true)
14    ConversationRenderer.update(bot_message)
15  end
16 end

```

Si el modelo decide llamar a una función, necesitas “despachar” esa llamada a función

(nombre y argumentos) de tal manera que se ejecute y los mensajes de `function_call` y `function_result` se añadan a la transcripción de la conversación

En mi experiencia, es mejor manejar la creación de mensajes de función en un solo lugar de tu base de código, en lugar de depender de las implementaciones de herramientas. Es más limpio, pero también tiene una razón práctica muy importante: si el modelo de IA llama a una función, y no ve los mensajes resultantes de la llamada y el resultado en la transcripción cuando haces el bucle, *volverá a llamar a la misma función*. Potencialmente para siempre. Recuerda que la IA es completamente sin estado, así que a menos que hagas eco de esas llamadas a funciones de vuelta a ella, es como si nunca hubieran ocurrido.

```
1 # PromptSubscriber#dispatch
2
3 def dispatch(name:, arguments:)
4   # adds a function_call message to the conversation transcript
5   # plus dispatches to tool and returns result
6   conversation.function_call!(name, arguments).then do |result|
7     # add function result message to the transcript
8     conversation.function_result!(name, result)
9   end
10 end
```



Limpiar el historial de llamadas a funciones después del envío es tan importante como asegurarse de que la llamada y los resultados terminen en tu transcripción, para que no sigas llamando las mismas funciones una y otra vez cada vez que iteras el bucle.

El “Bucle de Conversación”

I sólo mencionando el bucle, pero si eres nuevo en las llamadas a funciones, puede que no sea obvio *por qué* necesitamos hacer un bucle. La razón es que una vez que la IA te “pide”

ejecutar funciones de herramientas en su nombre, dejará de responder. Depende de ti ejecutar esas funciones, recopilar los resultados, añadir los resultados a la transcripción y luego enviar el prompt original nuevamente para obtener un nuevo conjunto de llamadas a funciones o resultados dirigidos al usuario.

En la clase `PromptSubscriber`, usamos el método `prompt` del módulo `PromptDeclarations` para definir el comportamiento del bucle de conversación. El parámetro `until` está configurado como `-> { bot_message.complete? }`, lo que significa que el bucle continuará hasta que el `bot_message` esté marcado como completo.

```
1 prompt text: -> { user_message.content },
2       stream: -> { ReplyStream.new(self) },
3       until: -> { bot_message.complete? }
```



Pero, ¿cuándo se marca `bot_message` como completo? Si lo has olvidado, consulta la línea 13 del método `finalize`.

Repasemos toda la lógica del procesamiento de stream.

1. El `PromptSubscriber` recibe un nuevo mensaje del usuario a través del método `message_created`, que es invocado por el sistema pub/sub Wisper cada vez que el usuario final crea un nuevo prompt.
2. El método de clase `prompt` define de manera declarativa el comportamiento de la lógica de completado de chat para el `PromptSubscriber`. El modelo de IA ejecutará un completado de chat con el contenido del mensaje del usuario, una nueva instancia de `ReplyStream` como parámetro de `stream`, y la condición de bucle especificada.
3. El modelo de IA procesa el prompt y comienza a generar una respuesta. A medida que se transmite la respuesta, se invoca el método `call` de la instancia `ReplyStream` para cada fragmento de datos.

4. Si el modelo de IA decide llamar a una función de herramienta, el nombre de la función y los argumentos se extraen del fragmento y se almacenan en los arrays `f_name` y `f_arguments`, respectivamente.
5. Si el modelo de IA genera contenido visible para el usuario, este se almacena en búfer y se envía al cliente a través del método `send_to_client`.
6. Una vez que se completa el procesamiento del stream, se llama al método `finalize`. Si se invocaron funciones de herramienta durante el stream, estas se despachan utilizando el método `dispatch` del `PromptSubscriber`.
7. El método `dispatch` agrega un mensaje `function_call` a la transcripción de la conversación, ejecuta la función de herramienta correspondiente y agrega un mensaje `function_result` a la transcripción con el resultado de la llamada a la función.
8. Después de despachar las funciones de herramienta, se limpia el historial de llamadas a funciones para evitar llamadas duplicadas en bucles subsiguientes.
9. Si no se invocaron funciones de herramienta, el método `finalize` actualiza el `bot_message` con el contenido final, lo marca como completo y envía el mensaje actualizado al cliente.
10. Se evalúa la condición de bucle `-> { bot_message.complete? }`. Si el `bot_message` no está marcado como completo, el bucle continúa y el prompt original se envía nuevamente con la transcripción de conversación actualizada.
11. Los pasos 3-10 se repiten hasta que el `bot_message` se marca como completo, lo que indica que el modelo de IA ha terminado de generar su respuesta y no es necesario ejecutar más funciones de herramienta.

Al implementar este bucle de conversación, permite que el modelo de IA participe en una interacción bidireccional con la aplicación, ejecutando funciones de herramienta según sea necesario y generando respuestas visibles para el usuario hasta que la conversación llegue a una conclusión natural.

La combinación del procesamiento de stream y el bucle de conversación permite experiencias dinámicas e interactivas impulsadas por IA, donde el modelo de IA puede

procesar la entrada del usuario, utilizar varias herramientas y funciones, y proporcionar respuestas en tiempo real basadas en el contexto evolutivo de la conversación.

Continuación Automática

Es importante tener en cuenta las limitaciones de la salida de IA. La mayoría de los modelos tienen un número máximo de tokens que pueden generar en una sola respuesta, que está determinado por el parámetro `max_tokens`. Si el modelo de IA alcanza este límite mientras genera una respuesta, se detendrá abruptamente e indicará que la salida fue truncada.

En la respuesta de streaming de la API de la plataforma de IA, puedes detectar esta situación examinando la variable `finish_reason` en el fragmento. Si la `finish_reason` está configurada como "length" (o algún otro valor clave específico del modelo), significa que el modelo alcanzó su límite máximo de tokens durante la generación y la salida se ha cortado.

Una forma de manejar este escenario de manera elegante y proporcionar una experiencia de usuario fluida es implementar un mecanismo de continuación automática en tu lógica de procesamiento de stream. Al agregar un patrón de coincidencia para las razones de finalización relacionadas con la longitud, puedes optar por hacer un bucle y continuar automáticamente la salida desde donde se quedó.

Aquí hay un ejemplo intencionadamente simplificado de cómo puedes modificar el método `call` en la clase `ReplyStream` para admitir la continuación automática:

```
1 LENGTH_STOPS = %w[length MAX_TOKENS]
2
3 def call(chunk, _bytesize)
4   case chunk.deep_symbolize_keys
5     # ...
6
7   in {
8     choices: [
9       { delta: {content: new_content},
10         finish_reason: finish_reason } ] }
11
12   buffer << new_content.to_s
13
14   if finish_reason.blank?
15     send_to_client if new_content.to_s.match?(/\n\n/)
16   elsif LENGTH_STOPS.include?(finish_reason)
17     continue_cutoff
18   else
19     finalize
20   end
21
22   # ...
23 end
24 end
25
26 private
27
28 def continue_cutoff
29   conversation.bot_message!(buffer.join, visible: false)
30   conversation.user_message!("please continue", visible: false)
31   bot_message.update_column(:created_at, Time.current)
32 end
```

En esta versión modificada, cuando el `finish_reason` indica una salida truncada, en lugar de finalizar el stream, agregamos un par de mensajes a la transcripción sin finalizarla, movemos el mensaje original orientado al usuario al “fondo” de la transcripción actualizando su atributo `created_at`, y luego dejamos que ocurra el bucle, para que la IA continúe generando donde se quedó.

Recuerda que el endpoint de completado de la IA es sin estado. Solo “sabe” lo que

le dices a través de la transcripción. En este caso, la forma en que comunicamos a la IA que fue interrumpida es agregando mensajes “invisibles” (para el usuario final) a la transcripción. Recuerda, sin embargo, que este es un ejemplo intencionalmente simplificado. Una implementación real necesitaría realizar una gestión adicional de la transcripción para asegurar que no desperdiciemos tokens y/o confundamos a la IA con mensajes duplicados del asistente en la transcripción.

Una implementación real de auto-continuación también debería tener la llamada “lógica de interruptor automático” implementada para prevenir bucles descontrolados. La razón es que, dados ciertos tipos de indicaciones del usuario y configuraciones bajas de `max_tokens`, la IA podría continuar generando salidas orientadas al usuario indefinidamente.

Ten en cuenta que cada bucle requiere una solicitud separada, y que cada solicitud consume tu transcripción completa nuevamente. Definitivamente deberías considerar el equilibrio entre la experiencia del usuario y el uso de la API al decidir si implementar la auto-continuación en tu aplicación. La auto-continuación en particular puede ser peligrosamente costosa, especialmente cuando se utilizan modelos comerciales premium.

Conclusión

El procesamiento de streams es un aspecto crítico en la construcción de aplicaciones impulsadas por IA que combinan el uso de herramientas con respuestas de IA en tiempo real. Al manejar eficientemente los datos en streaming desde las APIs de plataformas de IA, puedes proporcionar una experiencia de usuario fluida e interactiva, manejar respuestas grandes, optimizar el uso de recursos y gestionar errores con elegancia.

La clase `Conversation::ReplyStream` proporcionada demuestra cómo el

procesamiento de streams puede ser implementado en una aplicación Ruby utilizando coincidencia de patrones y arquitectura dirigida por eventos. Al comprender y aprovechar las técnicas de procesamiento de streams, puedes desbloquear todo el potencial de la integración de IA en tus aplicaciones y ofrecer experiencias de usuario poderosas y atractivas.

Datos Autorreparables



Los datos autorreparables son un enfoque poderoso para garantizar la integridad, consistencia y calidad de los datos en las aplicaciones aprovechando las capacidades de los modelos de lenguaje grande (LLMs). Esta categoría de patrones se centra en la idea de utilizar la IA para detectar, diagnosticar y corregir automáticamente anomalías, inconsistencias o errores en los datos, reduciendo así la carga sobre los desarrolladores y manteniendo un alto nivel de fiabilidad de los datos.

En su esencia, los patrones de datos autorreparables reconocen que los datos son el elemento vital de cualquier aplicación, y garantizar su precisión e integridad es crucial para el funcionamiento adecuado y la experiencia del usuario de la aplicación. Sin embargo, gestionar y mantener la calidad de los datos puede ser una tarea compleja y que consume mucho tiempo, especialmente cuando las aplicaciones crecen en tamaño y complejidad. Aquí es donde entra en juego el poder de la IA.

En los patrones de datos autorreparables, se emplean trabajadores de IA para monitorear y analizar continuamente los datos de tu aplicación. Estos modelos tienen la capacidad de comprender e interpretar patrones, relaciones y anomalías dentro de los datos. Al aprovechar sus capacidades de procesamiento y comprensión del lenguaje natural, pueden identificar posibles problemas o inconsistencias en los datos y tomar las acciones apropiadas para rectificarlos.

El proceso de datos autorreparables típicamente involucra varios pasos clave:

1. **Monitoreo de Datos:** Los trabajadores de IA monitorean constantemente los flujos de datos, bases de datos o sistemas de almacenamiento de la aplicación, buscando cualquier señal de anomalías, inconsistencias o errores. Alternativamente, puedes activar un componente de IA en reacción a una excepción.
2. **Detección de Anomalías:** Cuando se detecta un problema, el trabajador de IA analiza los datos en detalle para identificar la naturaleza específica y el alcance del problema. Esto podría involucrar la detección de valores faltantes, formatos inconsistentes o datos que violan reglas o restricciones predefinidas.
3. **Diagnóstico y Corrección:** Una vez que se identifica el problema, el trabajador de IA utiliza su conocimiento y comprensión del dominio de datos para determinar el curso de acción apropiado. Esto podría involucrar la corrección automática de los datos, el llenado de valores faltantes o el marcado del problema para intervención humana si es necesario.
4. **Aprendizaje Continuo (opcional, dependiendo del caso de uso):** A medida que tu trabajador de IA encuentra y resuelve varios problemas de datos, puede generar salidas que describan lo que sucedió y cómo respondió. Estos metadatos pueden alimentar procesos de aprendizaje que te permiten (y quizás al modelo subyacente, mediante ajuste fino) volverte más efectivo y eficiente con el tiempo en la identificación y resolución de anomalías en los datos.

Al detectar y corregir automáticamente los problemas de datos, puedes asegurar que tu

aplicación opere con datos de alta calidad y confiables. Esto reduce el riesgo de errores, inconsistencias o fallos relacionados con los datos que afecten la funcionalidad o la experiencia del usuario de la aplicación.

Una vez que tienes trabajadores de IA manejando la tarea de monitoreo y corrección de datos, puedes enfocar tus esfuerzos en otros aspectos críticos de la aplicación. Esto ahorra tiempo y recursos que de otro modo se gastarían en limpieza y mantenimiento manual de datos. De hecho, a medida que tus aplicaciones crecen en tamaño y complejidad, gestionar manualmente la calidad de los datos se vuelve cada vez más desafiante. Los patrones de “Datos Autorreparables” escalan efectivamente al aprovechar el poder de la IA para manejar grandes volúmenes de datos y detectar problemas en tiempo real.



Debido a su naturaleza, los modelos de IA pueden adaptarse a patrones de datos cambiantes, esquemas o requisitos a lo largo del tiempo con poca o ninguna supervisión. Siempre que sus directivas proporcionen una guía adecuada, especialmente con respecto a los resultados previstos, tu aplicación puede evolucionar y manejar nuevos escenarios de datos sin requerir una intervención manual extensiva o cambios en el código.

Los patrones de datos autorreparables se alinean bien con otras categorías de patrones que hemos discutido, como “Multitud de Trabajadores”. La capacidad de datos autorreparables puede verse como un tipo especializado de trabajador que se centra específicamente en asegurar la calidad e integridad de los datos. Este tipo de trabajador opera junto con otros trabajadores de IA, cada uno contribuyendo a diferentes aspectos de la funcionalidad de la aplicación.

Implementar patrones de datos autorreparables en la práctica requiere un diseño cuidadoso y la integración de modelos de IA en la arquitectura de la aplicación. Debido a los riesgos de pérdida y corrupción de datos, debes definir pautas claras sobre cómo utilizarás esta técnica. También debes considerar factores como el rendimiento, la escalabilidad y la seguridad de los datos.

Caso Práctico: Arreglando JSON Dañado

Una de las formas más prácticas y convenientes de aprovechar los datos autorreparables es también muy simple de explicar: arreglar JSON dañado.

Esta técnica se puede aplicar al desafío común de manejar datos imperfectos o inconsistentes generados por LLMs, como JSON dañado, y proporciona un enfoque para detectar y corregir automáticamente estos problemas.

En Olympia me encuentro regularmente con escenarios donde los LLM generan datos JSON que no son perfectamente válidos. Esto puede ocurrir por varias razones, como cuando el LLM añade comentarios antes o después del código JSON real, o introduce errores de sintaxis como comas faltantes o comillas dobles sin escapar. Estos problemas pueden provocar errores de análisis sintáctico y causar interrupciones en la funcionalidad de la aplicación.

Para abordar este problema, he implementado una solución práctica en forma de una clase `JsonFixer`. Esta clase incorpora el patrón “Self-Healing Data” (datos autorreparables) tomando el JSON defectuoso como entrada y aprovechando un LLM para corregirlo mientras preserva la mayor cantidad posible de información e intención.

```
1 class JsonFixer
2     include Raix::ChatCompletion
3
4     def call(bad_json, error_message)
5         raise "No data provided" if bad_json.blank? || error_message.blank?
6
7         transcript << {
8             system: "Consider user-provided JSON that generated a parse
9                 exception. Do your best to fix it while preserving the
10                original content and intent as much as possible."
11            transcript << { user: bad_json }
12            transcript << { assistant: "What is the error message?" }
13            transcript << { user: error_message }
14            transcript << { assistant: "Here is the corrected JSON\n```\njson\n```" }
```

```
16     self.stop = [ "^\n^" ]  
17  
18     chat_completion(json: true)  
19   end  
20  
21   def model  
22     "mistralai/mixtral-8x7b-instruct:nitro"  
23   end  
24 end
```



Observe cómo JsonFixer utiliza [Ventriloquist](#) para guiar las respuestas de la IA.

El proceso de datos JSON autocorrectivos funciona de la siguiente manera:

1. **Generación de JSON:** Se utiliza un LLM para generar datos JSON basados en ciertos prompts o requerimientos. Sin embargo, debido a la naturaleza de los LLMs, el JSON generado puede no ser siempre perfectamente válido. El analizador JSON, por supuesto, generará un `ParserError` si se le proporciona un JSON inválido.

