



Introducción a LLMs y Agentes

Arquitecturas y Aplicaciones en Ingeniería de Software

IV ESCUELA DE INFORMÁTICA

J. ANDRÉS DÍAZ PACE
2025

Presentaciones



- Profesor titular en UNICEN (Tandil, Argentina)
- Arquitecto de soluciones y Científico de Datos Sr. en Globant (Argentina)
- Investigador principal en CONICET-Argentina
- Miembro del Software Engineering Institute (SEI, CMU) – 2007-2010
- Dr. in Ciencias de la Computación

andres.diazpace@isistan.unicen.edu.ar



Agenda tentativa

1. Introducción a ML/AI
2. Posibles aplicaciones a Ing/Arq. de Software
3. Conceptos básicos de LLMs y prompting
4. Búsqueda semántica y RAG
5. Workflows
6. Agentes y tools (MCP)
7. Perspectiva



IA, Machine Learning y GenAI

AI
ML

Algoritmos capaces de **realizar tareas humanas**,
de la misma forma que un humano

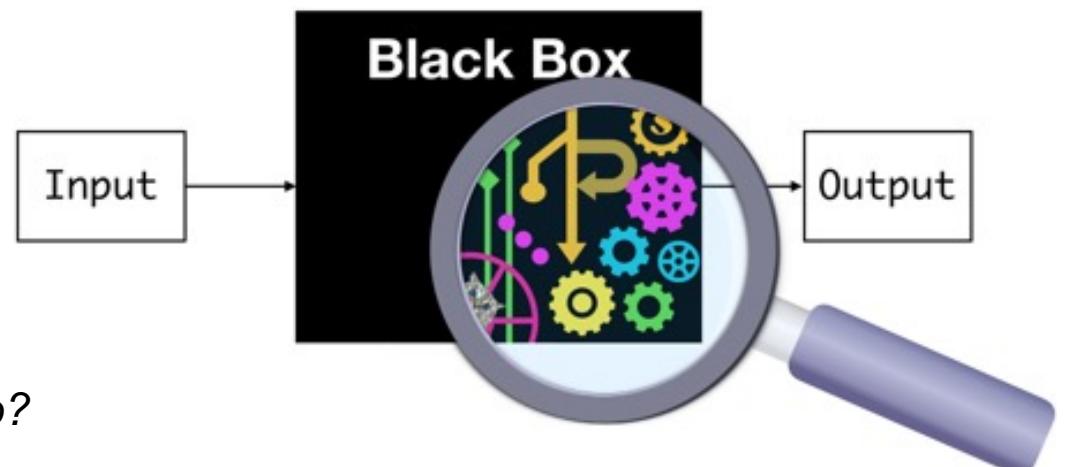
Algoritmos que **permiten a las computadoras aprender por sí mismas** cómo realizar una tarea humana, **sin ser programadas específicamente para ello**

- ML es un subconjunto de la AI
 - Permite detectar automáticamente patrones en los datos
 - Normalmente basado en modelado estadístico
 - El modelo es entrenado automáticamente
 - Y puede mejorarse en base a mayor cantidad de datos