```
1 begin  
2   JSON.parse(l1m_generated_json)  
3 rescue JSON::ParserError => e  
4   JsonFixer.new.call(l1m_generated_json, e.message)  
5 end
```

Tenga en cuenta que el mensaje de excepción también se pasa a la llamada `JsonFixer` para que no necesite asumir completamente qué está mal con los datos, especialmente porque el analizador sintáctico a menudo te dirá exactamente qué está mal.

2. **Corrección basada en LLM:** La clase `JsonFixer` envía el JSON defectuoso de vuelta a un LLM, junto con un prompt o instrucción específica para corregir el JSON mientras preserva la información y la intención original tanto como

sea posible. El LLM, entrenado con grandes cantidades de datos y con un entendimiento de la sintaxis JSON, intenta corregir los errores y generar una cadena JSON válida. Se utiliza [Response Fencing](#) para restringir la salida del LLM, y elegimos Mixtral 8x7B como modelo de IA, ya que es particularmente bueno para este tipo de tarea.

3. **Validación e Integración:** La cadena JSON corregida devuelta por el LLM es analizada por la propia clase `JSONFixer`, porque llamó a `chat_completion(json: true)`. Si el JSON corregido pasa la validación, se integra de nuevo en el flujo de trabajo de la aplicación, permitiendo que la aplicación continúe procesando los datos sin problemas. El JSON defectuoso ha sido “sanado”.

Aunque he escrito y reescrito mi propia implementación de `JSONFixer` varias veces, dudo que el tiempo total invertido en todas esas versiones sea más de una o dos horas.

Tenga en cuenta que la preservación de la intención es un elemento clave de cualquier patrón de datos auto-reparables. El proceso de corrección basado en LLM tiene como objetivo preservar la información original y la intención del JSON generado tanto como sea posible. Esto asegura que el JSON corregido mantiene su significado semántico y puede ser utilizado efectivamente dentro del contexto de la aplicación.

Esta implementación práctica del enfoque de “Self-Healing Data” en Olympia demuestra claramente cómo la IA, específicamente los LLMs, puede ser aprovechada para resolver desafíos de datos del mundo real. Muestra el poder de combinar técnicas de programación tradicionales con capacidades de IA para construir aplicaciones robustas y eficientes.

La Ley de Postel y el Patrón “Self-Healing Data”

“Self-Healing Data”, como se ejemplifica en la clase JSONFixer, se alinea bien con el principio conocido como la Ley de Postel, también referido como el Principio de Robustez. La Ley de Postel establece:

“Sé conservador en lo que haces, sé liberal en lo que aceptas de otros.”

Este principio, originalmente articulado por Jon Postel, un pionero de los primeros días de Internet, enfatiza la importancia de construir sistemas que sean tolerantes con entradas diversas o incluso ligeramente incorrectas mientras mantienen una estricta adherencia a los protocolos especificados al enviar salidas.

En el contexto de “Self-Healing Data”, la clase JSONFixer encarna la Ley de Postel al ser liberal en la aceptación de datos JSON rotos o imperfectos generados por LLMs. No rechaza ni falla inmediatamente cuando encuentra JSON que no se adhiere estrictamente al formato esperado. En su lugar, toma un enfoque tolerante e intenta corregir el JSON usando el poder de los LLMs.

Al ser liberal en la aceptación de JSON imperfecto, la clase JSONFixer demuestra robustez y flexibilidad. Reconoce que los datos en el mundo real a menudo vienen en varias formas y pueden no siempre conformarse a especificaciones estrictas. Al manejar y corregir estas desviaciones con elegancia, la clase asegura que la aplicación pueda continuar funcionando sin problemas, incluso en presencia de datos imperfectos.

Por otro lado, la clase JSONFixer también se adhiere al aspecto conservador de la Ley de Postel cuando se trata de la salida. Después de corregir el JSON usando LLMs, la clase valida el JSON corregido para asegurar que se ajuste estrictamente al formato esperado. Mantiene la integridad y corrección de los datos antes de pasarllos a otras partes de la aplicación. Este enfoque conservador garantiza que la salida de la clase JSONFixer sea confiable y consistente, promoviendo la interoperabilidad y previniendo la propagación de errores.

Datos interesantes sobre Jon Postel:

- Jon Postel (1943-1998) fue un científico computacional estadounidense que jugó un papel crucial en el desarrollo de Internet. Era conocido como el “Dios de Internet” por sus significativas contribuciones a los protocolos y estándares fundamentales.
- Postel fue el editor de la serie de documentos Request for Comments (RFC), que es una serie de notas técnicas y organizativas sobre Internet. Escribió o co-escribió más de 200 RFCs, incluyendo los protocolos fundamentales como TCP, IP y SMTP.
- Además de sus contribuciones técnicas, Postel era conocido por su enfoque humilde y colaborativo. Creía en la importancia de alcanzar consensos y trabajar juntos para construir una red robusta e interoperable.
- Postel sirvió como Director de la División de Redes de Computadoras en el Instituto de Ciencias de la Información (ISI) de la Universidad del Sur de California (USC) desde 1977 hasta su prematura muerte en 1998.
- En reconocimiento a sus inmensas contribuciones, Postel fue galardonado póstumamente con el prestigioso Premio Turing en 1998, a menudo referido como el “Premio Nobel de la Computación.”

La clase `JSONFixer` promueve la robustez, flexibilidad e interoperabilidad, que fueron valores fundamentales que Postel defendió durante toda su carrera. Al construir sistemas que son tolerantes a las imperfecciones mientras mantienen una estricta adherencia a los protocolos, podemos crear aplicaciones que son más resilientes y adaptables frente a los desafíos del mundo real.

Consideraciones y Contraindicaciones

La aplicabilidad de los enfoques de datos auto-reparables depende completamente del tipo de datos que maneja tu aplicación. Hay una razón por la que podrías no querer simplemente hacer un monkeypatch de `JSON.parse` para corregir automáticamente

todos los errores de análisis JSON en tu aplicación: no todos los errores pueden o deben ser corregidos automáticamente.

La auto-reparación es particularmente delicada cuando está vinculada a requisitos regulatorios o de cumplimiento relacionados con el manejo y procesamiento de datos. Algunas industrias, como la sanitaria y la financiera, tienen regulaciones tan estrictas sobre la integridad y auditabilidad de los datos que realizar cualquier tipo de corrección de datos en “caja negra” sin la supervisión o registro adecuados puede violar estas regulaciones. Es crucial asegurar que cualquier técnica de datos auto-reparables que desarrolles se alinee con los marcos legales y regulatorios aplicables.

La aplicación de técnicas de datos auto-reparables, particularmente aquellas que involucran modelos de IA, también puede tener un gran impacto en el rendimiento de la aplicación y la utilización de recursos. Procesar grandes volúmenes de datos a través de modelos de IA para la detección y corrección de errores puede ser computacionalmente intensivo. Es importante evaluar el equilibrio entre los beneficios de los datos auto-reparables y los costos asociados de rendimiento y recursos.

Dicho esto, profundicemos en los factores involucrados en decidir cuándo y dónde aplicar este poderoso enfoque.

Criticidad de los Datos

Al considerar la aplicación de técnicas de datos auto-reparables, es crucial evaluar la criticidad de los datos que se están procesando. El nivel de criticidad se refiere a la importancia y sensibilidad de los datos en el contexto de tu aplicación y su dominio de negocio.

En algunos casos, corregir automáticamente errores de datos puede no ser apropiado, especialmente si los datos son altamente sensibles o tienen implicaciones legales. Por ejemplo, considera los siguientes escenarios:

1. **Transacciones Financieras:** En aplicaciones financieras, como sistemas bancarios

o plataformas de trading, la precisión de los datos es de suma importancia. Incluso errores menores en datos financieros pueden tener consecuencias significativas, como saldos de cuenta incorrectos, fondos mal dirigidos o decisiones de trading erróneas. En estos casos, las correcciones automatizadas sin una verificación y auditoría exhaustivas pueden introducir riesgos inaceptables.

2. **Registros Médicos:** Las aplicaciones de salud manejan datos de pacientes altamente sensibles y confidenciales. Las inexactitudes en los registros médicos pueden tener graves implicaciones para la seguridad del paciente y las decisiones de tratamiento. Modificar automáticamente datos médicos sin la supervisión y validación adecuada por profesionales de la salud calificados puede violar requisitos regulatorios y poner en riesgo el bienestar del paciente.
3. **Documentos Legales:** Las aplicaciones que manejan documentos legales, como contratos, acuerdos o presentaciones judiciales, requieren una estricta precisión e integridad. Incluso errores menores en datos legales pueden tener ramificaciones legales significativas. Las correcciones automatizadas en este dominio pueden no ser apropiadas, ya que los datos a menudo requieren revisión manual y verificación por expertos legales para asegurar su validez y ejecutabilidad.

En estos escenarios de datos críticos, los riesgos asociados con las correcciones automatizadas a menudo superan los beneficios potenciales. Las consecuencias de introducir errores o modificar datos incorrectamente pueden ser graves, llevando a pérdidas financieras, responsabilidades legales o incluso daños a individuos.

Cuando se trata de datos altamente críticos, es esencial priorizar los procesos de verificación y validación manual. La supervisión humana y la experiencia son cruciales para asegurar la precisión e integridad de los datos. Las técnicas de auto-reparación automatizadas aún pueden emplearse para señalar posibles errores o inconsistencias, pero la decisión final sobre las correcciones debe involucrar el juicio y la aprobación humana.

Sin embargo, es importante notar que no todos los datos en una aplicación pueden tener

el mismo nivel de criticidad. Dentro de la misma aplicación, puede haber subconjuntos de datos que son menos sensibles o tienen menor impacto si ocurren errores. En tales casos, las técnicas de datos auto-reparables pueden aplicarse selectivamente a esos subconjuntos específicos de datos, mientras que los datos críticos permanecen sujetos a verificación manual.

La clave es evaluar cuidadosamente la criticidad de cada categoría de datos en tu aplicación y definir pautas y procesos claros para manejar correcciones basadas en los riesgos e implicaciones asociados. Al diferenciar entre datos críticos (es decir, libros contables, registros médicos) y no críticos (es decir, direcciones postales, advertencias de recursos), puedes encontrar un equilibrio entre aprovechar los beneficios de las técnicas de datos auto-reparables donde sea apropiado y mantener un control y supervisión estrictos donde sea necesario.

En última instancia, la decisión de aplicar técnicas de datos auto-reparables a datos críticos debe tomarse en consulta con expertos del dominio, asesores legales y otros interesados relevantes. Es esencial considerar los requisitos específicos, las regulaciones y los riesgos asociados con los datos de tu aplicación y alinear las estrategias de corrección de datos en consecuencia.

Severidad del Error

Al aplicar técnicas de datos auto-reparables, es importante evaluar la severidad y el impacto de los errores en los datos. No todos los errores son iguales, y el curso de acción apropiado puede variar dependiendo de la severidad del problema.

Las inconsistencias menores o problemas de formato pueden ser adecuados para la corrección automática. Por ejemplo, un trabajador de datos auto-reparables encargado de arreglar JSON defectuoso puede manejar comas faltantes o comillas dobles sin escapar sin alterar significativamente el significado o la estructura de los datos. Estos tipos de errores son a menudo sencillos de corregir y tienen un impacto mínimo en la integridad general de los datos.

Sin embargo, los errores más graves que cambian fundamentalmente el significado o la integridad de los datos pueden requerir un enfoque diferente. En tales casos, las correcciones automatizadas pueden no ser suficientes, y la intervención humana puede ser necesaria para garantizar la precisión y validez de los datos.

Aquí es donde entra en juego el concepto de usar la IA misma para ayudar a determinar la gravedad del error. Al aprovechar las capacidades de los modelos de IA, podemos diseñar trabajadores de datos autorreparables que no solo corrijan errores, sino que también evalúen la gravedad de estos errores y tomen decisiones informadas sobre cómo manejarlos.

Por ejemplo, consideremos un trabajador de datos autorreparable responsable de corregir inconsistencias en el flujo de datos hacia una base de datos de clientes. El trabajador puede ser diseñado para analizar los datos e identificar posibles errores, como información faltante o contradictoria. Sin embargo, en lugar de corregir automáticamente todos los errores, el trabajador puede estar equipado con llamadas a herramientas adicionales que le permitan marcar los errores graves para revisión humana.

Aquí hay un ejemplo de cómo se puede implementar esto:

```
1 class CustomerDataReviewer
2   include Raix::ChatCompletion
3   include Raix::FunctionDeclarations
4
5   attr_accessor :customer
6
7   function :flag_for_review, reason: { type: "string" } do |params|
8     AdminNotifier.review_request(customer, params[:reason])
9   end
10
11  def initialize(customer)
12    self.customer = customer
13  end
14
15  def call(customer_data)
16    transcript << {
```

```
17     system: "You are a customer data reviewer. Your task is to identify
18         and correct inconsistencies in customer data.
19
20         < additional instructions here... >
21
22         If you encounter severe errors that require human review, use the
23         `flag_for_review` tool to flag the data for manual intervention." }
24
25     transcript << { user: customer.to_json }
26     transcript << { assistant: "Reviewed/corrected data:\n```\n"
27
28     self.stop = ["```"]
29
30     chat_completion(json: true).then do |result|
31       return if result.blank?
32
33       customer.update(result)
34     end
35   end
36 end
```

En este ejemplo, el trabajador `CustomerDataHealer` está diseñado para identificar y corregir inconsistencias en los datos de clientes. Una vez más, utilizamos `Response Fencing` y `Ventriloquist` para obtener una salida estructurada. Es importante destacar que la directiva del sistema del trabajador incluye instrucciones para usar la función `flag_for_review` si se encuentran errores graves.

Cuando el trabajador procesa los datos del cliente, analiza la información e intenta corregir cualquier inconsistencia. Si el trabajador determina que los errores son graves y requieren intervención humana, puede utilizar la herramienta `flag_for_review` para marcar los datos y proporcionar una razón para el marcado.

El método `chat_completion` se llama con `json: true` para analizar los datos corregidos del cliente como JSON. No hay provisión para bucles después de una llamada a función, por lo que el resultado estará en blanco si se invocó `flag_for_review`. De lo contrario, el cliente se actualiza con los datos revisados y potencialmente corregidos.

Al incorporar la evaluación de la gravedad de los errores y la opción de marcar datos

para revisión humana, el trabajador de datos auto-reparables se vuelve más inteligente y adaptable. Puede manejar errores menores automáticamente mientras escala los errores graves a expertos humanos para intervención manual.

Los criterios específicos para determinar la gravedad del error pueden definirse en la directiva del trabajador basándose en el conocimiento del dominio y los requisitos del negocio. Factores como el impacto en la integridad de los datos, el potencial de pérdida o corrupción de datos, y las consecuencias de datos incorrectos pueden considerarse al evaluar la gravedad.

Al aprovechar la IA para evaluar la gravedad de los errores y proporcionar opciones para la intervención humana, las técnicas de datos auto-reparables pueden lograr un equilibrio entre la automatización y el mantenimiento de la precisión de los datos. Este enfoque asegura que los errores menores se corrijan eficientemente mientras que los errores graves reciban la atención y experiencia necesaria de los revisores humanos.

Complejidad del Dominio

Al considerar la aplicación de técnicas de datos auto-reparables, es importante evaluar la complejidad del dominio de datos y las reglas que gobiernan su estructura y relaciones. La complejidad del dominio puede impactar significativamente en la efectividad y viabilidad de los enfoques de corrección automatizada de datos.

Las técnicas de datos auto-reparables funcionan bien cuando los datos siguen patrones y restricciones bien definidos. En dominios donde la estructura de datos es relativamente simple y las relaciones entre elementos de datos son directas, las correcciones automatizadas pueden aplicarse con un alto grado de confianza. Por ejemplo, corregir problemas de formato o aplicar restricciones básicas de tipos de datos a menudo puede ser manejado efectivamente por trabajadores de datos auto-reparables.

Sin embargo, a medida que aumenta la complejidad del dominio de datos, también crecen los desafíos asociados con la corrección automatizada de datos. En dominios con

lógica de negocio intrincada, relaciones complejas entre entidades de datos, o reglas y excepciones específicas del dominio, las técnicas de datos auto-reparables pueden no siempre capturar los matices y pueden introducir consecuencias no deseadas.

Consideremos un ejemplo de un dominio complejo: un sistema de trading financiero. En este dominio, los datos involucran varios instrumentos financieros, datos de mercado, reglas de trading y requisitos regulatorios. Las relaciones entre diferentes elementos de datos pueden ser intrincadas, y las reglas que gobiernan la validez y consistencia de los datos pueden ser altamente específicas del dominio.

En un dominio tan complejo, un trabajador de datos auto-reparables encargado de corregir inconsistencias en datos de trading necesitaría tener un profundo entendimiento de las reglas y restricciones específicas del dominio. Necesitaría considerar factores como las regulaciones del mercado, límites de trading, cálculos de riesgo y procedimientos de liquidación. Las correcciones automatizadas en este contexto pueden no siempre capturar la complejidad total del dominio y pueden inadvertidamente introducir errores o violar reglas específicas del dominio.

Para abordar los desafíos de la complejidad del dominio, las técnicas de datos auto-reparables pueden mejorarse incorporando conocimiento y reglas específicas del dominio en los modelos y trabajadores de IA. Esto puede lograrse a través de técnicas como:

1. **Entrenamiento Específico del Dominio:** Los modelos de IA utilizados para datos auto-reparables pueden ser dirigidos o incluso afinados en conjuntos de datos específicos del dominio que capturan las complejidades y reglas del dominio particular. Al exponer los modelos a datos y escenarios representativos, pueden aprender los patrones, restricciones y excepciones específicas del dominio.
2. **Restricciones Basadas en Reglas:** Los trabajadores de datos auto-reparables pueden aumentarse con restricciones explícitas basadas en reglas que codifican el conocimiento específico del dominio. Estas reglas pueden ser definidas por expertos del dominio e integradas en el proceso de corrección de datos. Los

modelos de IA pueden entonces usar estas reglas para guiar sus decisiones y asegurar el cumplimiento de los requisitos específicos del dominio.

3. **Colaboración con Expertos del Dominio:** En dominios complejos, es crucial involucrar a expertos del dominio en el diseño y desarrollo de técnicas de datos auto-reparables. Los expertos del dominio pueden proporcionar valiosas perspectivas sobre las complejidades de los datos, las reglas de negocio y los posibles casos límite. Su conocimiento puede incorporarse en los modelos y trabajadores de IA para mejorar la precisión y fiabilidad de las correcciones automatizadas de datos utilizando patrones de [Human In The Loop](#).
4. **Enfoque Incremental e Iterativo:** Al tratar con dominios complejos, a menudo es beneficioso adoptar un enfoque incremental e iterativo para los datos auto-reparables. En lugar de intentar automatizar correcciones para todo el dominio de una vez, enfocarse en subdominios específicos o categorías de datos donde las reglas y restricciones son bien comprendidas. Gradualmente expandir el alcance de las técnicas auto-reparables a medida que crece el entendimiento del dominio y las técnicas demuestran ser efectivas.

Al considerar la complejidad del dominio de datos e incorporar el conocimiento específico del dominio en las técnicas de datos autorreparables, se puede lograr un equilibrio entre la automatización y la precisión. Es importante reconocer que los datos autorreparables no son una solución única para todos los casos y que el enfoque debe adaptarse a los requisitos y desafíos específicos de cada dominio.

En dominios complejos, un enfoque híbrido que combine técnicas de datos autorreparables con la experiencia y supervisión humana puede ser más efectivo. Las correcciones automatizadas pueden manejar casos rutinarios y bien definidos, mientras que los escenarios complejos o las excepciones pueden marcarse para revisión e intervención humana. Este enfoque colaborativo asegura que se obtengan los beneficios de la automatización mientras se mantiene el control y la precisión necesarios en dominios de datos complejos.

Explicabilidad y Transparencia

La explicabilidad se refiere a la capacidad de comprender e interpretar el razonamiento detrás de las decisiones tomadas por los modelos de IA, mientras que la transparencia implica proporcionar una visibilidad clara del proceso de corrección de datos.

En muchos contextos, las modificaciones de datos deben ser auditables y justificables. Las partes interesadas, incluyendo usuarios empresariales, auditores y organismos reguladores, pueden requerir explicaciones sobre por qué se realizaron ciertas correcciones de datos y cómo los modelos de IA llegaron a esas decisiones. Esto es especialmente crucial en dominios donde la precisión e integridad de los datos tienen implicaciones significativas, como las finanzas, la atención médica y los asuntos legales.

Para abordar la necesidad de explicabilidad y transparencia, las técnicas de datos autorreparables deben incorporar mecanismos que proporcionen información sobre el proceso de toma de decisiones de los modelos de IA. Esto se puede lograr a través de varios enfoques:

1. **Cadena de Pensamiento:** Pedir al modelo que explique su razonamiento “en voz alta” antes de aplicar cambios a los datos puede permitir una comprensión más fácil del proceso de toma de decisiones y puede generar explicaciones legibles por humanos para las correcciones realizadas. El compromiso es un poco más de complejidad en la separación de la explicación de la salida de datos estructurados, lo cual puede abordarse mediante...
2. **Generación de Explicaciones:** Los trabajadores de datos autorreparables pueden estar equipados con la capacidad de generar explicaciones legibles por humanos para las correcciones que realizan. Esto se puede lograr pidiendo al modelo que genere su proceso de toma de decisiones como explicaciones fácilmente comprensibles *integradas en los propios datos*. Por ejemplo, un trabajador de datos autorreparables podría generar un informe que destaque las inconsistencias específicas de datos que identificó, las correcciones que aplicó y el razonamiento

detrás de esas correcciones.

3. **Importancia de Características:** Los modelos de IA pueden ser instruidos con información sobre la importancia de diferentes características o atributos en el proceso de corrección de datos como parte de sus directivas. Estas directivas, a su vez, pueden ser expuestas a las partes interesadas humanas. Al identificar los factores clave que influyen en las decisiones del modelo, las partes interesadas pueden obtener información sobre el razonamiento detrás de las correcciones y evaluar su validez.
4. **Registro y Auditoría:** Implementar mecanismos completos de registro y auditoría es crucial para mantener la transparencia en el proceso de datos autorreparables. Cada corrección de datos realizada por los modelos de IA debe ser registrada, incluyendo los datos originales, los datos corregidos y las acciones específicas tomadas. Este registro de auditoría permite el análisis retrospectivo y proporciona un registro claro de las modificaciones realizadas a los datos.
5. **Enfoque con Humano en el Ciclo:** Incorporar un enfoque con humano en el ciclo puede mejorar la explicabilidad y transparencia de las técnicas de datos autorreparables. Al involucrar a expertos humanos en la revisión y validación de las correcciones generadas por IA, las organizaciones pueden asegurar que las correcciones se alineen con el conocimiento del dominio y los requisitos empresariales. La supervisión humana añade una capa adicional de responsabilidad y permite la identificación de posibles sesgos o errores en los modelos de IA.
6. **Monitoreo y Evaluación Continua:** El monitoreo y evaluación regulares del rendimiento de las técnicas de datos autorreparables es esencial para mantener la transparencia y la confianza. Al evaluar la precisión y efectividad de los modelos de IA a lo largo del tiempo, las organizaciones pueden identificar cualquier desviación o anomalía y tomar acciones correctivas. El monitoreo continuo ayuda a asegurar que el proceso de datos autorreparables permanezca confiable y alineado con los resultados deseados.

La explicabilidad y la transparencia son consideraciones críticas al implementar técnicas de datos autorreparables. Al proporcionar explicaciones claras para las correcciones de datos, mantener registros de auditoría completos e involucrar la supervisión humana, las organizaciones pueden construir confianza en el proceso de datos autorreparables y asegurar que las modificaciones realizadas a los datos sean justificables y estén alineadas con los objetivos empresariales.

Es importante encontrar un equilibrio entre los beneficios de la automatización y la necesidad de transparencia. Si bien las técnicas de datos autorreparables pueden mejorar significativamente la calidad y eficiencia de los datos, no deberían implicar la pérdida de visibilidad y control sobre el proceso de corrección de datos. Al diseñar trabajadores de datos autorreparables con la explicabilidad y transparencia en mente, las organizaciones pueden aprovechar el poder de la IA mientras mantienen el nivel necesario de responsabilidad y confianza en los datos.

Consecuencias No Intencionadas

Si bien las técnicas de datos autorreparables tienen como objetivo mejorar la calidad y consistencia de los datos, es crucial ser consciente del potencial de consecuencias no intencionadas. Las correcciones automatizadas, si no se diseñan y monitorizan cuidadosamente, pueden alterar inadvertidamente el significado o contexto de los datos, llevando a problemas derivados.

Uno de los principales riesgos de los datos autorreparables es la introducción de sesgos o errores en el proceso de corrección de datos. Los modelos de IA, como cualquier otro sistema de software, pueden estar sujetos a sesgos presentes en los datos de entrenamiento o introducidos a través del diseño de los algoritmos. Si estos sesgos no son identificados y mitigados, pueden propagarse a través del proceso de datos autorreparables y resultar en modificaciones de datos sesgadas o incorrectas.

Por ejemplo, consideremos un trabajador de datos autorreparables encargado de corregir inconsistencias en datos demográficos de clientes. Si el modelo de IA ha aprendido sesgos

de datos históricos, como asociar ciertas ocupaciones o niveles de ingresos con géneros o etnias específicas, podría hacer suposiciones incorrectas y modificar los datos de una manera que refuerce estos sesgos. Esto puede llevar a perfiles de clientes inexactos, decisiones comerciales equivocadas y resultados potencialmente discriminatorios.

Otra consecuencia no intencionada potencial es la pérdida de información valiosa o contexto durante el proceso de corrección de datos. Las técnicas de datos autorreparables a menudo se centran en estandarizar y normalizar datos para garantizar la consistencia. Sin embargo, en algunos casos, los datos originales pueden contener matices, excepciones o información contextual que es importante para comprender el panorama completo. Las correcciones automatizadas que aplican la estandarización de manera ciega pueden inadvertidamente eliminar u ocultar esta información valiosa.

Por ejemplo, imaginemos un trabajador de datos autorreparables responsable de corregir inconsistencias en registros médicos. Si el trabajador encuentra el historial médico de un paciente con una condición rara o un plan de tratamiento inusual, podría intentar normalizar los datos para ajustarlos a un patrón más común. Sin embargo, al hacerlo, podría perder los detalles específicos y el contexto que son cruciales para representar con precisión la situación única del paciente. Esta pérdida de información puede tener serias implicaciones para la atención del paciente y la toma de decisiones médicas.

Para mitigar los riesgos de consecuencias no intencionadas, es esencial adoptar un enfoque proactivo al diseñar e implementar técnicas de datos autorreparables:

1. **Pruebas y Validación Exhaustivas:** Antes de implementar trabajadores de datos autorreparables en producción, es crucial probar y validar exhaustivamente su comportamiento contra una amplia gama de escenarios. Esto incluye pruebas con conjuntos de datos representativos que cubren varios casos límite, excepciones y sesgos potenciales. Las pruebas rigurosas ayudan a identificar y abordar cualquier consecuencia no intencionada antes de que afecte a datos del mundo real.
2. **Monitoreo y Evaluación Continua:** Implementar mecanismos de monitoreo y evaluación continua es esencial para detectar y mitigar consecuencias no

intencionadas a lo largo del tiempo. Revisar regularmente los resultados de los procesos de datos autorreparables, analizar el impacto en los sistemas posteriores y la toma de decisiones, y recopilar retroalimentación de las partes interesadas puede ayudar a identificar cualquier efecto adverso y provocar acciones correctivas oportunas. Si su organización tiene paneles operativos, probablemente sea una buena idea agregar métricas claramente visibles relacionadas con cambios automatizados de datos. ¡Aregar alarmas conectadas a grandes desviaciones de la actividad normal de cambios de datos es probablemente una idea aún mejor!

3. **Supervisión e Intervención Humana:** Mantener la supervisión humana y la capacidad de intervenir en el proceso de datos autorreparables es crucial. Si bien la automatización puede mejorar enormemente la eficiencia, es importante que expertos humanos revisen y validen las correcciones realizadas por los modelos de IA, especialmente en dominios críticos o sensibles. El juicio humano y la experiencia en el dominio pueden ayudar a identificar y abordar cualquier consecuencia no intencionada que pueda surgir.
4. **IA Explicable (XAI) y Transparencia:** Como se discutió en la subsección anterior, incorporar técnicas de IA explicable y asegurar la transparencia en el proceso de datos autorreparables puede ayudar a mitigar las consecuencias no intencionadas. Al proporcionar explicaciones claras para las correcciones de datos y mantener pistas de auditoría completas, las organizaciones pueden comprender mejor y rastrear el razonamiento detrás de las modificaciones realizadas por los modelos de IA.
5. **Enfoque Incremental e Iterativo:** Adoptar un enfoque incremental e iterativo para los datos autorreparables puede ayudar a minimizar el riesgo de consecuencias no intencionadas. En lugar de aplicar correcciones automatizadas a todo el conjunto de datos de una vez, comience con un subconjunto de datos y gradualmente expanda el alcance a medida que las técnicas demuestren ser efectivas y confiables. Esto permite un monitoreo cuidadoso y ajustes en el camino, reduciendo el impacto de cualquier consecuencia no intencionada.

6. Colaboración y Retroalimentación: Involucrar a las partes interesadas de diferentes dominios y fomentar la colaboración y retroalimentación durante todo el proceso de datos autorreparables puede ayudar a identificar y abordar consecuencias no intencionadas. Buscar regularmente la opinión de expertos en el dominio, consumidores de datos y usuarios finales puede proporcionar información valiosa sobre el impacto en el mundo real de las correcciones de datos y destacar cualquier problema que pueda haberse pasado por alto.

Al abordar proactivamente el riesgo de consecuencias no intencionadas e implementar salvaguardas apropiadas, las organizaciones pueden aprovechar los beneficios de las técnicas de datos autorreparables mientras minimizan los efectos adversos potenciales. Es importante abordar los datos autorreparables como un proceso iterativo y colaborativo, monitoreando, evaluando y refinando continuamente las técnicas para asegurar que se alineen con los resultados deseados y mantengan la integridad y confiabilidad de los datos.

Al considerar el uso de patrones de datos autorreparables, es esencial evaluar cuidadosamente estos factores y sopesar los beneficios contra los riesgos y limitaciones potenciales. En algunos casos, un enfoque híbrido que combine correcciones automatizadas con supervisión e intervención humana puede ser la solución más apropiada.

También vale la pena señalar que las técnicas de datos autorreparables no deben verse como un reemplazo de los mecanismos robustos de validación de datos, sanitización de entrada y manejo de errores. Estas prácticas fundamentales siguen siendo críticas para garantizar la integridad y seguridad de los datos. Los datos autorreparables deben verse como un enfoque complementario que puede aumentar y mejorar estas medidas existentes.

En última instancia, la decisión de emplear patrones de datos autorreparables depende de los requisitos específicos, restricciones y prioridades de su aplicación. Al considerar cuidadosamente las consideraciones descritas anteriormente y alinearlas con los objetivos y la arquitectura de su aplicación, puede tomar decisiones informadas sobre cuándo y cómo aprovechar las técnicas de datos autorreparables de manera efectiva.

Generación de Contenido Contextual



Los patrones de Generación de Contenido Contextual aprovechan el poder de los modelos de lenguaje grande (LLMs) para generar contenido dinámico y específico al contexto dentro de las aplicaciones. Esta categoría de patrones reconoce la importancia de entregar contenido personalizado y relevante a los usuarios basado en sus necesidades específicas, preferencias e incluso interacciones previas y actuales con la aplicación.

En el contexto de este enfoque, “contenido” se refiere tanto al contenido primario (es decir, publicaciones de blog, artículos, etc.) como al meta-contenido, como las recomendaciones al contenido primario.

Los patrones de Generación de Contenido Contextual pueden jugar un papel crucial en mejorar tus niveles de participación del usuario, proporcionando experiencias personalizadas y automatizando tareas de creación de contenido tanto para ti como para tus usuarios. Al utilizar los patrones que describimos en este capítulo, puedes crear aplicaciones que generen contenido de manera dinámica, adaptándose al contexto y a las entradas en tiempo real.

Los patrones funcionan integrando LLMs en las salidas de la aplicación, desde la interfaz de usuario (a veces referida como “chrome”), hasta correos electrónicos y otras formas de notificaciones, así como cualquier pipeline de generación de contenido.

Cuando un usuario interactúa con la aplicación o activa una solicitud específica de contenido, la aplicación captura el contexto relevante, como las preferencias del usuario, interacciones previas o indicaciones específicas. Esta información contextual se introduce luego en el LLM, junto con cualquier plantilla o directriz necesaria y se utiliza para producir una salida textual que de otro modo tendría que ser codificada directamente, almacenada en una base de datos o generada algorítmicamente.

El contenido generado por LLM puede tomar varias formas, como recomendaciones personalizadas, descripciones dinámicas de productos, respuestas personalizadas por correo electrónico o incluso artículos o publicaciones de blog completos. Uno de los usos más radicales de este contenido que desarrollé hace más de un año es la generación dinámica de elementos de UI como etiquetas de formularios, tooltips y otros tipos de texto explicativo.

Personalización

Uno de los beneficios clave de los patrones de Generación de Contenido Contextual es la capacidad de ofrecer experiencias altamente personalizadas a los usuarios. Al generar contenido basado en el contexto específico del usuario, estos patrones permiten que las aplicaciones adapten el contenido a los intereses, preferencias e interacciones individuales de los usuarios.

La personalización va más allá de simplemente insertar el nombre de un usuario en contenido genérico. Implica aprovechar el rico contexto disponible sobre cada usuario para generar contenido que resuene con sus necesidades y deseos específicos. Este contexto puede incluir una amplia gama de factores, como:

1. **Información del Perfil de Usuario:** En el nivel más general de aplicación de esta técnica, los datos demográficos, intereses, preferencias y otros atributos del perfil pueden utilizarse para generar contenido que se alinee con los antecedentes y características del usuario.
2. **Datos de Comportamiento:** Las interacciones pasadas del usuario con la aplicación, como páginas vistas, enlaces clicados o productos comprados, pueden proporcionar información valiosa sobre su comportamiento e intereses. Estos datos pueden utilizarse para generar sugerencias de contenido que reflejen sus patrones de participación y predigan sus necesidades futuras.
3. **Factores Contextuales:** El contexto actual del usuario, como su ubicación, dispositivo, hora del día o incluso el clima, puede influir en el proceso de generación de contenido. Por ejemplo, una aplicación de viajes podría tener un trabajador de IA capaz de generar recomendaciones personalizadas basadas en la ubicación actual del usuario y las condiciones climáticas predominantes.

Al aprovechar estos factores contextuales, los patrones de Generación de Contenido Contextual permiten que las aplicaciones entreguen contenido que se siente hecho a medida para cada usuario individual. Este nivel de personalización tiene varios beneficios significativos:

1. **Mayor Participación:** El contenido personalizado captura la atención de los usuarios y los mantiene comprometidos con la aplicación. Cuando los usuarios sienten que el contenido es relevante y habla directamente a sus necesidades, es más probable que pasen más tiempo interactuando con la aplicación y explorando sus características.

2. **Mejor Satisfacción del Usuario:** El contenido personalizado demuestra que la aplicación entiende y se preocupa por los requisitos únicos del usuario. Al proporcionar contenido que es útil, informativo y alineado con sus intereses, la aplicación puede mejorar la satisfacción del usuario y construir una conexión más fuerte con sus usuarios.
3. **Mayores Tasas de Conversión:** En el contexto de aplicaciones de comercio electrónico o marketing, el contenido personalizado puede impactar significativamente en las tasas de conversión. Al presentar a los usuarios productos, ofertas o recomendaciones que están adaptadas a sus preferencias y comportamiento, la aplicación puede aumentar la probabilidad de que los usuarios realicen las acciones deseadas, como realizar una compra o registrarse en un servicio.

Productividad

Los patrones de Generación de Contenido Contextual pueden aumentar significativamente ciertos tipos de productividad al reducir la necesidad de generación y edición manual de contenido en procesos creativos. Al aprovechar el poder de los LLMs, puedes generar contenido de alta calidad a escala, ahorrando tiempo y esfuerzo que tus creadores de contenido y desarrolladores tendrían que gastar de otro modo haciendo trabajo manual tedioso.

Tradicionalmente, los creadores de contenido necesitan investigar, escribir, editar y formatear contenido para asegurar que cumpla con los requisitos de la aplicación y las expectativas de los usuarios. Este proceso puede consumir mucho tiempo y recursos, especialmente a medida que el volumen de contenido crece.

Sin embargo, con los patrones de Generación de Contenido Contextual, el proceso de creación de contenido puede automatizarse en gran medida. Los LLMs pueden generar contenido coherente, gramaticalmente correcto y contextualmente relevante basado en

las indicaciones y pautas proporcionadas. Esta automatización ofrece varios beneficios de productividad:

1. **Reducción del Esfuerzo Manual:** Al delegar las tareas de generación de contenido a los LLMs, los creadores de contenido pueden enfocarse en tareas de nivel superior como la estrategia de contenido, la ideación y el aseguramiento de la calidad. Pueden proporcionar el contexto necesario, las plantillas y las pautas al LLM y dejar que este se encargue de la generación real del contenido. Esto reduce el esfuerzo manual requerido para escribir y editar, permitiendo que los creadores de contenido sean más productivos y eficientes.
2. **Creación de Contenido Más Rápida:** Los LLMs pueden generar contenido mucho más rápido que los escritores humanos. Con las indicaciones y pautas adecuadas, un LLM puede producir múltiples piezas de contenido en cuestión de segundos o minutos. Esta velocidad permite que las aplicaciones generen contenido a un ritmo mucho más rápido, manteniéndose al día con las demandas de los usuarios y el panorama digital en constante cambio.

¿Está la creación más rápida de contenido llevando a una situación de “tragedia de los comunes” donde internet se está ahogando en contenido que nadie lee? Lamentablemente, sospecho que la respuesta es sí.

3. **Consistencia y Calidad:** Los LLMs pueden revisar trivialmente el contenido para que sea consistente en estilo, tono y calidad. Proporcionando pautas claras y ejemplos, ciertos tipos de aplicaciones (es decir, redacciones periodísticas, relaciones públicas, etc.) pueden asegurar que su contenido generado por humanos se alinee con la voz de su marca y cumpla con los estándares de calidad deseados. Esta consistencia reduce la necesidad de ediciones y revisiones extensas, ahorrando tiempo y esfuerzo en el proceso de creación de contenido.

4. Iteración y Optimización: Los patrones de Generación de Contenido Contextual permiten una rápida iteración y optimización del contenido. Al ajustar las indicaciones, plantillas o pautas proporcionadas al LLM, sus aplicaciones pueden generar rápidamente variaciones de contenido y probar diferentes enfoques de una manera automatizada que nunca fue posible en el pasado. Este proceso iterativo permite una experimentación y refinamiento más rápidos de las estrategias de contenido, llevando a un contenido más efectivo y atractivo con el tiempo. Esta técnica en particular puede ser un cambio revolucionario para aplicaciones como el comercio electrónico que viven y mueren según las tasas de rebote y el engagement.



Es importante señalar que si bien los patrones de Generación de Contenido Contextual pueden mejorar enormemente la productividad, no eliminan por completo la necesidad de intervención humana. Los creadores y editores de contenido siguen desempeñando un papel crucial en la definición de la estrategia general de contenido, proporcionando orientación al LLM y asegurando la calidad y adecuación del contenido generado.

Al automatizar los aspectos más repetitivos y que consumen más tiempo de la creación de contenido, los patrones de Generación de Contenido Contextual liberan valioso tiempo y recursos humanos que pueden redirigirse hacia tareas de mayor valor. Este aumento en la productividad le permite entregar contenido más personalizado y atractivo a los usuarios mientras optimiza los flujos de trabajo de creación de contenido.

Iteración Rápida y Experimentación

Los patrones de Generación de Contenido Contextual le permiten iterar y experimentar rápidamente con diferentes variaciones de contenido, permitiendo una optimización y refinamiento más rápido de su estrategia de contenido. Puede generar múltiples

versiones de contenido en cuestión de segundos, simplemente ajustando el contexto, las plantillas o las pautas proporcionadas al modelo.

Esta capacidad de iteración rápida ofrece varios beneficios clave:

1. **Pruebas y Optimización:** Con la capacidad de generar variaciones de contenido rápidamente, puede probar fácilmente diferentes enfoques y medir su efectividad. Por ejemplo, puede generar múltiples versiones de una descripción de producto o un mensaje de marketing, cada uno adaptado a un segmento de usuario específico o contexto. Al analizar las métricas de participación del usuario, como las tasas de clics o las tasas de conversión, puede identificar las variaciones de contenido más efectivas y optimizar su estrategia de contenido en consecuencia.
2. **Pruebas A/B:** Los patrones de Generación de Contenido Contextual permiten realizar pruebas A/B de contenido sin problemas. Puede generar dos o más variaciones de contenido y servirlas aleatoriamente a diferentes grupos de usuarios. Al comparar el rendimiento de cada variación, puede determinar qué contenido resuena mejor con su audiencia objetivo. Este enfoque basado en datos le permite tomar decisiones informadas y refinar continuamente su contenido para maximizar la participación del usuario y lograr los resultados deseados.
3. **Experimentos de Personalización:** La iteración rápida y la experimentación son particularmente valiosas cuando se trata de personalización. Con los patrones de Generación de Contenido Contextual, puede generar rápidamente variaciones de contenido personalizado basadas en diferentes segmentos de usuarios, preferencias o comportamientos. Al experimentar con diferentes estrategias de personalización, puede identificar los enfoques más efectivos para involucrar a usuarios individuales y entregar experiencias personalizadas.
4. **Adaptación a las Tendencias Cambiantes:** La capacidad de iterar y experimentar rápidamente te permite mantener la agilidad y adaptarte a las tendencias cambiantes y las preferencias de los usuarios. A medida que surgen nuevos temas, palabras clave o comportamientos de usuario, puedes generar rápidamente

contenido que se alinee con estas tendencias. Al experimentar y refinar continuamente tu contenido, puedes mantener la relevancia y conservar una ventaja competitiva en el panorama digital en constante evolución.

5. **Experimentación Rentable:** La experimentación tradicional de contenido a menudo implica tiempo y recursos significativos, ya que los creadores de contenido necesitan desarrollar y probar manualmente diferentes variaciones. Sin embargo, con los patrones de Generación Contextual de Contenido, el costo de la experimentación se reduce enormemente. Los MLGs pueden generar variaciones de contenido rápidamente y a escala, permitiéndote explorar una amplia gama de ideas y enfoques sin incurrir en costos sustanciales.

Para aprovechar al máximo la iteración rápida y la experimentación, es importante tener un marco de experimentación bien definido. Este marco debe incluir:

- Objetivos e hipótesis claros para cada experimento
- Métricas y mecanismos de seguimiento apropiados para medir el rendimiento del contenido
- Estrategias de segmentación y orientación para asegurar que las variaciones de contenido relevantes lleguen a los usuarios correctos
- Herramientas de análisis e informes para obtener información de los datos experimentales
- Un proceso para incorporar aprendizajes y optimizaciones en tu estrategia de contenido

Al adoptar la iteración y experimentación rápida, puedes refinar y optimizar continuamente tu contenido, asegurando que permanezca atractivo, relevante y efectivo en el logro de los objetivos de tu aplicación. Este enfoque ágil para la creación de contenido te permite mantenerte a la vanguardia y ofrecer experiencias de usuario excepcionales.

Escalabilidad y Eficiencia

A medida que las aplicaciones crecen y aumenta la demanda de contenido personalizado, los patrones de generación contextual de contenido permiten escalar eficientemente la creación de contenido. Los MLGs pueden generar contenido para un gran número de usuarios y contextos simultáneamente, sin necesidad de un aumento proporcional en recursos humanos. Esta escalabilidad permite que las aplicaciones ofrezcan experiencias personalizadas a una base de usuarios en crecimiento sin sobrecargar sus capacidades de creación de contenido.



Ten en cuenta que la generación contextual de contenido se puede utilizar efectivamente para internacionalizar tu aplicación “sobre la marcha”. De hecho, eso es exactamente lo que hice usando mi Gema Instant18n para ofrecer Olympia en más de media docena de idiomas, a pesar de que tenemos menos de un año.

Localización Impulsada por IA

Si me permites presumir por un momento, creo que mi biblioteca Instant18n para aplicaciones Rails es un ejemplo revolucionario del patrón de “Generación Contextual de Contenido” en acción, mostrando el potencial transformador de la IA en el desarrollo de aplicaciones. Esta gema aprovecha el poder del modelo de lenguaje grande GPT de OpenAI para revolucionar la forma en que se manejan la internacionalización y la localización en las aplicaciones Rails.

Tradicionalmente, internacionalizar una aplicación Rails implica definir manualmente claves de traducción y proporcionar las traducciones correspondientes para cada idioma soportado. Este proceso puede consumir mucho tiempo, recursos y ser propenso a inconsistencias. Sin embargo, con la gema Instant18n, el paradigma de localización se redefine completamente.

Al integrar un modelo de lenguaje grande, la gema Instant18n te permite generar traducciones sobre la marcha, basándose en el contexto y significado del texto. En lugar de depender de claves de traducción predefinidas y traducciones estáticas, la gema traduce dinámicamente el texto usando el poder de la IA. Este enfoque ofrece varios beneficios clave:

1. **Localización Sin Problemas:** Con la gema Instant18n, los desarrolladores ya no necesitan definir y mantener manualmente archivos de traducción para cada idioma soportado. La gema genera automáticamente traducciones basadas en el texto proporcionado y el idioma objetivo deseado, haciendo que el proceso de localización sea sin esfuerzo y fluido.
2. **Precisión Contextual:** La IA puede recibir suficiente contexto para comprender los matices del texto que se está traduciendo. Puede tener en cuenta el contexto circundante, modismos y referencias culturales para generar traducciones que sean precisas, naturales y contextualmente apropiadas.
3. **Soporte Extenso de Idiomas:** La gema Instant18n aprovecha el vasto conocimiento y las capacidades lingüísticas de GPT, permitiendo traducciones a una extensa gama de idiomas. Desde idiomas comunes como el español y el francés hasta idiomas más oscuros o ficticios como el klingon y el élfico, la gema puede manejar una amplia variedad de requisitos de traducción.
4. **Flexibilidad y Creatividad:** La gema va más allá de las traducciones tradicionales de idiomas y permite opciones de localización creativas y no convencionales. Los desarrolladores pueden traducir texto a varios estilos, dialectos o incluso idiomas ficticios, abriendo nuevas posibilidades para experiencias de usuario únicas y contenido atractivo.
5. **Optimización del Rendimiento:** La gema Instant18n incorpora mecanismos de caché para mejorar el rendimiento y reducir la sobrecarga de traducciones repetidas. El texto traducido se almacena en caché, permitiendo que las solicitudes posteriores de la misma traducción se sirvan rápidamente sin necesidad de llamadas API redundantes.

La gema Instant18n ejemplifica el poder del patrón de “Generación Contextual de Contenido” al aprovechar la IA para generar contenido localizado dinámicamente. Demuestra cómo la IA puede integrarse en la funcionalidad central de una aplicación Rails, transformando la forma en que los desarrolladores abordan la internacionalización y la localización.

Al eliminar la necesidad de gestionar las traducciones manualmente y permitir traducciones en tiempo real basadas en el contexto, la gema Instant18n ahorra a los desarrolladores tiempo y esfuerzo significativos. Les permite concentrarse en construir las características principales de su aplicación mientras se aseguran de que el aspecto de localización se maneje de manera fluida y precisa.

La Importancia de las Pruebas de Usuario y la Retroalimentación

Finalmente, ten siempre presente la importancia de las pruebas de usuario y la retroalimentación. Es crucial validar que la generación de contenido contextual cumpla con las expectativas del usuario y se alinee con los objetivos de la aplicación. Itera y refina continuamente el contenido generado basándote en las perspectivas de los usuarios y los análisis. Si estás generando contenido dinámico a gran escala que sería imposible de validar manualmente por ti y tu equipo, considera agregar mecanismos de retroalimentación que permitan a los usuarios reportar contenido que sea extraño o incorrecto, junto con una explicación del por qué. ¡Esa valiosa retroalimentación incluso puede ser alimentada a un trabajador de IA encargado de realizar ajustes al componente que generó el contenido!

UI Generativa



La atención es tan valiosa en estos días que para lograr un compromiso efectivo del usuario ahora se requieren experiencias de software que no solo sean fluidas e intuitivas, sino también altamente personalizadas según las necesidades, preferencias y contextos individuales. Como resultado, los diseñadores y desarrolladores se enfrentan cada vez más al desafío de crear interfaces de usuario que puedan adaptarse y satisfacer los requisitos únicos de cada usuario *a escala*.

La UI Generativa (GenUI) es un enfoque verdaderamente revolucionario para el diseño de interfaces de usuario que aprovecha el poder de los modelos de lenguaje grande (LLMs) para crear experiencias de usuario altamente personalizadas y dinámicas en tiempo real. Quería asegurarme de darte al menos una introducción sobre GenUI en este libro, porque creo que es una de las oportunidades más prometedoras que existe actualmente en el ámbito del diseño y marcos de aplicaciones. Estoy convencido de que docenas o más proyectos comerciales y de código abierto exitosos surgirán en este nicho

particular.

En su esencia, GenUI combina los principios de [Generación de Contenido Contextual](#) con técnicas avanzadas de IA para generar elementos de interfaz de usuario, como texto, imágenes y diseños, de manera dinámica basándose en una comprensión profunda del contexto, preferencias y objetivos del usuario. GenUI permite a los diseñadores y desarrolladores crear interfaces que se adaptan y evolucionan en respuesta a las interacciones del usuario, proporcionando un nivel de personalización que antes era inalcanzable.

GenUI representa un cambio fundamental en la forma en que abordamos el diseño de interfaces de usuario. En lugar de diseñar para las masas, GenUI nos permite diseñar para el individuo. El contenido y las interfaces personalizadas tienen el potencial de crear experiencias de usuario que resuenan con cada usuario a un nivel más profundo, aumentando el compromiso, la satisfacción y la lealtad.

Como técnica de vanguardia, la transición a GenUI está llena de desafíos conceptuales y prácticos. Integrar la IA en el proceso de diseño, asegurar que las interfaces generadas no solo sean personalizadas sino también utilizables, accesibles y alineadas con la marca general y la experiencia del usuario, todos estos son desafíos que hacen de GenUI una búsqueda para pocos, no para muchos. Además, la participación de la IA plantea cuestiones sobre la privacidad de datos, la transparencia e incluso implicaciones éticas.

A pesar de los desafíos, las experiencias personalizadas a escala tienen el poder de transformar completamente la manera en que interactuamos con productos y servicios digitales. Abre posibilidades para crear interfaces inclusivas y accesibles que satisfagan las diversas necesidades de los usuarios, independientemente de sus capacidades, antecedentes o preferencias.

En este capítulo, exploraremos el concepto de GenUI, examinando algunas características definitorias, beneficios clave y desafíos potenciales. Comenzamos considerando la forma más básica y accesible de GenUI: la generación de texto para interfaces de usuario diseñadas e implementadas de manera tradicional.

Generando Texto para Interfaces de Usuario

Los elementos de texto que existen en la interfaz de tu aplicación, como etiquetas de formularios, tooltips y texto explicativo, típicamente están codificados en las plantillas o componentes de UI, proporcionando una experiencia consistente pero genérica para todos los usuarios. Utilizando patrones de generación de contenido contextual, puedes transformar estos elementos estáticos en componentes dinámicos, conscientes del contexto y personalizados.

Formularios Personalizados

Los formularios son una parte ubicua de las aplicaciones web y móviles, sirviendo como el medio principal para recopilar información del usuario. Sin embargo, los formularios tradicionales a menudo presentan una experiencia genérica e impersonal, con etiquetas y campos estándar que no siempre se alinean con el contexto o las necesidades específicas del usuario. Los usuarios son más propensos a completar formularios que se sienten adaptados a sus necesidades y preferencias, lo que lleva a mayores tasas de conversión y satisfacción del usuario.

Sin embargo, es importante encontrar un equilibrio entre la personalización y la consistencia. Si bien adaptar los formularios a usuarios individuales puede ser beneficioso, es crucial mantener un nivel de familiaridad y previsibilidad. Los usuarios aún deberían poder reconocer y navegar por los formularios fácilmente, incluso con elementos personalizados.

Aquí hay algunas ideas de formularios personalizados para inspirarse:

Sugerencias Contextuales de Campo

GenUI puede analizar las interacciones previas del usuario, preferencias y datos para proporcionar sugerencias inteligentes de campo como predicciones. Por ejemplo,

si el usuario ha ingresado previamente su dirección de envío, el formulario puede autocompletar automáticamente los campos relevantes con su información guardada. Esto no solo ahorra tiempo, sino que también demuestra que la aplicación entiende y recuerda las preferencias del usuario.

Un momento, ¿no es esta técnica algo que podría hacerse sin involucrar la IA? Por supuesto, pero la belleza de impulsar este tipo de funcionalidad con IA es doble: 1) lo fácil que puede ser de implementar y 2) lo resistente que puede ser a medida que tu IU cambia y evoluciona con el tiempo.

Vamos a crear rápidamente un servicio para nuestro sistema teórico de gestión de pedidos, que intente completar proactivamente la dirección de envío correcta para el usuario.

```
1 class OrderShippingAddressSubscriber
2   include Raix::ChatCompletion
3
4   attr_accessor :order
5
6   delegate :customer, to: :order
7
8   DIRECTIVE = "You are a smart order processing assistant. Given the
9   customer's order history, guess the most likely shipping address
10  for the current order."
11
12  def order_created(order)
13    return unless order.pending? && order.shipping_address.blank?
14
15    self.order = order
16
17    transcript.clear
18    transcript << { system: DIRECTIVE }
19    transcript << { user: "Order History: #{order_history.to_json}" }
20    transcript << { user: "Current Order: #{order.to_json}" }
21
22    response = chat_completion
23    apply_predicted_shipping_address(order, response)
24  end
25
```

```
26     private
27
28     def apply_predicted_shipping_address(order, response)
29         # extract the shipping address from the response...
30         # ...and assume there's some sort of live update of the address fields
31         order.update(shipping_address:)
32     end
33
34     def order_history
35         customer.orders.successful.limit(100).map do |order|
36             {
37                 date: order.date,
38                 description: order.description,
39                 shipping_address: order.shipping_address
40             }
41     end
42   end
43 end
```

Este ejemplo está muy simplificado, pero debería funcionar para la mayoría de los casos. La idea es permitir que la IA haga una suposición de la misma manera que lo haría un humano. Para dejar claro de qué estoy hablando, consideremos algunos datos de muestra:

```
1 Order History:
2 [
3     {"date": "2024-01-03", "description": "garden soil mix",
4      "shipping_address": "123 Country Lane, Rural Town"},  

5     {"date": "2024-01-15", "description": "hardcover fiction novels",
6      "shipping_address": "456 City Apt, Metroville"},  

7     {"date": "2024-01-22", "description": "baby diapers", "shipping_address":  

8      "789 Suburb St, Quietville"},  

9     {"date": "2024-02-01", "description": "organic vegetables",
10      "shipping_address": "123 Country Lane, Rural Town"},  

11     {"date": "2024-02-17", "description": "mystery thriller book set",
12      "shipping_address": "456 City Apt, Metroville"},  

13     {"date": "2024-02-25", "description": "baby wipes",
14      "shipping_address": "789 Suburb St, Quietville"},  

15     {"date": "2024-03-05", "description": "flower seeds",
16      "shipping_address": "123 Country Lane, Rural Town"},
```

```
17 {"date": "2024-03-20", "description": "biographies",  
18   "shipping_address": "456 City Apt, Metroville"},  
19 {"date": "2024-03-30", "description": "baby formula",  
20   "shipping_address": "789 Suburb St, Quietville"},  
21 {"date": "2024-04-12", "description": "lawn fertilizer",  
22   "shipping_address": "123 Country Lane, Rural Town"},  
23 {"date": "2024-04-22", "description": "science fiction novels",  
24   "shipping_address": "456 City Apt, Metroville"},  
25 {"date": "2024-05-02", "description": "infant toys",  
26   "shipping_address": "789 Suburb St, Quietville"},  
27 {"date": "2024-05-14", "description": "outdoor grill",  
28   "shipping_address": "123 Country Lane, Rural Town"},  
29 {"date": "2024-05-29", "description": "literary classics",  
30   "shipping_address": "456 City Apt, Metroville"},  
31 {"date": "2024-06-11", "description": "baby clothes",  
32   "shipping_address": "789 Suburb St, Quietville"},  
33 {"date": "2024-07-01", "description": "watering can",  
34   "shipping_address": "123 Country Lane, Rural Town"},  
35 {"date": "2024-07-18", "description": "non-fiction essays",  
36   "shipping_address": "456 City Apt, Metroville"},  
37 {"date": "2024-07-28", "description": "baby bath items",  
38   "shipping_address": "789 Suburb St, Quietville"},  
39 {"date": "2024-08-09", "description": "herb garden kit",  
40   "shipping_address": "123 Country Lane, Rural Town"},  
41 {"date": "2024-08-24", "description": "children's books",  
42   "shipping_address": "456 City Apt, Metroville"}  
43 ]
```

¿Notaste el patrón en los datos? Te garantizo que esto es algo sencillo para un LLM. Para demostrarlo, preguntémosle a GPT-4 cuál es la dirección de envío más probable para un “termómetro”.

1 From the order history you've provided, it looks like the purchases are
2 generally clustered into three main types based on the shipping addresses:
3
4 123 Country Lane, Rural Town - This address often orders garden and
5 outdoor-related items like soil mix, vegetables, flower seeds, lawn
6 fertilizer, an outdoor grill, a watering can, and an herb garden kit.
7
8 456 City Apt, Metroville - This address tends to order various types of
9 books, indicating an interest in reading across genres like fiction,
10 thrillers, biographies, science fiction, literary classics, non-fiction
11 essays, and children's books.
12
13 789 Suburb St, Quietville - The purchases for this address are consistently
14 baby-related products, such as diapers, wipes, baby formula, infant toys,
15 baby clothes, and baby bath items.
16
17 Given these patterns, a digital thermometer could potentially be a fit for
18 any of the addresses based on different needs: outdoor temperature tracking,
19 a tool for a passionate reader interested in science gadgets, or a health
20 tool for a baby. However, considering its common use in monitoring health,
21 especially for babies, it's most likely to be purchased by someone who's
22 already buying baby care products.
23
24 Therefore, the most likely shipping address for a digital thermometer, based
25 on the purchasing patterns, would be 789 Suburb St, Quietville.

Si estás pensando que es exagerado usar un modelo costoso como GPT-4 para esta tarea, ¡tienes razón! Probé el mismo prompt en Mistral 7B Instruct y produjo la siguiente respuesta a 75 tokens por segundo, y un costo ínfimo de \$0.000218 USD.

1 Based on the given order history, it appears that the customer has received
2 orders for baby-related items (diapers, wipes, formula, clothes, bath items)
3 at the address "789 Suburb St, Quietville" and garden-related items (soil,
4 seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
5 Town". The customer has also received books (fiction, non-fiction, literary
6 classics, mystery-thriller, science fiction, biographies, children's books)
7 at the address "456 City Apt, Metroville".
8
9 Given that the new order is for a "thermometer", which doesn't seem to fit
10 into any of the categories that the customer has previously ordered, it's
11 hard to accurately predict the shipping address based solely on the order
12 history. However, given the 50%-50% split between baby-related and
13 garden-related items, it could somewhat lean towards the Baby-related items
14 address ("789 Suburb St, Quietville"). But remember, this is an assumption
15 and cannot be definitively confirmed without more context or information.

¿Vale la pena el costo y la sobrecarga de esta técnica para hacer que la experiencia de pago sea más mágica? Para muchos minoristas en línea, definitivamente sí. Y por lo que parece, el costo del cómputo de IA solo va a disminuir, especialmente para los proveedores de alojamiento de modelos de código abierto en una carrera hacia el precio más bajo.



Utiliza una [Prompt Template](#) y [StructuredIO](#) junto con [Response Fencing](#) para optimizar este tipo de completado de chat.

Ordenamiento Adaptativo de Campos

El orden en que se presentan los campos del formulario puede impactar significativamente la experiencia del usuario y las tasas de finalización. Con GenUI, puedes ajustar dinámicamente el orden de los campos según el contexto del usuario y la importancia de cada campo. Por ejemplo, si el usuario está completando un formulario de registro para una aplicación de fitness, el formulario podría priorizar los campos relacionados con sus objetivos y preferencias de acondicionamiento físico, haciendo el proceso más relevante y atractivo.

Microtextos Personalizados

El texto instructivo, mensajes de error y otros microtextos asociados con los formularios también pueden personalizarse usando GenUI. En lugar de mostrar mensajes de error genéricos como “Dirección de correo electrónico inválida”, puedes generar mensajes más útiles y contextuales como “Por favor, ingresa una dirección de correo electrónico válida para recibir tu confirmación de pedido”. Estos toques personalizados pueden hacer que la experiencia del formulario sea más amigable y menos frustrante.

Validación Personalizada

En la misma línea que los Microtextos Personalizados, podrías usar la IA para validar el formulario de maneras que parezcan mágicas. Imagina permitir que una IA valide un formulario de perfil de usuario, buscando posibles errores a nivel *semántico*.

Create your account

Full name

Obie Fernandez

Email

obiefenandez@gmail.com



Did you mean obiefernandez@gmail.com? [Yes, update.](#)

Country

United States



Password

.....



Nice work. This is an excellent password.

Figura 9. ¿Puedes detectar la validación semántica en acción?

Revelación Progresiva

GenUI puede determinar de manera inteligente qué campos del formulario son esenciales basándose en el contexto del usuario y revelar gradualmente campos adicionales según sea necesario. Esta técnica de revelación progresiva ayuda a reducir la carga cognitiva y hace que el proceso de completar el formulario sea más manejable.

Por ejemplo, si un usuario se está registrando para una suscripción básica, el formulario puede presentar inicialmente solo los campos esenciales, y a medida que el usuario avanza o selecciona opciones específicas, se pueden introducir dinámicamente campos adicionales relevantes.

Texto Explicativo Sensible al Contexto

Los tooltips se utilizan frecuentemente para proporcionar información adicional u orientación a los usuarios cuando pasan el cursor o interactúan con elementos específicos. Con un enfoque de “Generación de Contenido Contextual”, puedes generar tooltips que se adapten al contexto del usuario y proporcionen información relevante. Por ejemplo, si un usuario está explorando una función compleja, el tooltip puede ofrecer consejos personalizados o ejemplos basados en sus interacciones previas o nivel de habilidad.

El texto explicativo, como instrucciones, descripciones o mensajes de ayuda, puede generarse dinámicamente según el contexto del usuario. En lugar de presentar explicaciones genéricas, puedes usar LLMs para generar texto que se adapte a las necesidades o preguntas específicas del usuario. Por ejemplo, si un usuario está teniendo dificultades con un paso particular en un proceso, el texto explicativo puede proporcionar orientación personalizada o consejos para solucionar problemas.

Los microtextos se refieren a los pequeños fragmentos de texto que guían a los usuarios a través de tu aplicación, como etiquetas de botones, mensajes de error o avisos de confirmación. Al aplicar el enfoque de [Generación de Contenido Contextual](#) a los microtextos, puedes crear una interfaz adaptativa que responda a las acciones del usuario y proporcione texto relevante y útil. Por ejemplo, si un usuario está a punto de realizar una acción crítica, el aviso de confirmación puede generarse dinámicamente para proporcionar un mensaje claro y personalizado.

El texto explicativo y los tooltips personalizados pueden mejorar enormemente el proceso de incorporación de nuevos usuarios. Al proporcionar orientación y ejemplos

específicos al contexto, puedes ayudar a los usuarios a comprender y navegar rápidamente por la aplicación, reduciendo la curva de aprendizaje y aumentando la adopción.

Los elementos de interfaz dinámicos y sensibles al contexto también pueden hacer que la aplicación se sienta más intuitiva y atractiva. Es más probable que los usuarios interactúen y exploren las funciones cuando el texto que las acompaña está adaptado a sus necesidades e intereses específicos.

Hasta ahora hemos cubierto ideas para mejorar los paradigmas existentes de UI con IA, pero ¿qué hay sobre repensar cómo se diseñan e implementan las interfaces de usuario de una manera más radical?

Definiendo la UI Generativa

A diferencia del diseño tradicional de UI, donde los diseñadores crean interfaces fijas y estáticas, la GenUI sugiere un futuro en el que nuestro software presenta experiencias flexibles y personalizadas que pueden evolucionar y adaptarse en tiempo real. Cada vez que usamos una interfaz conversacional impulsada por IA, estamos permitiendo que la IA se adapte a las necesidades particulares del usuario. La GenUI va un paso más allá al aplicar ese nivel de adaptabilidad a la interfaz *visual* del software.

La razón por la que es posible experimentar con ideas de GenUI hoy en día es que los modelos de lenguaje grandes ya comprenden la programación y su conocimiento base incluye tecnologías y marcos de UI. La pregunta es, por tanto, si los modelos de lenguaje grandes pueden utilizarse para generar elementos de UI, como texto, imágenes, diseños e incluso interfaces completas, que estén adaptados a cada usuario individual. Se podría instruir al modelo para que tenga en cuenta varios factores, como las interacciones pasadas del usuario, las preferencias declaradas, la información

demográfica y el contexto actual de uso, para crear interfaces altamente personalizadas y relevantes.

La GenUI se diferencia del diseño tradicional de interfaces de usuario en varios aspectos clave:

1. **Dinámica y Adaptativa:** El diseño tradicional de UI implica crear interfaces fijas y estáticas que permanecen iguales para todos los usuarios. En contraste, la GenUI permite interfaces que pueden adaptarse y cambiar dinámicamente según las necesidades y el contexto del usuario. Esto significa que la misma aplicación puede presentar diferentes interfaces a diferentes usuarios o incluso al mismo usuario en diferentes situaciones.
2. **Personalización a Escala:** Con el diseño tradicional, crear experiencias personalizadas para cada usuario suele ser poco práctico debido al tiempo y recursos requeridos. La GenUI, por otro lado, permite la personalización a escala. Al aprovechar la IA, los diseñadores pueden crear interfaces que se adaptan automáticamente a las necesidades y preferencias únicas de cada usuario, sin tener que diseñar y desarrollar manualmente interfaces separadas para cada segmento de usuarios.
3. **Enfoque en Resultados:** El diseño tradicional de UI a menudo se centra en crear interfaces visualmente atractivas y funcionales. Si bien estos aspectos siguen siendo importantes en la GenUI, el enfoque principal cambia hacia la consecución de los resultados deseados por el usuario. La GenUI busca crear interfaces que estén optimizadas para los objetivos y tareas específicas de cada usuario, priorizando la usabilidad y efectividad sobre consideraciones puramente estéticas.
4. **Aprendizaje y Mejora Continua:** Los sistemas GenUI pueden aprender y mejorar continuamente con el tiempo basándose en las interacciones y retroalimentación de los usuarios. A medida que los usuarios interactúan con las interfaces generadas, los modelos de IA pueden recopilar datos sobre el comportamiento, preferencias y resultados del usuario, utilizando esta información para refinar

y optimizar futuras generaciones de interfaces. Este proceso de aprendizaje iterativo permite que los sistemas GenUI se vuelvan cada vez más efectivos para satisfacer las necesidades de los usuarios con el tiempo.

Es importante señalar que la GenUI no es lo mismo que las herramientas de diseño asistidas por IA, como aquellas que proporcionan sugerencias o automatizan ciertas tareas de diseño. Si bien estas herramientas pueden ser útiles para agilizar el proceso de diseño, todavía dependen de los diseñadores para tomar decisiones finales y crear interfaces estáticas. La GenUI, por otro lado, implica que el sistema de IA tome un papel más activo en la generación y adaptación real de interfaces basadas en datos y contexto del usuario.

La GenUI representa un cambio significativo en cómo abordamos el diseño de interfaces de usuario, alejándonos de soluciones universales y avanzando hacia experiencias altamente personalizadas y adaptativas. Al aprovechar el poder de la IA, la GenUI tiene el potencial de revolucionar la forma en que interactuamos con productos y servicios digitales, creando interfaces que son más intuitivas, atractivas y efectivas para cada usuario individual.

Ejemplo

Para ilustrar el concepto de GenUI, consideremos una aplicación hipotética de fitness llamada “FitAI”. Esta aplicación tiene como objetivo proporcionar planes de entrenamiento personalizados y consejos de nutrición a los usuarios basados en sus objetivos individuales, niveles de condición física y preferencias.

En un enfoque tradicional de diseño de UI, FitAI podría tener un conjunto fijo de pantallas y elementos que son los mismos para todos los usuarios. Sin embargo, con GenUI, la interfaz de la aplicación podría adaptarse dinámicamente a las necesidades únicas y al contexto de cada usuario.

Este enfoque es algo difícil de imaginar implementar en 2024 y podría incluso no tener un ROI adecuado, pero es posible.

Así es como podría funcionar:

1. Incorporación:

- En lugar de un cuestionario estándar, FitAI utiliza una IA conversacional para recopilar información sobre los objetivos del usuario, su nivel actual de condición física y preferencias.
- Basándose en esta interacción inicial, la IA genera un diseño de panel personalizado, destacando las características y la información más relevante para los objetivos del usuario.
- La tecnología de IA actual podría tener una selección de componentes de pantalla a su disposición para usar en la composición del panel personalizado.
- La tecnología de IA futura podría asumir el papel de un diseñador de UI experimentado y realmente crear el panel *desde cero*.

2. Planificador de ejercicios:

- La interfaz del planificador de ejercicios es adaptada por la IA específicamente según el nivel de experiencia del usuario y el equipo disponible.
- Para un principiante sin equipo, podría mostrar ejercicios simples con el peso corporal con instrucciones detalladas y videos.
- Para un usuario avanzado con acceso a un gimnasio, podría mostrar rutinas más complejas con menos contenido explicativo.
- El contenido del planificador de ejercicios no es simplemente filtrado de un gran superconjunto. Puede generarse *sobre la marcha* basándose en una base de conocimientos que se consulta con un contexto que incluye todo lo que se sabe sobre el usuario.

3. Seguimiento de progreso:

- La interfaz de seguimiento de progreso evoluciona según los objetivos y patrones de participación del usuario.
- Si un usuario está principalmente enfocado en la pérdida de peso, la interfaz podría mostrar de manera prominente una gráfica de tendencia de peso y estadísticas de quema de calorías.
- Para un usuario que está desarrollando músculo, podría resaltar las ganancias de fuerza y los cambios en la composición corporal.
- La IA puede adaptar esta parte de la aplicación al progreso real del usuario. Si el progreso se detiene durante un período de tiempo, la aplicación puede cambiar a un modo donde intenta persuadir al usuario para que revele las razones del retroceso, con el fin de mitigarlas.

4. Consejos nutricionales:

- La sección de nutrición se adapta a las preferencias y restricciones dietéticas del usuario.
- Para un usuario vegano, podría mostrar sugerencias de comidas basadas en plantas y fuentes de proteínas.
- Para un usuario con intolerancia al gluten, automáticamente filtraría los alimentos que contienen gluten de las recomendaciones.
- Nuevamente, el contenido no se extrae de un superconjunto masivo de datos de comidas que se aplica a todos los usuarios, sino que se sintetiza a partir de una base de conocimientos que contiene información adaptable según la situación y restricciones específicas del usuario.
- Por ejemplo, las recetas se generan con especificaciones de ingredientes que coinciden con las necesidades calóricas en constante cambio del usuario a medida que evoluciona su nivel de condición física y estadísticas corporales.

5. Elementos motivacionales:

- El contenido motivacional y las notificaciones de la aplicación se personalizan según el tipo de personalidad del usuario y su respuesta a diferentes estrategias motivacionales.

- Algunos usuarios podrían recibir mensajes de ánimo, mientras que otros obtienen retroalimentación más orientada a datos.

En este ejemplo, GenUI permite a FitAI crear una experiencia altamente personalizada para cada usuario, potencialmente aumentando el compromiso, la satisfacción y la probabilidad de alcanzar objetivos de fitness. Los elementos de la interfaz, el contenido e incluso la “personalidad” de la aplicación se adaptan para servir mejor a las necesidades y preferencias de cada usuario individual.

El Cambio hacia el Diseño Orientado a Resultados

GenUI representa un cambio fundamental en el enfoque del diseño de interfaces de usuario, pasando de un enfoque en la creación de elementos específicos de interfaz a un enfoque más holístico y orientado a resultados. Este cambio tiene varias implicaciones importantes:

1. Enfoque en los Objetivos del Usuario:

- Los diseñadores necesitarán pensar más profundamente sobre los objetivos del usuario y los resultados deseados en lugar de componentes específicos de la interfaz.
- El énfasis estará en crear sistemas que puedan generar interfaces que ayuden a los usuarios a alcanzar sus objetivos de manera eficiente y efectiva.
- Surgirán nuevos marcos de trabajo de UI que proporcionarán a los diseñadores basados en IA las herramientas necesarias para generar experiencias de usuario *sobre la marcha y desde cero* en lugar de basarse en especificaciones de pantalla predefinidas.

2. Cambio en el Rol de los Diseñadores:

- Los diseñadores pasarán de crear diseños fijos a definir reglas, restricciones y pautas que los sistemas de IA deben seguir al generar interfaces.
- Necesitarán desarrollar habilidades en áreas como análisis de datos, ingeniería de prompts y pensamiento sistémico para guiar efectivamente los sistemas GenUI.

3. Importancia de la Investigación de Usuarios:

- La investigación de usuarios se vuelve aún más crítica en un contexto de GenUI, ya que los diseñadores necesitan entender no solo las preferencias del usuario, sino también cómo estas preferencias y necesidades cambian en diferentes contextos.
- Las pruebas continuas con usuarios y los ciclos de retroalimentación serán esenciales para refinar y mejorar la capacidad de la IA para generar interfaces efectivas.

4. Diseñando para la Variabilidad:

- En lugar de crear una única interfaz “perfecta”, los diseñadores necesitarán considerar múltiples variaciones posibles y asegurar que el sistema pueda generar interfaces apropiadas para diversas necesidades de usuario.
- Esto incluye diseñar para casos extremos y asegurar que las interfaces generadas mantengan la usabilidad y accesibilidad en diferentes configuraciones.
- La diferenciación de productos adquiere nuevas dimensiones que involucran perspectivas divergentes sobre la psicología del usuario y el aprovechamiento de conjuntos de datos únicos y bases de conocimiento no disponibles para los competidores.

Desafíos y Consideraciones

Si bien GenUI ofrece posibilidades emocionantes, también presenta varios desafíos y consideraciones:

1. Limitaciones Técnicas:

- La tecnología de IA actual, aunque avanzada, todavía tiene limitaciones para comprender intenciones complejas del usuario y generar interfaces verdaderamente conscientes del contexto.
- Problemas de rendimiento relacionados con la generación en tiempo real de elementos de interfaz, especialmente en dispositivos menos potentes.

2. Requisitos de Datos:

- Dependiendo del caso de uso, los sistemas GenUI efectivos podrían requerir cantidades significativas de datos de usuario para generar interfaces personalizadas.
- Los desafíos en la obtención ética de datos auténticos de usuarios plantean preocupaciones sobre la privacidad y seguridad de los datos, así como posibles sesgos en los datos utilizados para entrenar modelos GenUI.

3. Usabilidad y Consistencia:

- Al menos hasta que la práctica se generalice, una aplicación con interfaces en constante cambio podría generar problemas de usabilidad, ya que los usuarios pueden tener dificultades para encontrar elementos familiares o navegar eficientemente.
- Será crucial encontrar un equilibrio entre la personalización y el mantenimiento de una interfaz consistente y fácil de aprender.

4. Dependencia Excesiva de la IA:

- Existe el riesgo de delegar excesivamente las decisiones de diseño a los sistemas de IA, lo que potencialmente puede llevar a elecciones de interfaz poco inspiradas, problemáticas o simplemente defectuosas.
- La supervisión humana y la capacidad de anular los diseños generados por IA seguirán siendo importantes en el futuro previsible.

5. Preocupaciones de Accesibilidad:

- Garantizar que las interfaces generadas dinámicamente sigan siendo accesibles para usuarios con discapacidades presenta desafíos completamente nuevos, lo cual es preocupante dado el bajo nivel de cumplimiento de accesibilidad demostrado por los sistemas típicos.
- Por otro lado, los diseñadores de IA pueden implementarse con una preocupación *incorporada* por la accesibilidad y capacidades para construir interfaces accesibles sobre la marcha, al igual que construyen interfaces para usuarios sin discapacidades.
- En cualquier caso, los sistemas GenUI deberían diseñarse con pautas y procesos de prueba de accesibilidad robustos.

6. Confianza y Transparencia del Usuario:

- Los usuarios pueden sentirse incómodos con interfaces que parecen “saber demasiado” sobre ellos o que cambian de formas que no comprenden.
- Proporcionar transparencia sobre cómo y por qué se personalizan las interfaces será importante para construir la confianza del usuario.

Perspectivas Futuras y Oportunidades

El futuro de la Interfaz de Usuario Generativa (GenUI) tiene un inmenso potencial para revolucionar la forma en que interactuamos con productos y servicios digitales.

A medida que esta tecnología continúa evolucionando, podemos anticipar un cambio sísmico en cómo se diseñan, implementan y experimentan las interfaces de usuario. Creo que GenUI es el fenómeno que finalmente impulsará nuestro software al ámbito de lo que ahora se considera ciencia ficción.

Una de las perspectivas más emocionantes de GenUI es su potencial para mejorar la accesibilidad a una escala que va más allá de simplemente asegurarse de que las personas con discapacidades graves no estén completamente excluidas del uso del software. Al adaptar automáticamente las interfaces a las necesidades individuales de los usuarios, GenUI podría hacer que las experiencias digitales sean más inclusivas que nunca. Imagine interfaces que se ajustan sin problemas para proporcionar texto más grande para usuarios jóvenes o con discapacidad visual, o diseños simplificados para aquellos con discapacidades cognitivas, todo sin requerir configuración manual o versiones “accesibles” separadas de las aplicaciones.

Es probable que las capacidades de personalización de GenUI impulsen un mayor compromiso, satisfacción y lealtad del usuario en una amplia gama de productos digitales. A medida que las interfaces se sintonizan más con las preferencias y comportamientos individuales, los usuarios encontrarán las experiencias digitales más intuitivas y agradables, lo que potencialmente conducirá a interacciones más profundas y significativas con la tecnología.

GenUI también tiene el potencial de transformar el proceso de incorporación de nuevos usuarios. Al crear experiencias intuitivas y personalizadas para usuarios principiantes que se adaptan rápidamente al nivel de experiencia de cada usuario, GenUI podría reducir significativamente la curva de aprendizaje asociada con nuevas aplicaciones. Esto podría conducir a tasas de adopción más rápidas y mayor confianza de los usuarios para explorar nuevas características y funcionalidades.

Otra posibilidad emocionante es la capacidad de GenUI para mantener una experiencia de usuario consistente en diferentes dispositivos y plataformas mientras se optimiza para cada contexto específico de uso. Esto podría resolver el desafío de larga data de

proporcionar experiencias coherentes a través de un panorama de dispositivos cada vez más fragmentado, desde teléfonos inteligentes y tabletas hasta computadoras de escritorio y tecnologías emergentes como las gafas de realidad aumentada.

La naturaleza basada en datos de GenUI abre oportunidades para la iteración rápida y la mejora en el diseño de interfaces de usuario. Al recopilar datos en tiempo real sobre cómo los usuarios interactúan con las interfaces generadas, los diseñadores y desarrolladores pueden obtener información sin precedentes sobre el comportamiento y las preferencias de los usuarios. Este ciclo de retroalimentación podría conducir a mejoras continuas en el diseño de UI, impulsadas por patrones de uso reales en lugar de suposiciones o pruebas de usuario limitadas.

Para prepararse para este cambio, los diseñadores necesitarán evolucionar sus habilidades y mentalidades. El enfoque cambiará de crear diseños fijos a desarrollar sistemas de diseño y pautas integrales que puedan informar la generación de interfaces impulsada por IA. Los diseñadores necesitarán cultivar una comprensión profunda del análisis de datos, tecnologías de IA y pensamiento sistémico para guiar efectivamente los sistemas GenUI.

Además, a medida que GenUI desdibuja las líneas entre diseño y tecnología, los diseñadores necesitarán colaborar más estrechamente con desarrolladores y científicos de datos. Este enfoque interdisciplinario será crucial para crear sistemas GenUI que no solo sean visualmente atractivos y fáciles de usar, sino también técnicamente robustos y éticamente sólidos.

Las implicaciones éticas de GenUI también pasarán a primer plano a medida que la tecnología madure. Los diseñadores jugarán un papel crucial en el desarrollo de marcos de trabajo para el uso responsable de la IA en el diseño de interfaces, asegurando que la personalización mejore las experiencias de usuario sin comprometer la privacidad ni manipular el comportamiento del usuario de manera poco ética.

Al mirar hacia el futuro, GenUI presenta tanto oportunidades emocionantes como desafíos significativos. Tiene el potencial de crear experiencias digitales más intuitivas,

eficientes y satisfactorias para usuarios en todo el mundo. Si bien requerirá que los diseñadores se adapten y adquieran nuevas habilidades, también ofrece una oportunidad sin precedentes para dar forma al futuro de la interacción humano-computadora de maneras profundas y significativas. El camino hacia sistemas GenUI completamente desarrollados será sin duda complejo, pero las recompensas potenciales en términos de mejores experiencias de usuario y accesibilidad digital lo convierten en un futuro por el que vale la pena luchar.

Orquestación Inteligente de Flujos de Trabajo



En el ámbito del desarrollo de aplicaciones, los flujos de trabajo juegan un papel crucial en la definición de cómo se estructuran y ejecutan las tareas, procesos e interacciones del usuario. A medida que las aplicaciones se vuelven más complejas y las expectativas de los usuarios continúan aumentando, la necesidad de una orquestación inteligente y adaptativa de flujos de trabajo se vuelve cada vez más evidente.

El enfoque de “Orquestación Inteligente de Flujos de Trabajo” se centra en aprovechar los componentes de IA para orquestar y optimizar dinámicamente flujos de trabajo complejos dentro de las aplicaciones. El objetivo es crear aplicaciones que sean más eficientes, receptivas y adaptables a datos y contextos en tiempo real.

En este capítulo, exploraremos los principios y patrones clave que sustentan el enfoque de orquestación inteligente de flujos de trabajo. Consideraremos cómo se puede utilizar

la IA para enrutar tareas de manera inteligente, automatizar la toma de decisiones y adaptar dinámicamente los flujos de trabajo basándose en varios factores como el comportamiento del usuario, el rendimiento del sistema y las reglas de negocio. A través de ejemplos prácticos y escenarios del mundo real, demostraremos el potencial transformador de la IA en la racionalización y optimización de los flujos de trabajo de las aplicaciones.

Ya sea que esté construyendo aplicaciones empresariales con procesos de negocio intrincados o aplicaciones orientadas al consumidor con recorridos de usuario dinámicos, los patrones y técnicas discutidos en este capítulo lo equiparán con el conocimiento y las herramientas para crear flujos de trabajo inteligentes y eficientes que mejoren la experiencia general del usuario e impulsen el valor comercial.

Necesidad Empresarial

Los enfoques tradicionales para la gestión de flujos de trabajo a menudo se basan en reglas predefinidas y árboles de decisión estáticos, que pueden ser rígidos, inflexibles e incapaces de hacer frente a la naturaleza dinámica de las aplicaciones modernas.

Considere un escenario donde una aplicación de comercio electrónico necesita manejar un proceso complejo de cumplimiento de pedidos. El flujo de trabajo puede involucrar múltiples pasos como la validación del pedido, verificación de inventario, procesamiento de pagos, envío y notificaciones al cliente. Cada paso puede tener su propio conjunto de reglas, dependencias, integraciones externas y mecanismos de manejo de excepciones. Gestionar tal flujo de trabajo manualmente o a través de lógica codificada puede volverse rápidamente engorroso, propenso a errores y difícil de mantener.

Además, a medida que la aplicación escala y el número de usuarios concurrentes crece, el flujo de trabajo puede necesitar adaptarse y optimizarse según los datos en tiempo real y el rendimiento del sistema. Por ejemplo, durante períodos de alto tráfico, la aplicación puede necesitar ajustar dinámicamente el flujo de trabajo para priorizar ciertas tareas, asignar recursos eficientemente y asegurar una experiencia de usuario fluida.

Aquí es donde entra en juego el enfoque de “Orquestación Inteligente de Fluxos de Trabajo”. Al aprovechar los componentes de IA, los desarrolladores pueden crear fluxos de trabajo que son inteligentes, adaptativos y auto-optimizables. La IA puede analizar grandes cantidades de datos, aprender de experiencias pasadas y tomar decisiones informadas en tiempo real para orquestar el flujo de trabajo de manera efectiva.

Beneficios Clave

- 1. Mayor Eficiencia:** La IA puede optimizar la asignación de tareas, la utilización de recursos y la ejecución del flujo de trabajo, lo que lleva a tiempos de procesamiento más rápidos y una mejor eficiencia general.
- 2. Adaptabilidad:** Los fluxos de trabajo impulsados por IA pueden adaptarse dinámicamente a condiciones cambiantes, como fluctuaciones en la demanda de usuarios, rendimiento del sistema o requisitos comerciales, asegurando que la aplicación permanezca receptiva y resiliente.
- 3. Toma de Decisiones Automatizada:** La IA puede automatizar procesos complejos de toma de decisiones dentro del flujo de trabajo, reduciendo la intervención manual y minimizando el riesgo de errores humanos.
- 4. Personalización:** La IA puede analizar el comportamiento, las preferencias y el contexto del usuario para personalizar el flujo de trabajo y ofrecer experiencias adaptadas a usuarios individuales.
- 5. Escalabilidad:** Los fluxos de trabajo potenciados por IA pueden escalar sin problemas para manejar volúmenes crecientes de datos e interacciones de usuarios, sin comprometer el rendimiento o la fiabilidad.

En las siguientes secciones, exploraremos los patrones y técnicas clave que permiten la implementación de fluxos de trabajo inteligentes y mostraremos ejemplos del mundo real de cómo la IA está transformando la gestión de fluxos de trabajo en aplicaciones modernas.

Patrones Clave

Para implementar la orquestación inteligente de flujos de trabajo en aplicaciones, los desarrolladores pueden aprovechar varios patrones clave que aprovechan el poder de la IA. Estos patrones proporcionan un enfoque estructurado para diseñar y gestionar flujos de trabajo, permitiendo que las aplicaciones se adapten, optimicen y automaticen procesos basados en datos y contexto en tiempo real. Exploraremos algunos de los patrones fundamentales en la orquestación inteligente de flujos de trabajo.

Enrutamiento Dinámico de Tareas

Este patrón implica usar IA para enrutar inteligentemente tareas dentro de un flujo de trabajo basado en varios factores como la prioridad de la tarea, la disponibilidad de recursos y el rendimiento del sistema. Los algoritmos de IA pueden analizar las características de cada tarea, considerar el estado actual del sistema y tomar decisiones informadas para asignar tareas a los recursos o rutas de procesamiento más apropiados. El enrutamiento dinámico de tareas asegura que las tareas se distribuyan y ejecuten eficientemente, optimizando el rendimiento general del flujo de trabajo.

```
1 class TaskRouter
2   include Raix::ChatCompletion
3   include Raix::FunctionDispatch
4
5   attr_accessor :task
6
7   # list of functions that can be called by the AI entirely at its
8   # discretion depending on the task received
9
10  function :analyze_task_priority do
11    TaskPriorityAnalyzer.perform(task)
12  end
13
14  function :check_resource_availability, # ...
15  function :assess_system_performance, # ...
```

```
16     function :assign_task_to_resource, # ...
17
18     DIRECTIVE = "You are a task router, responsible for intelligently
19         assigning tasks to available resources based on priority, resource
20         availability, and system performance..."
21
22     def initialize(task)
23         self.task = task
24         transcript << { system: DIRECTIVE }
25         transcript << { user: task.to_json }
26     end
27
28     def perform
29         while task.unassigned?
30             chat_completion
31
32             # todo: add max loop counter and break
33         end
34
35         # capture the transcript for later analysis
36         task.update(routing_transcript: transcript)
37     end
38 end
```

Observe el bucle creado por la expresión `while` en la línea 29, que continúa solicitando al IA hasta que se asigna la tarea. En la línea 35, guardamos la transcripción de la tarea para análisis y depuración posterior, si fuera necesario.

Toma de Decisiones Contextual

Puede usar un código muy similar para tomar decisiones conscientes del contexto dentro de un flujo de trabajo. Al analizar puntos de datos relevantes como las preferencias del usuario, patrones históricos y entradas en tiempo real, los componentes de IA pueden determinar el curso de acción más apropiado en cada punto de decisión del flujo de trabajo. Adapte el comportamiento de su flujo de trabajo según el contexto específico de cada usuario o escenario, proporcionando experiencias personalizadas y optimizadas.

Composición Adaptativa del Flujo de Trabajo

Este patrón se centra en componer y ajustar dinámicamente los flujos de trabajo según los requisitos o condiciones cambiantes. La IA puede analizar el estado actual del flujo de trabajo, identificar cuellos de botella o ineficiencias, y modificar automáticamente la estructura del flujo de trabajo para optimizar el rendimiento. La composición adaptativa del flujo de trabajo permite que las aplicaciones evolucionen y mejoren continuamente sus procesos sin requerir intervención manual.

Manejo y Recuperación de Excepciones

El manejo y recuperación de excepciones son aspectos críticos de la orquestación inteligente de flujos de trabajo. Cuando se trabaja con componentes de IA y flujos de trabajo complejos, es esencial anticipar y manejar las excepciones de manera elegante para garantizar la estabilidad y fiabilidad del sistema.

Aquí hay algunas consideraciones y técnicas clave para el manejo y recuperación de excepciones en flujos de trabajo inteligentes:

1. **Propagación de Excepciones:** Implemente un enfoque consistente para propagar excepciones a través de los componentes del flujo de trabajo. Cuando ocurre una excepción dentro de un componente, debe ser capturada, registrada y propagada al orquestador o a un componente discreto responsable de manejar excepciones. La idea es centralizar el manejo de excepciones y evitar que las excepciones sean silenciosamente ignoradas, además de abrir posibilidades para el [Manejo Inteligente de Errores](#).
2. **Mecanismos de Reintento:** Los mecanismos de reintento ayudan a mejorar la resiliencia del flujo de trabajo y manejar las fallas intermitentes de manera elegante. Definitivamente trate de implementar mecanismos de reintento para excepciones transitorias o recuperables, como problemas de conectividad de red o

indisponibilidad de recursos que pueden reintentarse automáticamente después de un retraso especificado. Tener un orquestador o manejador de excepciones impulsado por IA significa que sus estrategias de reintento no tienen que ser mecánicas por naturaleza, dependiendo de algoritmos fijos como el retroceso exponencial. Puede dejar el manejo del reintento a la “discreción” del componente de IA responsable de decidir cómo manejar la excepción.

3. **Estrategias de Respaldo:** Si un componente de IA falla en proporcionar una respuesta válida o encuentra un error—una ocurrencia común dado su carácter de vanguardia—tenga un mecanismo de respaldo en su lugar para asegurar que el flujo de trabajo pueda continuar. Esto podría involucrar el uso de valores predeterminados, algoritmos alternativos, o un [Humano en el Ciclo](#) para tomar decisiones y mantener el flujo de trabajo en movimiento.
4. **Acciones Compensatorias:** Las directivas del orquestador deben incluir instrucciones sobre acciones compensatorias para manejar excepciones que no pueden resolverse automáticamente. Las acciones compensatorias son pasos tomados para deshacer o mitigar los efectos de una operación fallida. Por ejemplo, si falla un paso de procesamiento de pago, una acción compensatoria podría ser revertir la transacción y notificar al usuario. Las acciones compensatorias ayudan a mantener la consistencia e integridad de los datos frente a las excepciones.
5. **Monitoreo y Alertas de Excepciones:** Configure mecanismos de monitoreo y alertas para detectar y notificar a las partes interesadas relevantes sobre excepciones críticas. El orquestador puede ser consciente de umbrales y reglas para activar alertas cuando las excepciones excedan ciertos límites o cuando ocurran tipos específicos de excepciones. Esto permite la identificación y resolución proactiva de problemas antes de que impacten el sistema general.

Aquí hay un ejemplo de manejo y recuperación de excepciones en un componente de flujo de trabajo en Ruby:

```
1 class InventoryManager
2   def check_availability(order)
3     begin
4       # Perform inventory check logic
5       inventory = Inventory.find_by(product_id: order.product_id)
6       if inventory.available_quantity >= order.quantity
7         return true
8       else
9         raise InsufficientInventoryError,
10           "Insufficient inventory for product #{order.product_id}"
11       end
12     rescue InsufficientInventoryError => e
13       # Log the exception
14       logger.error("Inventory check failed: #{e.message}")
15
16       # Retry the operation after a delay
17       retry_count ||= 0
18       if retry_count < MAX_RETRIES
19         retry_count += 1
20         sleep(RETRY_DELAY)
21         retry
22       else
23         # Fallback to manual intervention
24         NotificationService.admin("Inventory check failed: Order #{order.id}")
25         return false
26       end
27     end
28   end
29 end
```

En este ejemplo, el componente `InventoryManager` verifica la disponibilidad de un producto para un pedido determinado. Si la cantidad disponible es insuficiente, lanza un `InsufficientInventoryError`. La excepción es capturada, registrada, y se implementa un mecanismo de reintento. Si se excede el límite de reintentos, el componente recurre a la intervención manual notificando a un administrador.

Mediante la implementación de mecanismos robustos de manejo y recuperación de excepciones, puedes asegurar que tus flujos de trabajo inteligentes sean resilientes,

mantenibles y capaces de manejar situaciones inesperadas de manera elegante.

Estos patrones forman la base de la orquestación de flujos de trabajo inteligentes y pueden combinarse y adaptarse para satisfacer los requisitos específicos de diferentes aplicaciones. Al aprovechar estos patrones, los desarrolladores pueden crear flujos de trabajo que son flexibles, resilientes y optimizados para el rendimiento y la experiencia del usuario.

En la siguiente sección, exploraremos cómo estos patrones pueden implementarse en la práctica, utilizando ejemplos del mundo real y fragmentos de código para ilustrar la integración de componentes de IA en la gestión de flujos de trabajo.

Implementación Práctica de la Orquestación de Flujos de Trabajo Inteligentes

Ahora que hemos explorado los patrones clave en la orquestación de flujos de trabajo inteligentes, profundicemos en cómo estos patrones pueden implementarse en aplicaciones del mundo real. Proporcionaremos ejemplos prácticos y fragmentos de código para ilustrar la integración de componentes de IA en la gestión de flujos de trabajo.

Procesador Inteligente de Pedidos

Profundicemos en un ejemplo práctico de implementación de orquestación de flujos de trabajo inteligentes utilizando un componente OrderProcessor potenciado por IA en una aplicación de comercio electrónico en Ruby on Rails. El OrderProcessor implementa el concepto de [Gestor de Procesos de Integración Empresarial](#) que encontramos por primera vez en el Capítulo 3 al discutir [Multitud de Trabajadores](#).

El componente será responsable de gestionar el flujo de trabajo de cumplimiento de pedidos, tomar decisiones de enrutamiento basadas en resultados intermedios y orquestar la ejecución de varios pasos de procesamiento.

El proceso de cumplimiento de pedidos involucra múltiples pasos como la validación del pedido, verificación de inventario, procesamiento de pagos y envío. Cada paso se implementa como un proceso trabajador separado que realiza una tarea específica y devuelve el resultado al `OrderProcessor`. Los pasos no son obligatorios, y ni siquiera necesitan realizarse en un orden preciso.

Aquí hay un ejemplo de implementación del `OrderProcessor`. Cuenta con dos mixins de [Raix](#). El primero (`ChatCompletion`) le da la capacidad de hacer completado de chat, lo que lo convierte en un componente de IA. El segundo (`FunctionDispatch`) habilita la llamada a funciones por parte de la IA, permitiéndole responder a un prompt con una invocación de función en lugar de un mensaje de texto.

Las funciones trabajadoras (`validate_order`, `check_inventory`, et al) delegan en sus respectivas clases trabajadoras, que pueden ser componentes de IA o no IA, con el único requisito de que devuelvan los resultados de su trabajo en un formato que pueda representarse como una cadena de texto.



Como con todos los otros ejemplos en esta parte del libro, este código es prácticamente pseudo-código y solo tiene la intención de transmitir el significado del patrón e inspirar tus propias creaciones. Las descripciones completas de los patrones y ejemplos de código completos se incluyen en la Parte 2.

```
1  class OrderProcessor
2      include Raix::ChatCompletion
3      include Raix::FunctionDispatch
4
5      SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."
6
7      def initialize(order)
8          self.order = order
9          transcript << { system: SYSTEM_DIRECTIVE }
10         transcript << { user: order.to_json }
11     end
12
13     def perform
14         # will continue looping until `stop_looping!` is called
15         chat_completion(loop: true)
16     end
17
18     # list of functions available to be called by the AI
19     # truncated for brevity
20
21     def functions
22         [
23             {
24                 name: "validate_order",
25                 description: "Invoke to check validity of order",
26                 parameters: {
27                     ...
28                 },
29                 ...
30             }
31         ]
32     end
33
34     # implementation of functions that can be called by the AI
35     # entirely at its discretion, depending on the needs of the order
36
37     def validate_order
38         OrderValidationWorker.perform(@order)
39     end
40
41     def check_inventory
42         InventoryCheckWorker.perform(@order)
43     end
```

```
43
44  def process_payment
45      PaymentProcessingWorker.perform(@order)
46  end
47
48  def schedule_shipping
49      ShippingSchedulerWorker.perform(@order)
50  end
51
52  def send_confirmation
53      OrderConfirmationWorker.perform(@order)
54  end
55
56  def finished_processing
57      @order.update!(transcript:, processed_at: Time.current)
58      stop_looping!
59  end
60 end
```

En el ejemplo, el OrderProcessor se inicializa con un objeto de orden y mantiene una transcripción de la ejecución del flujo de trabajo, en el formato típico de transcripción de conversación que es nativo de los modelos de lenguaje grandes. Se le da control completo a la IA para orquestar la ejecución de varios pasos de procesamiento, como la validación de la orden, la verificación del inventario, el procesamiento del pago y el envío.

Cada vez que se llama al método `chat_completion`, la transcripción se envía a la IA para que proporcione una respuesta como una llamada a función. Depende completamente de la IA analizar el resultado del paso anterior y determinar la acción apropiada a tomar. Por ejemplo, si la verificación del inventario revela niveles bajos de existencias, el OrderProcessor puede programar una tarea de reabastecimiento. Si el procesamiento del pago falla, puede iniciar un reintento o notificar al servicio de atención al cliente.

El ejemplo anterior no tiene funciones definidas para el reabastecimiento o la notificación al servicio de atención al cliente, pero definitivamente podría tenerlas.

La transcripción crece cada vez que se llama a una función y sirve como registro de la ejecución del flujo de trabajo, incluyendo los resultados de cada paso y las instrucciones generadas por la IA para los siguientes pasos. Esta transcripción puede utilizarse para depuración, auditoría y para proporcionar visibilidad en el proceso de cumplimiento de órdenes.

Al aprovechar la IA en el OrderProcessor, la aplicación de comercio electrónico puede adaptar dinámicamente el flujo de trabajo basándose en datos en tiempo real y manejar excepciones de manera inteligente. El componente de IA puede tomar decisiones informadas, optimizar el flujo de trabajo y asegurar un procesamiento fluido de las órdenes incluso en escenarios complejos.

El hecho de que el único requisito para los procesos trabajadores sea devolver una salida inteligible para que la IA considere qué hacer a continuación, podría empezar a hacerte dar cuenta de cómo este enfoque puede reducir el trabajo de mapeo de entrada/salida que típicamente está involucrado cuando se integran sistemas dispares entre sí.

Moderador de Contenido Inteligente

Las aplicaciones de redes sociales generalmente requieren al menos una moderación de contenido mínima para asegurar una comunidad segura y saludable. Este ejemplo del componente ContentModerator aprovecha la IA para orquestar de manera inteligente el flujo de trabajo de moderación, tomando decisiones basadas en las características del contenido y los resultados de varios pasos de moderación.

El proceso de moderación involucra múltiples pasos como el análisis de texto, reconocimiento de imágenes, evaluación de la reputación del usuario y revisión

manual. Cada paso se implementa como un proceso trabajador separado que realiza una tarea específica y devuelve el resultado al ContentModerator.

Aquí hay un ejemplo de implementación del ContentModerator:

```
1 class ContentModerator
2   include Raix::ChatCompletion
3   include Raix::FunctionDispatch
4
5   SYSTEM_DIRECTIVE = "You are a content moderator process manager,
6     tasked with the workflow involved in moderating user-generated content..."
7
8   def initialize(content)
9     @content = content
10    @transcript = [
11      { system: SYSTEM_DIRECTIVE },
12      { user: content.to_json }
13    ]
14  end
15
16  def perform
17    complete(@transcript)
18  end
19
20  def model
21    "openai/gpt-4"
22  end
23
24  # list of functions available to be called by the AI
25  # truncated for brevity
26
27  def functions
28    [
29      {
30        name: "analyze_text",
31        # ...
32      },
33      {
34        name: "recognize_image",
35        description: "Invoke to describe images...",
36        # ...
37      },
38    ]
39  end
```

```
38      {
39          name: "assess_user_reputation",
40          # ...
41      },
42      {
43          name: "escalate_to_manual_review",
44          # ...
45      },
46      {
47          name: "approve_content",
48          # ...
49      },
50      {
51          name: "reject_content",
52          # ...
53      }
54  ]
55 end
56
57 # implementation of functions that can be called by the AI
58 # entirely at its discretion, depending on the needs of the order
59
60 def analyze_text
61     result = TextAnalysisWorker.perform(@content)
62     continue_with(result)
63 end
64
65 def recognize_image
66     result = ImageRecognitionWorker.perform(@content)
67     continue_with(result)
68 end
69
70 def assess_user_reputation
71     result = UserReputationWorker.perform(@content.user)
72     continue_with(result)
73 end
74
75 def escalate_to_manual_review
76     ManualReviewWorker.perform(@content)
77     @content.update!(status: 'pending', transcript: @transcript)
78 end
79
```

```
80  def approve_content
81      @content.update!(status: 'approved', transcript: @transcript)
82  end
83
84  def reject_content
85      @content.update!(status: 'rejected', transcript: @transcript)
86  end
87
88  private
89
90  def continue_with(result)
91      @transcript << { function: result }
92      complete(@transcript)
93  end
94 end
```

En este ejemplo, el ContentModerator se inicializa con un objeto de contenido y mantiene un registro de moderación en formato de conversación. El componente de IA tiene control total sobre el flujo de trabajo de moderación, decidiendo qué pasos ejecutar según las características del contenido y los resultados de cada paso.

Las funciones trabajadoras disponibles que la IA puede invocar incluyen `analyze_text`, `recognize_image`, `assess_user_reputation` y `escalate_to_manual_review`. Cada función delega la tarea a un proceso trabajador correspondiente (`TextAnalysisWorker`, `ImageRecognitionWorker`, etc.) y añade el resultado al registro de moderación, con la excepción de la función de escalamiento, que actúa como un estado final. Finalmente, las funciones `approve_content` y `reject_content` también actúan como estados finales.

El componente de IA analiza el contenido y determina la acción apropiada a tomar. Si el contenido contiene referencias a imágenes, puede llamar al trabajador `recognize_image` para asistir con una revisión visual. Si algún trabajador advierte sobre contenido potencialmente dañino, la IA puede decidir escalar el contenido para revisión manual o simplemente rechazarlo directamente. Pero dependiendo de la gravedad de la advertencia, la IA puede optar por utilizar los resultados de la evaluación

de reputación del usuario para decidir cómo manejar el contenido sobre el que no está segura. Dependiendo del caso de uso, quizás los usuarios confiables tengan más libertad en lo que pueden publicar. Y así sucesivamente...

Al igual que en el ejemplo anterior del gestor de procesos, el registro de moderación sirve como un registro de la ejecución del flujo de trabajo, incluyendo los resultados de cada paso y las decisiones generadas por la IA. Este registro puede utilizarse para auditoría, transparencia y mejora del proceso de moderación a lo largo del tiempo.

Al aprovechar la IA en el ContentModerator, la aplicación de redes sociales puede adaptar dinámicamente el flujo de trabajo de moderación basándose en las características del contenido y manejar escenarios de moderación complejos de manera inteligente. El componente de IA puede tomar decisiones informadas, optimizar el flujo de trabajo y asegurar una experiencia comunitaria segura y saludable.

Exploraremos dos ejemplos más que demuestran la programación predictiva de tareas y el manejo y recuperación de excepciones dentro del contexto de la orquestación inteligente de flujos de trabajo.

Programación Predictiva de Tareas en un Sistema de Soporte al Cliente

En una aplicación de soporte al cliente construida con Ruby on Rails, gestionar y priorizar eficientemente los tickets de soporte es crucial para proporcionar asistencia oportuna a los clientes. El componente SupportTicketScheduler aprovecha la IA para programar y asignar de manera predictiva los tickets de soporte a los agentes disponibles basándose en varios factores como la urgencia del ticket, la experiencia del agente y la carga de trabajo.

```
1  class SupportTicketScheduler
2      include Raix::ChatCompletion
3      include Raix::FunctionDispatch
4
5      SYSTEM_DIRECTIVE = "You are a support ticket scheduler,
6          tasked with intelligently assigning tickets to available agents..."
7
8      def initialize(ticket)
9          @ticket = ticket
10         @transcript = [
11             { system: SYSTEM_DIRECTIVE },
12             { user: ticket.to_json }
13         ]
14     end
15
16     def perform
17         complete(@transcript)
18     end
19
20     def model
21         "openai/gpt-4"
22     end
23
24     def functions
25         [
26             {
27                 name: "analyze_ticket_urgency",
28                 # ...
29             },
30             {
31                 name: "list_available_agents",
32                 description: "Includes expertise of available agents",
33                 # ...
34             },
35             {
36                 name: "predict_agent_workload",
37                 description: "Uses historical data to predict upcoming workloads",
38                 # ...
39             },
40             {
41                 name: "assign_ticket_to_agent",
42                 # ...
```

```
43      },
44      {
45        name: "reschedule_ticket",
46        # ...
47      }
48    ]
49  end
50
51  # implementation of functions that can be called by the AI
52  # entirely at its discretion, depending on the needs of the order
53
54  def analyze_ticket_urgency
55    result = TicketUrgencyAnalyzer.perform(@ticket)
56    continue_with(result)
57  end
58
59  def list_available_agents
60    result = ListAvailableAgents.perform
61    continue_with(result)
62  end
63
64  def predict_agent_workload
65    result = AgentWorkloadPredictor.perform
66    continue_with(result)
67  end
68
69  def assign_ticket_to_agent
70    TicketAssigner.perform(@ticket, @transcript)
71  end
72
73  def delay_assignment(until)
74    until = DateTimeStandardizer.process(until)
75    SupportTicketScheduler.delay(@ticket, @transcript, until)
76  end
77
78  private
79
80  def continue_with(result)
81    @transcript << { function: result }
82    complete(@transcript)
83  end
84 end
```

En este ejemplo, el `SupportTicketScheduler` se inicializa con un objeto de ticket de soporte y mantiene un registro de programación. El componente de IA analiza los detalles del ticket y programa predictivamente la asignación del ticket basándose en factores como la urgencia del ticket, la experiencia del agente y la carga de trabajo prevista del agente.

Las funciones disponibles que la IA puede invocar incluyen `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload`, y `assign_ticket_to_agent`. Cada función delega la tarea a un componente analizador o predictor correspondiente y añade el resultado al registro de programación. La IA también tiene la opción de retrasar la asignación utilizando la función `delay_assignment`.

El componente de IA examina el registro de programación y toma decisiones informadas sobre la asignación de tickets. Considera la urgencia del ticket, la experiencia de los agentes disponibles y la carga de trabajo prevista de cada agente para determinar el agente más adecuado para manejar el ticket.

Al aprovechar la programación predictiva de tareas, la aplicación de atención al cliente puede optimizar la asignación de tickets, reducir los tiempos de respuesta y mejorar la satisfacción general del cliente. La gestión proactiva y eficiente de los tickets de soporte asegura que los tickets correctos sean asignados a los agentes correctos en el momento adecuado.

Manejo de Excepciones y Recuperación en un Pipeline de Procesamiento de Datos

El manejo de excepciones y la recuperación de fallos son esenciales para garantizar la integridad de los datos y prevenir la pérdida de datos. El componente `DataProcessingOrchestrator` utiliza IA para manejar inteligentemente las excepciones y orquestar el proceso de recuperación en un pipeline de procesamiento de datos.

```
1 class DataProcessingOrchestrator
2   include Raix::ChatCompletion
3   include Raix::FunctionDispatch
4
5   SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."
6
7   def initialize(data_batch)
8     @data_batch = data_batch
9     @transcript = [
10       { system: SYSTEM_DIRECTIVE },
11       { user: data_batch.to_json }
12     ]
13   end
14
15   def perform
16     complete(@transcript)
17   end
18
19   def model
20     "openai/gpt-4"
21   end
22
23   def functions
24     [
25       {
26         name: "validate_data",
27         # ...
28       },
29       {
30         name: "process_data",
31         # ...
32       },
33       {
34         name: "request_fix",
35         # ...
36       },
37       {
38         name: "retry_processing",
39         # ...
40       },
41       {
42         name: "mark_data_as_failed",
```

```
43      # ...
44    },
45    {
46      name: "finished",
47      # ...
48    }
49  ]
50 end
51
52 # implementation of functions that can be called by the AI
53 # entirely at its discretion, depending on the needs of the order
54
55 def validate_data
56   result = DataValidator.perform(@data_batch)
57   continue_with(result)
58 rescue ValidationException => e
59   handle_validation_exception(e)
60 end
61
62 def process_data
63   result = DataProcessor.perform(@data_batch)
64   continue_with(result)
65 rescue ProcessingException => e
66   handle_processing_exception(e)
67 end
68
69 def request_fix(description_of_fix)
70   result = SmartDataFixer.new(description_of_fix, @data_batch)
71   continue_with(result)
72 end
73
74 def retry_processing(timeout_in_seconds)
75   wait(timeout_in_seconds)
76   process_data
77 end
78
79 def mark_data_as_failed
80   @data_batch.update!(status: 'failed', transcript: @transcript)
81 end
82
83 def finished
84   @data_batch.update!(status: 'finished', transcript: @transcript)
```

```
85     end
86
87     private
88
89     def continue_with(result)
90         @transcript << { function: result }
91         complete(@transcript)
92     end
93
94     def handle_validation_exception(exception)
95         @transcript << { exception: exception.message }
96         complete(@transcript)
97     end
98
99     def handle_processing_exception(exception)
100        @transcript << { exception: exception.message }
101        complete(@transcript)
102    end
103 end
```

En este ejemplo, el `DataProcessingOrchestrator` se inicializa con un objeto de lote de datos y mantiene un registro de procesamiento. El componente de IA orquesta el pipeline de procesamiento de datos, manejando excepciones y recuperándose de fallos según sea necesario.

Las funciones disponibles para que la IA las invoque incluyen `validate_data`, `process_data`, `request_fix`, `retry_processing`, y `mark_data_as_failed`. Cada función delega la tarea a un componente correspondiente de procesamiento de datos y añade el resultado o los detalles de la excepción al registro de procesamiento.

Si ocurre una excepción de validación durante el paso `validate_data`, la función `handle_validation_exception` añade los datos de la excepción al registro y devuelve el control a la IA. De manera similar, si ocurre una excepción de procesamiento durante el paso `process_data`, la IA puede decidir sobre la estrategia de recuperación.

Dependiendo de la naturaleza de la excepción encontrada, la IA puede, a su discreción,

decidir llamar a `request_fix`, que delega a un componente `SmartDataFixer` impulsado por IA (ver el capítulo de Datos de Auto-reparación). El reparador de datos recibe una descripción en lenguaje natural de cómo debería modificar el `@data_batch` para que el procesamiento pueda ser reintentado. ¿Tal vez un reinicio exitoso implicaría eliminar registros del lote de datos que no han pasado la validación y/o copiarlos a un pipeline de procesamiento diferente para revisión humana? Las posibilidades son casi infinitas.

Al incorporar el manejo de excepciones y la recuperación impulsada por IA, la aplicación de procesamiento de datos se vuelve más resiliente y tolerante a fallos. El `DataProcessingOrchestrator` gestiona inteligentemente las excepciones, minimiza la pérdida de datos y asegura la ejecución fluida del flujo de trabajo de procesamiento de datos.

Monitoreo y Registro

El monitoreo y registro proporcionan visibilidad sobre el progreso, rendimiento y salud de los componentes del flujo de trabajo impulsados por IA, permitiendo a los desarrolladores rastrear y analizar el comportamiento del sistema. Implementar mecanismos efectivos de monitoreo y registro es esencial para la depuración, auditoría y mejora continua de los flujos de trabajo inteligentes.

Monitoreando el Progreso y Rendimiento del Flujo de Trabajo

Para asegurar la ejecución fluida de los flujos de trabajo inteligentes, es importante monitorear el progreso y rendimiento de cada componente del flujo de trabajo. Esto implica rastrear métricas y eventos clave a lo largo del ciclo de vida del flujo de trabajo.

Algunos aspectos importantes para monitorear incluyen:

1. **Tiempo de Ejecución del Flujo de Trabajo:** Medir el tiempo que toma cada componente del flujo de trabajo para completar su tarea. Esto ayuda a identificar cuellos de botella en el rendimiento y optimizar la eficiencia general del flujo de trabajo.
2. **Utilización de Recursos:** Monitorear la utilización de recursos del sistema, como CPU, memoria y almacenamiento, por cada componente del flujo de trabajo. Esto ayuda a asegurar que el sistema está operando dentro de su capacidad y puede manejar la carga de trabajo efectivamente.
3. **Tasas de Error y Excepciones:** Rastrear la ocurrencia de errores y excepciones dentro de los componentes del flujo de trabajo. Esto ayuda a identificar problemas potenciales y permite el manejo proactivo de errores y la recuperación.
4. **Puntos de Decisión y Resultados:** Monitorear los puntos de decisión dentro del flujo de trabajo y los resultados de las decisiones impulsadas por IA. Esto proporciona información sobre el comportamiento y la efectividad de los componentes de IA.

Los datos capturados por los procesos de monitoreo pueden mostrarse en tableros o utilizarse como entradas para informes programados que informan a los administradores del sistema sobre la salud del sistema.



¡Los datos de monitoreo pueden alimentar a un proceso de administrador de sistema impulsado por IA para su revisión y posible acción!

Registrando Eventos y Decisiones Clave

El registro es una práctica esencial que implica capturar y almacenar información relevante sobre eventos clave, decisiones y excepciones que ocurren durante la ejecución del flujo de trabajo.

Algunos aspectos importantes para registrar incluyen:

- 1. Iniciación y Finalización del Flujo de Trabajo:** Registrar los tiempos de inicio y fin de cada instancia del flujo de trabajo, junto con cualquier metadato relevante como los datos de entrada y el contexto del usuario.
- 2. Ejecución de Componentes:** Registrar los detalles de ejecución de cada componente del flujo de trabajo, incluyendo los parámetros de entrada, resultados de salida y cualquier dato intermedio generado.
- 3. Decisiones y Razonamiento de IA:** Registrar las decisiones tomadas por los componentes de IA, junto con el razonamiento subyacente o los puntajes de confianza. Esto proporciona transparencia y permite la auditoría de decisiones impulsadas por IA.
- 4. Excepciones y Mensajes de Error:** Registrar cualquier excepción o mensaje de error encontrado durante la ejecución del flujo de trabajo, incluyendo el trazado de pila e información contextual relevante.

El registro puede implementarse utilizando varias técnicas, como escribir en archivos de registro, almacenar registros en una base de datos o enviar registros a un servicio de registro centralizado. Es importante elegir un marco de registro que proporcione flexibilidad, escalabilidad y fácil integración con la arquitectura de la aplicación.

Aquí hay un ejemplo de cómo se puede implementar el registro en una aplicación Ruby on Rails utilizando la clase `ActiveSupport::Logger`:

```
1 class WorkflowLogger
2   def self.log(message, severity = :info)
3     @logger ||= ActiveSupport::Logger.new('workflow.log')
4     @logger.formatter ||= proc do |severity, datetime, progname, msg|
5       "#{datetime} [#{severity}] #{msg}\n"
6     end
7     @logger.send(severity, message)
8   end
9 end
10
11 # Usage example
12 WorkflowLogger.log("Workflow initiated for order ##{@order.id}")
13 WorkflowLogger.log("Payment processing completed successfully")
14 WorkflowLogger.log("Inventory check failed for item ##{@item.id}", :error)
```

Mediante la colocación estratégica de declaraciones de registro a lo largo de los componentes del flujo de trabajo y los puntos de decisión de la IA, los desarrolladores pueden capturar información valiosa para la depuración, auditoría y análisis.

Beneficios del Monitoreo y Registro

La implementación del monitoreo y registro en la orquestación inteligente de flujos de trabajo ofrece varios beneficios:

- 1. Depuración y Resolución de Problemas:** Los registros detallados y los datos de monitoreo ayudan a los desarrolladores a identificar y diagnosticar problemas rápidamente. Proporcionan información sobre el flujo de ejecución del trabajo, las interacciones entre componentes y cualquier error o excepción encontrada.
- 2. Optimización del Rendimiento:** El monitoreo de métricas de rendimiento permite a los desarrolladores identificar cuellos de botella y optimizar los componentes del flujo de trabajo para una mejor eficiencia. Al analizar los tiempos de ejecución, la utilización de recursos y otras métricas, los desarrolladores pueden tomar decisiones informadas para mejorar el rendimiento general del sistema.
- 3. Auditoría y Cumplimiento:** El registro de eventos y decisiones clave proporciona un rastro de auditoría para el cumplimiento normativo y la rendición de cuentas. Permite a las organizaciones rastrear y verificar las acciones tomadas por los componentes de IA y asegurar el cumplimiento de las reglas comerciales y los requisitos legales.
- 4. Mejora Continua:** Los datos de monitoreo y registro sirven como entradas valiosas para la mejora continua de los flujos de trabajo inteligentes. Al analizar datos históricos, identificar patrones y medir la efectividad de las decisiones de IA, los desarrolladores pueden refinar y mejorar iterativamente la lógica de orquestación del flujo de trabajo.

Consideraciones y Mejores Prácticas

Al implementar el monitoreo y registro en la orquestación inteligente de flujos de trabajo, considere las siguientes mejores prácticas:

- 1. Definir Métricas Claras de Monitoreo:** Identifique las métricas y eventos clave que deben monitorearse según los requisitos específicos del flujo de trabajo. Concéntrese en métricas que proporcionen información significativa sobre el rendimiento, la salud y el comportamiento del sistema.
- 2. Implementar Registro Granular:** Asegúrese de que las declaraciones de registro se coloquen en puntos apropiados dentro de los componentes del flujo de trabajo y los puntos de decisión de IA. Capture información contextual relevante, como parámetros de entrada, resultados de salida y cualquier dato intermedio generado.
- 3. Usar Registro Estructurado:** Adopte un formato de registro estructurado para facilitar el análisis y el procesamiento de los datos de registro. El registro estructurado permite una mejor capacidad de búsqueda, filtrado y agregación de entradas de registro.
- 4. Gestionar la Retención y Rotación de Registros:** Implemente políticas de retención y rotación de registros para gestionar el almacenamiento y el ciclo de vida de los archivos de registro. Determine el período de retención apropiado según los requisitos legales, las restricciones de almacenamiento y las necesidades de análisis. Si es posible, delegue el registro a un servicio de terceros como [Papertrail](#).
- 5. Proteger la Información Sensible:** Sea cauteloso al registrar información sensible, como información de identificación personal (IIP) o datos comerciales confidenciales. Implemente medidas de seguridad apropiadas, como el enmascaramiento de datos o el cifrado, para proteger la información sensible en los archivos de registro.
- 6. Integrar con Herramientas de Monitoreo y Alertas:** Aproveche las herramientas de monitoreo y alertas para centralizar la recopilación, análisis y visualización de datos de monitoreo y registro. Estas herramientas pueden proporcionar información en tiempo real, generar alertas basadas en umbrales predefinidos y facilitar la detección y resolución proactiva de problemas. Mi herramienta favorita entre estas es [Datadog](#).

Mediante la implementación de mecanismos integrales de monitoreo y registro, los desarrolladores pueden obtener información valiosa sobre el comportamiento y rendimiento de los flujos de trabajo inteligentes. Estos conocimientos permiten una

depuración efectiva, optimización y mejora continua de los sistemas de orquestación de flujos de trabajo impulsados por IA.

Consideraciones de Escalabilidad y Rendimiento

La escalabilidad y el rendimiento son aspectos críticos a considerar al diseñar e implementar sistemas de orquestación de flujos de trabajo inteligentes. A medida que aumenta el volumen de flujos de trabajo concurrentes y la complejidad de los componentes impulsados por IA, se vuelve esencial asegurar que el sistema pueda manejar la carga de trabajo de manera eficiente y escalar sin problemas para satisfacer las demandas crecientes.

Manejo de Altos Volúmenes de Flujos de Trabajo Concurrentes

Los sistemas de orquestación de flujos de trabajo inteligentes a menudo necesitan manejar un gran número de flujos de trabajo concurrentes. Para garantizar la escalabilidad, considere las siguientes estrategias:

- 1. Procesamiento Asíncrono:** Implemente mecanismos de procesamiento asíncrono para desacoplar la ejecución de los componentes del flujo de trabajo. Esto permite que el sistema maneje múltiples flujos de trabajo concurrentemente sin bloquear o esperar a que cada componente se complete. El procesamiento asíncrono se puede lograr utilizando colas de mensajes, arquitecturas basadas en eventos o marcos de procesamiento de trabajos en segundo plano como Sidekiq.
- 2. Arquitectura Distribuida:** Diseñe la arquitectura del sistema para utilizar componentes sin servidor (como AWS Lambda) o simplemente distribuya la carga de trabajo entre múltiples nodos o servidores junto con su servidor de aplicaciones principal. Esto permite la escalabilidad horizontal, donde se pueden agregar nodos adicionales para manejar mayores volúmenes de flujos de trabajo.

3. Ejecución Paralela: Identifique oportunidades para la ejecución paralela dentro de los flujos de trabajo. Algunos componentes del flujo de trabajo pueden ser independientes entre sí y pueden ejecutarse concurrentemente. Al aprovechar técnicas de procesamiento paralelo, como multi-threading o colas de tareas distribuidas, el sistema puede optimizar la utilización de recursos y reducir el tiempo total de ejecución del flujo de trabajo.

Optimización del Rendimiento de Componentes Basados en IA

Los componentes basados en IA, como los modelos de aprendizaje automático o los motores de procesamiento de lenguaje natural, pueden ser computacionalmente intensivos e impactar el rendimiento general del sistema de orquestación de flujos de trabajo. Para optimizar el rendimiento de los componentes de IA, considera las siguientes técnicas:

- 1. Almacenamiento en caché:** Si tu procesamiento de IA es puramente generativo y no involucra búsquedas de información en tiempo real o integraciones externas para generar sus completaciones de chat, entonces puedes considerar mecanismos de caché para almacenar y reutilizar los resultados de operaciones frecuentemente accedidas o computacionalmente costosas.
- 2. Optimización de Modelos:** Optimiza continuamente la forma en que utilizas los modelos de IA en los componentes del flujo de trabajo. Esto puede involucrar técnicas como la *Destilación de Prompts* o simplemente puede ser cuestión de probar nuevos modelos conforme estén disponibles.
- 3. Procesamiento por Lotes:** Si estás trabajando con modelos clase GPT-4, podrías aprovechar las técnicas de procesamiento por lotes para procesar múltiples puntos de datos o solicitudes en un solo lote, en lugar de procesarlos individualmente. Al procesar datos en lotes, el sistema puede optimizar la utilización de recursos y reducir la sobrecarga de solicitudes repetidas al modelo.

Monitoreo y Perfilado del Rendimiento

Para identificar cuellos de botella en el rendimiento y optimizar la escalabilidad del sistema de orquestación de flujos de trabajo inteligente, es crucial implementar mecanismos de monitoreo y perfilado. Considera los siguientes enfoques:

- 1. Métricas de Rendimiento:** Define y rastrea métricas clave de rendimiento, como tiempo de respuesta, capacidad de procesamiento, utilización de recursos y latencia. Estas métricas proporcionan información sobre el rendimiento del sistema y ayudan a identificar áreas de optimización. El agregador de modelos de IA popular [OpenRouter](#) incluye métricas de Host¹ y Velocidad² en cada respuesta de API, haciendo trivial el seguimiento de estas métricas clave.
- 2. Herramientas de Perfilado:** Utiliza herramientas de perfilado para analizar el rendimiento de componentes individuales del flujo de trabajo y operaciones de IA. Las herramientas de perfilado pueden ayudar a identificar puntos críticos de rendimiento, rutas de código inefficientes u operaciones que consumen muchos recursos. Las herramientas de perfilado populares incluyen New Relic, Scout, o perfiladores incorporados proporcionados por el lenguaje de programación o marco de trabajo.
- 3. Pruebas de Carga:** Realiza pruebas de carga para evaluar el rendimiento del sistema bajo diferentes niveles de cargas de trabajo concurrentes. Las pruebas de carga ayudan a identificar los límites de escalabilidad del sistema, detectar la degradación del rendimiento y asegurar que el sistema pueda manejar el tráfico esperado sin comprometer el rendimiento.
- 4. Monitoreo Continuo:** Implementa mecanismos de monitoreo continuo y alertas para detectar proactivamente problemas y cuellos de botella de rendimiento. Configura tableros de monitoreo y alertas para rastrear indicadores clave de rendimiento (KPIs)

¹Host es el tiempo que tomó recibir el primer byte de la generación transmitida desde el host del modelo, también conocido como “tiempo hasta el primer byte.”

²La velocidad se calcula como el número de tokens de completación dividido por el tiempo total de generación. Para solicitudes no transmitidas, la latencia se considera parte del tiempo de generación.

y recibir notificaciones cuando se superen los umbrales predefinidos. Esto permite la identificación y resolución rápida de problemas de rendimiento.

Estrategias de Escalado

Para manejar cargas de trabajo crecientes y asegurar la escalabilidad del sistema de orquestación de flujos de trabajo inteligente, considera las siguientes estrategias de escalado:

- 1. Escalado Vertical:** El escalado vertical implica aumentar los recursos (por ejemplo, CPU, memoria) de nodos o servidores individuales para manejar cargas de trabajo más altas. Este enfoque es adecuado cuando el sistema requiere más poder de procesamiento o memoria para manejar flujos de trabajo complejos u operaciones de IA.
- 2. Escalado Horizontal:** El escalado horizontal implica agregar más nodos o servidores al sistema para distribuir la carga de trabajo. Este enfoque es efectivo cuando el sistema necesita manejar un gran número de flujos de trabajo concurrentes o cuando la carga de trabajo puede distribuirse fácilmente entre múltiples nodos. El escalado horizontal requiere una arquitectura distribuida y mecanismos de balanceo de carga para asegurar una distribución uniforme del tráfico.
- 3. Autoescalado:** Implementa mecanismos de autoescalado para ajustar automáticamente el número de nodos o recursos según la demanda de carga de trabajo. El autoescalado permite que el sistema escale dinámicamente hacia arriba o hacia abajo dependiendo del tráfico entrante, asegurando una utilización óptima de recursos y eficiencia en costos. Plataformas en la nube como Amazon Web Services (AWS) o Google Cloud Platform (GCP) proporcionan capacidades de autoescalado que pueden aprovecharse para sistemas de orquestación de flujos de trabajo inteligentes.

Técnicas de Optimización de Rendimiento

Además de las estrategias de escalado, considera las siguientes técnicas de optimización de rendimiento para mejorar la eficiencia del sistema de orquestación de flujos de trabajo

inteligente:

1. **Almacenamiento y Recuperación Eficiente de Datos:** Optimiza los mecanismos de almacenamiento y recuperación de datos utilizados por los componentes del flujo de trabajo. Utiliza indexación eficiente de bases de datos, técnicas de optimización de consultas y almacenamiento en caché de datos para minimizar la latencia y mejorar el rendimiento de las operaciones intensivas en datos.
2. **E/S asíncrona:** Utilice operaciones de E/S asíncronas para evitar bloqueos y mejorar la capacidad de respuesta del sistema. La E/S asíncrona permite que el sistema maneje múltiples solicitudes simultáneamente sin esperar a que se completen las operaciones de E/S, maximizando así la utilización de recursos.
3. **Serialización y deserialización eficiente:** Optimice los procesos de serialización y deserialización utilizados para el intercambio de datos entre componentes del flujo de trabajo. Utilice formatos de serialización eficientes, como Protocol Buffers o MessagePack, para reducir la sobrecarga de la serialización de datos y mejorar el rendimiento de la comunicación entre componentes.



Para aplicaciones basadas en Ruby, considere usar [Universal ID](#). Universal ID aprovecha tanto MessagePack como Brotli (una combinación diseñada para velocidad y compresión de datos de primera clase). Cuando se combinan, estas bibliotecas son hasta un 30% más rápidas y mantienen tasas de compresión entre 2-5% en comparación con Protocol Buffers.

4. **Compresión y codificación:** Aplique técnicas de compresión y codificación para reducir el tamaño de los datos transferidos entre los componentes del flujo de trabajo. Los algoritmos de compresión, como gzip o Brotli, pueden reducir significativamente el uso del ancho de banda de red y mejorar el rendimiento general del sistema.

Al considerar aspectos de escalabilidad y rendimiento durante el diseño e implementación de sistemas de orquestación de fluxos de trabajo inteligentes, puede asegurar que su sistema pueda manejar altos volúmenes de fluxos de trabajo

concurrentes, optimizar el rendimiento de los componentes basados en IA y escalar sin problemas para satisfacer las demandas crecientes. El monitoreo continuo, el perfilado y los esfuerzos de optimización son esenciales para mantener el rendimiento y la capacidad de respuesta del sistema a medida que la carga de trabajo y la complejidad aumentan con el tiempo.

Pruebas y validación de flujos de trabajo

Las pruebas y la validación son aspectos críticos del desarrollo y mantenimiento de sistemas de orquestación de flujos de trabajo inteligentes. Dada la naturaleza compleja de los flujos de trabajo basados en IA, es esencial asegurar que cada componente funcione según lo esperado, que el flujo de trabajo general se comporte correctamente y que las decisiones de IA sean precisas y confiables. En esta sección, exploraremos varias técnicas y consideraciones para probar y validar flujos de trabajo inteligentes.

Pruebas unitarias de componentes del flujo de trabajo

Las pruebas unitarias implican probar componentes individuales del flujo de trabajo de forma aislada para verificar su corrección y robustez. Al realizar pruebas unitarias de componentes del flujo de trabajo basados en IA, considere lo siguiente:

- 1. Validación de entrada:** Pruebe la capacidad del componente para manejar diferentes tipos de entradas, incluyendo datos válidos e inválidos. Verifique que el componente maneje adecuadamente los casos límite y proporcione mensajes de error o excepciones apropiados.
- 2. Verificación de salida:** Asegure que el componente produzca la salida esperada para un conjunto dado de entradas. Compare la salida real con los resultados esperados para garantizar la corrección.
- 3. Gestión de errores:** Pruebe los mecanismos de gestión de errores del componente simulando varios escenarios de error, como entrada inválida, indisponibilidad de

recursos o excepciones inesperadas. Verifique que el componente capture y maneje los errores de manera apropiada.

4. Condiciones límite: Pruebe el comportamiento del componente bajo condiciones límite, como entrada vacía, tamaño máximo de entrada o valores extremos. Asegúrese de que el componente maneje estas condiciones de manera elegante sin fallar o producir resultados incorrectos.

Aquí hay un ejemplo de una prueba unitaria para un componente de flujo de trabajo en Ruby utilizando el marco de pruebas RSpec:

```
1 RSpec.describe OrderValidator do
2   describe '#validate' do
3     context 'when order is valid' do
4       let(:order) { build(:order) }
5
6       it 'returns true' do
7         expect(subject.validate(order)).to be true
8       end
9     end
10
11    context 'when order is invalid' do
12      let(:order) { build(:order, total_amount: -100) }
13
14      it 'returns false' do
15        expect(subject.validate(order)).to be false
16      end
17    end
18  end
19 end
```

En este ejemplo, el componente `OrderValidator` se prueba utilizando dos casos de prueba: uno para una orden válida y otro para una orden inválida. Los casos de prueba verifican que el método `validate` devuelve el valor booleano esperado según la validez de la orden.

Interacciones en las Pruebas de Integración del Flujo de Trabajo

Las pruebas de integración se centran en verificar las interacciones y el flujo de datos entre diferentes componentes del flujo de trabajo. Aseguran que los componentes trabajen juntos sin problemas y produzcan los resultados esperados. Al realizar pruebas de integración en flujos de trabajo inteligentes, considera lo siguiente:

- 1. Interacción de Componentes:** Prueba la comunicación y el intercambio de datos entre los componentes del flujo de trabajo. Verifica que la salida de un componente se pase correctamente como entrada al siguiente componente en el flujo de trabajo.
- 2. Consistencia de Datos:** Asegura que los datos permanezcan consistentes y precisos mientras fluyen a través del flujo de trabajo. Verifica que las transformaciones de datos, cálculos y agregaciones se realicen correctamente.
- 3. Propagación de Excepciones:** Prueba cómo las excepciones y errores se propagan y manejan a través de los componentes del flujo de trabajo. Verifica que las excepciones sean capturadas, registradas y manejadas apropiadamente para prevenir la interrupción del flujo de trabajo.
- 4. Comportamiento Asíncrono:** Si el flujo de trabajo involucra componentes asíncronos o ejecución paralela, prueba los mecanismos de coordinación y sincronización. Asegura que el flujo de trabajo se comporte correctamente en escenarios concurrentes y asíncronos.

Aquí hay un ejemplo de una prueba de integración para un flujo de trabajo en Ruby utilizando el marco de pruebas RSpec:

```
1 RSpec.describe OrderProcessingWorkflow do
2
3   let(:order) { build(:order) }
4
5   it 'processes the order successfully' do
6     expect(OrderValidator).to receive(:validate).and_return(true)
7     expect(InventoryManager).to receive(:check_availability).and_return(true)
8     expect(PaymentProcessor).to receive(:process_payment).and_return(true)
9     expect(ShippingService).to receive(:schedule_shipping).and_return(true)
10
11   workflow = OrderProcessingWorkflow.new(order)
12   result = workflow.process
13
14   expect(result).to be true
15   expect(order.status).to eq('processed')
16 end
17
18 end
```

En este ejemplo, el `OrderProcessingWorkflow` se prueba verificando las interacciones entre diferentes componentes del flujo de trabajo. El caso de prueba establece expectativas para el comportamiento de cada componente y asegura que el flujo de trabajo procese el pedido exitosamente, actualizando el estado del pedido de manera correspondiente.

Pruebas de Puntos de Decisión de IA

Probar los puntos de decisión de IA es crucial para garantizar la precisión y fiabilidad de los flujos de trabajo impulsados por IA. Al probar los puntos de decisión de IA, considere lo siguiente:

- 1. Precisión de Decisiones:** Verifique que el componente de IA tome decisiones precisas basadas en los datos de entrada y el modelo entrenado. Compare las decisiones de la IA con los resultados esperados o los datos de referencia.
- 2. Casos Límite:** Pruebe el comportamiento del componente de IA en casos límite y

escenarios inusuales. Verifique que el componente de IA maneje estos casos de manera elegante y tome decisiones razonables.

3. Sesgo y Equidad: Evalúe el componente de IA en busca de sesgos potenciales y asegúrese de que tome decisiones justas e imparciales. Pruebe el componente con datos de entrada diversos y analice los resultados en busca de patrones discriminatorios.

4. Explicabilidad: Si el componente de IA proporciona explicaciones o razonamientos para sus decisiones, verifique la corrección y claridad de las explicaciones. Asegúrese de que las explicaciones se alineen con el proceso de toma de decisiones subyacente.

Aquí hay un ejemplo de prueba de un punto de decisión de IA en Ruby utilizando el marco de pruebas RSpec:

```
1 RSpec.describe FraudDetector do
2   describe '#detect_fraud' do
3     context 'when transaction is fraudulent' do
4       let(:tx) do
5         build(:transaction, amount: 10_000, location: 'High-Risk Country')
6       end
7
8       it 'returns true' do
9         expect(subject.detect_fraud(tx)).to be true
10      end
11    end
12
13    context 'when transaction is legitimate' do
14      let(:tx) do
15        build(:transaction, amount: 100, location: 'Low-Risk Country')
16      end
17
18      it 'returns false' do
19        expect(subject.detect_fraud(tx)).to be false
20      end
21    end
22  end
23 end
```

En este ejemplo, el componente de IA FraudDetector se prueba con dos casos de

prueba: uno para una transacción fraudulenta y otro para una transacción legítima. Los casos de prueba verifican que el método `detect_fraud` devuelve el valor booleano esperado según las características de la transacción.

Pruebas de Extremo a Extremo

Las pruebas de extremo a extremo implican probar todo el flujo de trabajo de principio a fin, simulando escenarios e interacciones de usuario del mundo real. Aseguran que el flujo de trabajo se comporte correctamente y produzca los resultados deseados. Al realizar pruebas de extremo a extremo para flujos de trabajo inteligentes, considere lo siguiente:

- 1. Escenarios de Usuario:** Identifique escenarios comunes de usuario y pruebe el comportamiento del flujo de trabajo bajo estos escenarios. Verifique que el flujo de trabajo maneje correctamente las entradas del usuario, tome decisiones apropiadas y produzca las salidas esperadas.
- 2. Validación de Datos:** Asegure que el flujo de trabajo valide y depure las entradas del usuario para prevenir inconsistencias en los datos o vulnerabilidades de seguridad. Pruebe el flujo de trabajo con varios tipos de datos de entrada, incluyendo datos válidos e inválidos.
- 3. Recuperación de Errores:** Pruebe la capacidad del flujo de trabajo para recuperarse de errores y excepciones. Simule escenarios de error y verifique que el flujo de trabajo los maneje adecuadamente, registre los errores y tome las acciones de recuperación apropiadas.
- 4. Rendimiento y Escalabilidad:** Evalúe el rendimiento y la escalabilidad del flujo de trabajo bajo diferentes condiciones de carga. Pruebe el flujo de trabajo con un gran volumen de solicitudes concurrentes y mida los tiempos de respuesta, la utilización de recursos y la estabilidad general del sistema.

Aquí hay un ejemplo de una prueba de extremo a extremo para un flujo de trabajo

en Ruby utilizando el marco de pruebas RSpec y la biblioteca Capybara para simular interacciones de usuario:

```
1 RSpec.describe 'Order Processing Workflow' do
2   scenario 'User places an order successfully' do
3     visit '/orders/new'
4     fill_in 'Product', with: 'Sample Product'
5     fill_in 'Quantity', with: '2'
6     fill_in 'Shipping Address', with: '123 Main St'
7     click_button 'Place Order'
8
9     expect(page).to have_content('Order Placed Successfully')
10    expect(Order.count).to eq(1)
11    expect(Order.last.status).to eq('processed')
12  end
13 end
```

En este ejemplo, la prueba de extremo a extremo simula un usuario realizando un pedido a través de la interfaz web. Rellena los campos requeridos del formulario, envía el pedido y verifica que el pedido se procese correctamente, mostrando el mensaje de confirmación apropiado y actualizando el estado del pedido en la base de datos.

Integración y Despliegue Continuo

Para garantizar la fiabilidad y mantenibilidad de los flujos de trabajo inteligentes, se recomienda integrar las pruebas y la validación en el pipeline de integración y despliegue continuo (CI/CD). Esto permite realizar pruebas y validaciones automatizadas de los cambios en el flujo de trabajo antes de que se implementen en producción. Considere las siguientes prácticas:

- 1. Ejecución Automatizada de Pruebas:** Configure el pipeline de CI/CD para ejecutar automáticamente el conjunto de pruebas cada vez que se realicen cambios en el código base del flujo de trabajo. Esto asegura que cualquier regresión o fallo se detecte temprano en el proceso de desarrollo.

- 2. Monitoreo de Cobertura de Pruebas:** Mida y monitoree la cobertura de pruebas de los componentes del flujo de trabajo y los puntos de decisión de IA. Apunte a una alta cobertura de pruebas para asegurar que las rutas y escenarios críticos sean probados exhaustivamente.
- 3. Retroalimentación Continua:** Integre los resultados de las pruebas y las métricas de calidad del código en el flujo de trabajo de desarrollo. Proporcione retroalimentación continua a los desarrolladores sobre el estado de las pruebas, la calidad del código y cualquier problema detectado durante el proceso de CI/CD.
- 4. Entornos de Preproducción:** Despliegue el flujo de trabajo en entornos de preproducción que reflejen fielmente el entorno de producción. Realice pruebas y validaciones adicionales en el entorno de preproducción para detectar cualquier problema relacionado con la infraestructura, la configuración o la integración de datos.
- 5. Mecanismos de Reversión:** Implemente mecanismos de reversión en caso de fallos en el despliegue o problemas críticos detectados en producción. Asegúrese de que el flujo de trabajo pueda revertirse rápidamente a una versión estable anterior para minimizar el tiempo de inactividad y el impacto en los usuarios.

Mediante la incorporación de pruebas y validación a lo largo del ciclo de vida del desarrollo de flujos de trabajo inteligentes, las organizaciones pueden asegurar la fiabilidad, precisión y mantenibilidad de sus sistemas potenciados por IA. Las pruebas y validaciones regulares ayudan a detectar errores, prevenir regresiones y generar confianza en el comportamiento y los resultados del flujo de trabajo.

Parte 2: Los Patrones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ingeniería de Prompts

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cadena de Pensamiento

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Generación de Contenido

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Creación de Entidades Estructuradas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Guía para Agentes LLM

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios y Consideraciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Mode Switch

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cuándo Utilizarlo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Asignación de Rol

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cuándo Usarlo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Prompt Object

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Prompt Template

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios y Consideraciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cuándo Usarlo:

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Structured IO

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Escalando el Structured IO

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios y Consideraciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Encadenamiento de Prompts

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cuándo Usarlo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo: Incorporación de Olympia

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Reescritor de Prompts

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Response Fencing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios y Consideraciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Manejo de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Analizador de Consultas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Implementación

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Etiquetado de Partes del Discurso (POS) y Reconocimiento de Entidades Nombradas (NER)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Clasificación de Intención

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Extracción de Palabras Clave

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Reescritor de Consultas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ventriloquist

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cuándo Usarlo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Componentes Discretos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Predicado

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cuándo Utilizarlo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Fachada de API

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios Clave

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cuándo Usarla

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Autenticación y Autorización

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Manejo de Solicitudes

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Formateo de Respuestas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Manejo de Errores y Casos Límite

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Consideraciones de Escalabilidad y Rendimiento

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Comparación con Otros Patrones de Diseño

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Intérprete de Resultados

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cuándo Usarlo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Máquina Virtual

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cuándo Usarlo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Detrás de la Magia

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Especificación y Pruebas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Especificando el Comportamiento

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Escribiendo Casos de Prueba

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo: Probando el Componente Traductor

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Reproducción de Interacciones HTTP

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Human In The Loop (HITL)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Patrones de Alto Nivel

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Inteligencia Híbrida

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Respuesta Adaptativa

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Intercambio de Roles entre Humano e IA

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Escalamiento

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios Clave

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Aplicación en el Mundo Real: Atención Médica

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Bucle de Retroalimentación

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Aplicaciones y Ejemplos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Técnicas Avanzadas en la Integración de Retroalimentación Humana

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Radiación Pasiva de Información

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Visualización de Información Contextual

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Notificaciones Proactivas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Perspectivas Explicativas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Exploración Interactiva

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Beneficios Clave

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Aplicaciones y Ejemplos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Toma de Decisiones Colaborativa (CDM)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Aprendizaje Continuo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Aplicaciones y Ejemplos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Consideraciones Éticas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Rol del HITL en la Mitigación de Riesgos de IA

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Avances Tecnológicos y Perspectivas Futuras

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Desafíos y Limitaciones de los Sistemas HITL

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Manejo Inteligente de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Enfoques Tradicionales de Manejo de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Diagnóstico Contextual de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ingeniería de Prompts para el Diagnóstico Contextual de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Generación Aumentada por Recuperación para el Diagnóstico Contextual de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Informes Inteligentes de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Prevención Predictiva de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Recuperación Inteligente de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Comunicación Personalizada de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Flujo de Trabajo Adaptativo de Manejo de Errores

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Control de Calidad

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Eval

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Problema

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Solución

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Consideraciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Comprendiendo las Referencias Doradas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funcionan las Evaluaciones Sin Referencias

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Mecanismo de Protección

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Problema

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Solución

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Cómo Funciona

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Ejemplo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Consideraciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Guardarraíles y Evaluaciones: Dos Caras de la Misma Moneda

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

La Intercambiabilidad de Guardarraíles y Evaluaciones sin Referencia

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Implementación de Guardarraíles y Evaluaciones de Doble Propósito

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Glosario

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Glosario

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

A

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

B

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

C

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

D

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

E

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

F

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

G

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

H

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

I

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

J

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

K

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

L

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

M

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

N

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

O

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

P

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Q

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

R

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

S

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

T

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

U

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

V

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

W

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Z

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en <http://leanpub.com/patterns-of-application-development-using-ai-es>.

Index

- accesibilidad, 219, 220
- Agéntico, 33
- AI, 100, 204
 - modelo, 100
- ajuste fino, 81
- ajuste por instrucciones, 10
 - modelos ajustados por instrucciones, 51, 53
- almacenamiento en caché, 252
- Alpaca, 13
- Altman, Sam, 18
- Amazon Web Services, 254
- Anthropic, 23, 40, 74, 131, 138
- antropomorfismo, 69
- análisis de sentimiento, 17, 113, 115, 119, 136
- Análisis de Sentimiento del Cliente, 101
- análisis de sentimientos, 101, 114, 119, 147
- APIs, 72, 125, 155
- Aplicaciones de Comercio Electrónico, 92
- aplicaciones educativas, 33
- aplicaciones modernas, 225
- aplicación de chatbot, 120
- Apoyo a la Decisión Clínica, 105
- aprendizaje con pocos ejemplos, 63
- aprendizaje no supervisado, 4
- Aprendizaje One-Shot, 61
- aprendizaje zero-shot, 60
- arquitectura de aplicaciones empresariales, 39
- Arquitectura de microservicios, 90
- arquitectura de software, 2
- arquitectura dirigida por eventos, 110
- arquitectura distribuida, 251
- arquitectura transformer, 6
- arreglos, 132
- asignación de tickets, 242
- asistentes virtuales, 34
- auditoría y cumplimiento, 249
- autoescalado, 254
- base de conocimientos de Olympia, 92
- bases de conocimiento, 7
- bases de datos, 125
 - estrategias de bloqueo, 111
 - objeto respaldado, 106
- BERT, 14, 24
- biblioteca Capybara, 262
- Bloqueo del Intérprete Global (GIL), 117
- bloqueo optimista, 111
- bloqueo pesimista, 111
- boundary conditions, 257
- Brotli, 255

- Byte Pair Encoding (BPE), 15
- C (Lenguaje de Programación), 118
- Cadena de Pensamiento (CoT), 46
- cadena de suministro
- optimización, 33
- capacidades
- de toma de decisiones, 101
- casos límite, 59
- Cercado de Respuestas, 178
- Chain of Thought (CoT), 140
- chatbots de servicio al cliente, 34
- ChatGPT, 30, 54
- ciencias de la computación, 73
- clasificación, 54
- clasificadores por rangos, 36
- classification, 122
- Claude, 8, 45, 78
- Claude 3, 51, 129, 131, 136, 138
- Claude 3 Opus, 75
- Claude v1, 17
- Claude v2, 17
- Codificación por Pares de Bytes (BPE), 13
- Cohere (Proveedor de LLM), 25
- Cohere (Proveedor de MLG), 23
- coincidencia de patrones, 154
- comercio electrónico, 224
- completado de alto rendimiento, 26
- comportamiento determinista, 59
- computadoras de escritorio, 221
- concurrent workflows, 256
- conectividad de red, 228
- confianza del usuario, 219
- conjuntos, 120
- conjunto de trabajadores, 119
- consistencia
- y reproducibilidad, 134
- construcción narrativa, 20
- contenido
- Categorización de Contenido, 113
 - filtrado, 27
- contenido generado por usuarios, 113
- context
- Generación de Contenido Contextual, 194, 195
 - infinitely long inputs, 16
 - window, 16
- contexto
- Aumento, 47
 - Generación de Contenido Contextual, 189, 201, 202
 - Sugerencias Contextuales de Campo, 202
 - toma de decisiones contextual, 227
 - ventana, 227
- Continuación Automática, 162
- conversación
- bucle, 159, 161
 - transcripción, 159, 161
- Cuantización, 29
- cuellos de botella, 228
- cuenta, 92
- Datadog, 250

- datos Dohan, et al., 45
análisis, 35, 149
flujo, 111
integridad, 242
persistencia, 110
pipeline de procesamiento, 242
preparación, 110
privacidad, 27, 218
Recuperación de Datos, 111
Sincronización de Datos, 111
tareas de procesamiento, 127
Validación de Datos, 261
Datos Autorreparables, 166
Datos de Auto-reparación, 246
datos de entrenamiento, 43
datos en streaming, 154
datos estructurados, 136
decisión
 puntos, 247
 árboles, 224
depuración, 227
 y pruebas, 134
 y resolución de problemas, 249
desafíos conceptuales y prácticos, 201
desarrollo de aplicaciones, 223
descubrimientos médicos, 102
detección de fraude
 sistema, 98
diccionarios, 132
directiva del sistema, 100, 130
diseño y marcos de aplicaciones, 200
document clustering, 122
e-commerce, 194
ecosistema, 150
eficiencia, 225
ejecución paralela, 252
ELK stack, 112
empleados de Databricks, 53
encadenamiento de workers de IA, 113
Enrutamiento Dinámico de Tareas, 226
ensambles, 118
entornos de desarrollo local, 157
entornos de preproducción, 263
entrada
 prompts, 57
errores
 manejo, 108, 111, 144
 Manejo Inteligente de Errores, 145
 recuperación, 261
 tasas, 112
errores de sintaxis, 133
errors
 handling, 257
escalabilidad, 225, 251
escritura creativa, 35, 54
espacio latente, 41, 43
estrategias de respaldo, 111
estrategias de segmentación y orientación, 196
estrategias motivacionales, 215
Estratificación de Riesgo, 104
estrechar el camino, 40

- etiquetado de estilo markup, 71
- Evaluación y Estratificación de Síntomas, 102
- eventos enviados por el servidor (SSE), 152
- experiencia de usuario, 196
- experimentación
- marco, 196
- explicabilidad, 260
- F#, 93
- Facebook, 25
- factores de riesgo, 96, 97
- filtrado basado en contenido, 92
- filtrado colaborativo, 92
- FitAI, 213
- flexibilidad y creatividad, 198
- flujo de trabajo adaptativo
- Composición Adaptativa del Flujo de Trabajo, 228
- flujo de trabajo múltiple, 113
- función
- fallo en la llamada, 135
 - historial, 158
 - llamada, 125, 159
 - nombres, 156
- gafas de realidad aumentada, 221
- Gemma 7B, 11
- Generación Aumentada por Recuperación (RAG), 32, 47, 80, 127
- generación de datos sintéticos, 54
- generación dinámica de UI, 190
- generación intermodal, 22
- Generative Pre-trained Transformer (GPT), 8
- gestión del conocimiento, 33
- gestión del tráfico, 33
- Gestor de Procesos, 105, 108
- Integración Empresarial, 231
- GitLab, 93
- Google, 23
- API, 64, 66
 - Cloud AI Platform, 24
 - Cloud Platform, 254
 - Gemini, 22
 - Gemini 1.5 Pro, 14, 17, 19
 - PaLM (Pathways Language Model), 17, 24
 - T5, 14
- GPT-3, 13, 17
- GPT-4, 6, 13, 17, 22, 31, 45, 51, 64, 106, 118, 121, 129, 135, 205, 206, 252
- Graham, Paul, 19
- GraphQL, 109
- Groq, 26, 121
- gzip, 255
- hardware, 29
- hash, 154
- hiperparámetro, 48
- Hohpe, Gregor, 105
- Honeybadger, 95
- HTTP, 152
- Humano en el Ciclo, 181
- IA, 66, 75, 130, 136, 145, 152, 212

- aplicaciones, 127, 140, 151, 164
- conversacional, 7, 32, 214
- modelo, 90, 157, 158, 160, 212
- puntos de decisión, 259
- sistemas compuestos, 31, 35
- idioma
 - Detección de Idioma, 113
- indicaciones con pocos ejemplos, 64
- Inferencia, 5
- información
 - extracción, 54
 - recuperación, 7, 128
- informática, 71
- input
 - validation, 256
- Integración y Despliegue Continuo
 - (CI/CD), 262
 - pipeline, 262
- integrando LLMs, 190
- intelligent workflow orchestration, 255
- interacciones de juego de roles, 7
- interfaces controladas por voz, 34
- interfaces inclusivas, 201
- interfaz adaptativa, 210
- Interfaz de Usuario (UI)
 - diseño, 221
 - interfaces, 200, 216
 - marcos de trabajo, 216
 - tecnologías, 211
- Interfaz de Usuario Generativa (GenUI),
 - 220
- Interfaz Generativa (GenUI), 207, 208
- interfaz visual, 211
- internacionalización, 197
- intervención manual, 230
- Intérprete de Resultados, 144
- inyecciones SQL, 71
- JSON (JavaScript Object Notation), 128,
 - 132, 133, 136, 149, 169
- K-means, 123
- Large Language Model (LLM), 16, 122
- latencia, 28
- Latent Dirichlet Allocation, 123
- lenguaje
 - modelos, 44, 66, 73
 - tareas relacionadas, 5
- lenguaje natural
 - Procesamiento del Lenguaje Natural (PLN), 102
 - Limpieza de Texto, 113
- Llama, 13
- Llama 2-70B, 51
- Llama 3 70B, 11
- Llama 3 8B, 11
- llamada a herramienta, 155
- Louvre, 43
- línea de comandos
 - Interfaz de Línea de Comandos (CLI), 26
- lógica de interruptor automático, 164
- Managed Streaming for Apache Kafka, 42
- manejadores de flujo, 153

- manejo de excepciones, 228, 230
marcos de desarrollo, 151
Markdown, 149
mecanismos de reintento, 111
mecanismos de reversión, 263
Memorial Sloan Kettering Cancer Center, 42
mensaje disparador, 106
Mercurio (dios romano), 46
Mercurio (elemento), 46
Mercurio (planeta), 46
MessagePack, 255
Meta, 25
Metropolitan Museum of Art, 43
minoristas en línea, 207
Mistral, 26
 7B, 11
 7B Instruct, 17, 206
Mixtral
 8x22B, 11
 8x7B, 57
modelado autorregresivo, 44
Modelo de Lenguaje Grande (LLM), 1, 3,
 18, 30, 68, 69, 72, 76, 78, 112, 125,
 126, 136, 142, 146, 149, 166, 169,
 189, 200, 205, 211, 234
 panorama, 27
Modelo de Lenguaje Grande (MLG), 88, 146
modelos basados en recuperación, 7
modelos base, 55
modelos gráficos, 44
modelos probabilísticos, 44
Moderador de Contenido Inteligente, 235
modularidad, 89
monitoreo
 métricas, 250
 y alertas, 229
 y registro, 112, 249
Monitoreo Continuo de Riesgo, 104
Muestreo Top-k, 49
Muestreo Top-p (núcleo), 49
Multi-Agente
 Solucionadores de Problemas, 32
Multimodal
 modelos, 20
 modelos de lenguaje, 21
Multitud de Trabajadores, 120, 168
método finalize, 158, 160, 161
Naive Bayes, 122
narrow the path, 39
natural language
 Natural Language Processing (NLP),
 122
New Relic, 253
Ollama, 25
Olympia, 34, 63, 130, 145, 153, 169
OpenAI, 3, 23, 40, 74
OpenRouter, 28, 29, 153, 253
OPT model, 25
orquestación de flujos de trabajo
 inteligente, 253
orquestación de flujos de trabajo
 inteligentes, 231

- orquestación inteligente de flujos de trabajo, 223
- output verification, 256
- panorama digital, 196
- paráfrasis, 54
- parámetro
- efectos, 130
 - Número de Parámetros, 28
 - rango, 11
- parámetros de entrada, 130
- patrones clave, 226
- Patrones de Integración Empresarial, 105
- patrones históricos, 227
- penalizaciones de repetición, 52
- Penalización de presencia, 49
- Perplexity (Proveedor), 12
- personalización, 27, 190, 220, 225
 - Formularios Personalizados, 202
 - Microtextos Personalizados, 208
- planificación de respuesta a emergencias, 33
- predicciones, 5
- principio del mínimo privilegio, 72
- problemas de usabilidad, 218
- procesamiento asíncrono, 251
- procesamiento de flujo, 152
- procesamiento de stream
- lógica, 160
- procesamiento de streams, 158, 164
- procesamiento por lotes, 252
- proceso de destilación, 77
- Productividad, 192
- programación funcional, 93
- prompts
- Destilación de Prompts, 47, 74, 79, 252
 - diseño, 59, 69
 - encadenamiento, 60, 72
 - ingeniería, 42, 46, 47, 57, 60, 66, 68, 217
 - Objeto Prompt, 75
 - Plantilla de Prompt, 60
 - Prompt Template, 207
 - refinamiento, 69
- propiedades ACID, 111
- Protocol Buffers, 255
- proveedores de alojamiento de modelos de código abierto, 207
- pruebas de extremo a extremo, 261, 262
- pruebas de integración, 258
- pruebas de usuario y retroalimentación, 199
- psicología del usuario, 217
- PyTorch, 25
- Qwen2 70B, 11
- Rails, 197
- Railway Oriented Programming (ROP), 95
- Raix, 232
 - biblioteca, 98
- rastreo de métricas clave, 246
- Recomendaciones de Productos, 92
- recomendaciones de productos personalizadas, 92
- Recopilación del Historial Médico, 102
- redes neuronales, 4, 6

- refinamiento iterativo, 76, 146
registro de auditoría, 108
registro estructurado, 250
registro granular, 250
reglas de negocio, 224
reglas gramaticales, 4
regresión lineal, 44
rendimiento, 28
compensaciones, 5
optimización, 134, 198, 249
problemas, 254
Response Fencing, 207
respuesta a preguntas cerradas y abiertas, 54
resumen, 54
retención y rotación de registros, 250
Retrieval Augmented Generation (RAG), 39
retroalimentación
 Bucle de Retroalimentación, 60
revelación progresiva, 209
RSpec, 257, 258, 262
Ruby, 93, 94, 114, 165, 262
Ruby on Rails, 1, 113, 231, 239
Rudall, Alex, 24
Rust (Lenguaje de Programación), 93, 118

Scout, 253
Selección Dinámica de Herramientas, 133
Selección Forzada de Herramientas, 133
servicios o APIs externos, 128
sesgo
 y equidad en IA, 260
sin estado, 159
sistemas de pregunta-respuesta, 7
sistemas de publicación-suscripción, 109
soporte al cliente, 33
Stripe, 131
Structured IO, 207
Support Vector Machines (SVM), 122

T5, 24
tabletas, 221
tareas complejas, 148
teléfonos inteligentes, 221
Temperatura, 55
teoría de la mente, 41
tiempo de procesamiento, 112
Tiempo hasta el Primer Token (TTFT), 28
Together.ai, 26
tokenización, 13
tokens, 6, 13
toma de decisiones
 casos de uso, 135
tono emocional, 147
topic identification, 122
traducción, 17, 198
tragedia de los comunes, 193
Transformador Generativo Pre-entrenado (GPT), 68

UI Generativa (GenUI), 200, 212, 216
Unicode-encodable language, 15
Universal ID, 255
uso de herramientas, 125, 151

Ventrílocuo, 178

XML, 136

Verificación de Seguro, 103

Yi-34B, 51

votación por mayoría, 118

Wall, Larry, 3

álgebra lineal, 44

Wisper, 95, 108, 153, 160

ética

Wooley, Chad, 93

implicaciones, 201