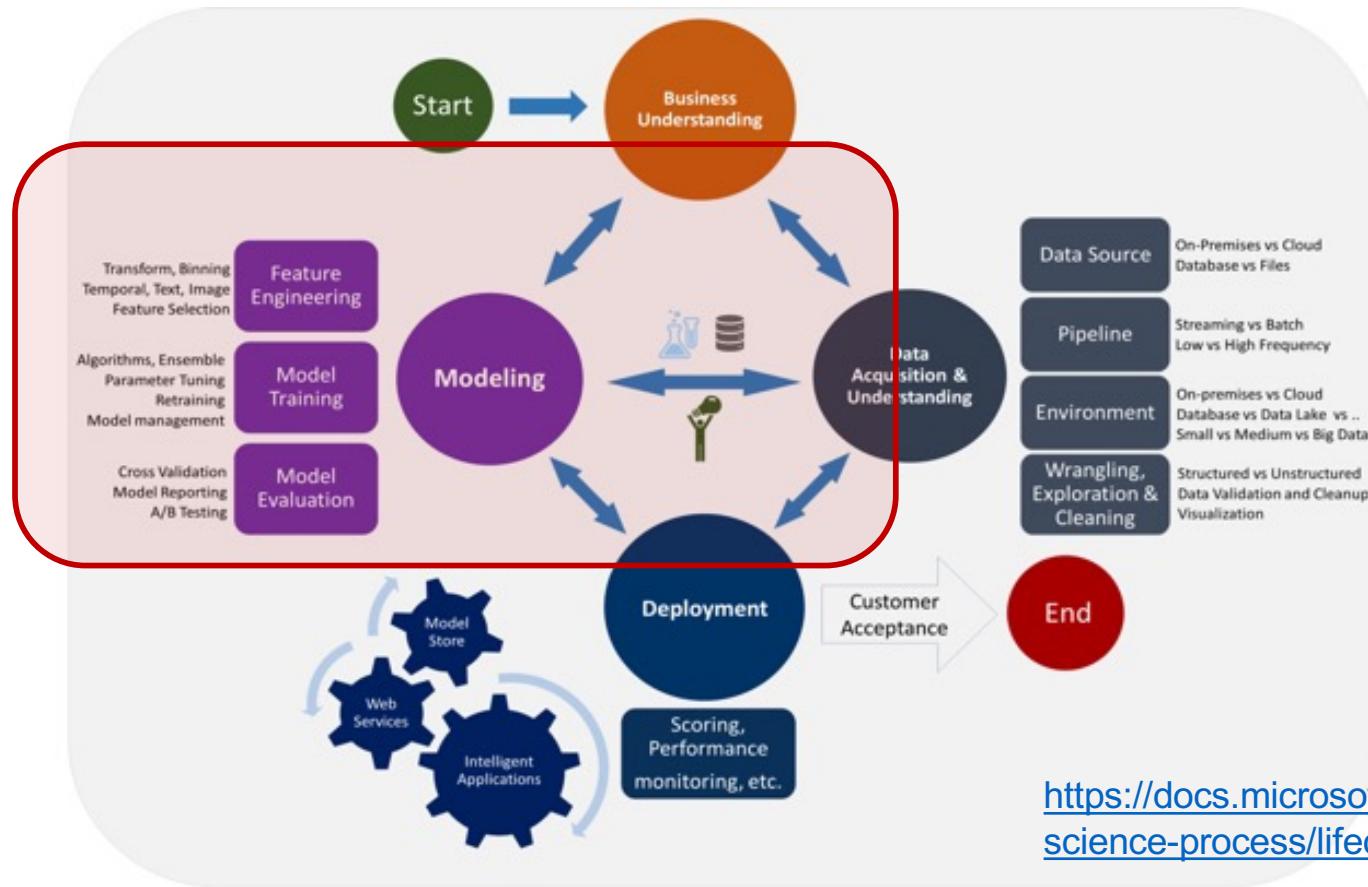


Noción de Modelo de ML

- Es una representación aproximada de un fenómeno complejo
 - Datos ciertos datos de entrada (features) produce una salida (target), que generalmente toma la forma de una predicción
 - Existen distintos algoritmos para construir/entrenar un modelo en base a un conjunto de datos
-
- Ejemplo1: ¿Cómo puedo estimar el peso estimado de una valija?
 - Ejemplo2 (Ing. de Soft.): ¿Cómo puedo estimar si un componente tendrá fallas o no? ¿Cómo puedo clasificar un requerimiento?



Modelo de Ciclo de Vida



Casos de Uso

- La estrategia de base es **data-driven**
- Análisis de imágenes de productos para clasificarlos de forma automática
- Análisis y clasificación automática de texto (por ej., noticias, tweets, spam, hate speech)
- Resúmenes automáticos de documentos (textuales)
- Chatbots para asistentes personales
- Predicción de indicadores (en base a información histórica)
- Detección de fraudes, o anomalías en datos
- Agrupamiento de items u entidades en base a distintos criterios (clustering)
- Sistemas de recomendación
- Recuperar documentos relevantes para un tema o consulta
- Q&A sobre documentos

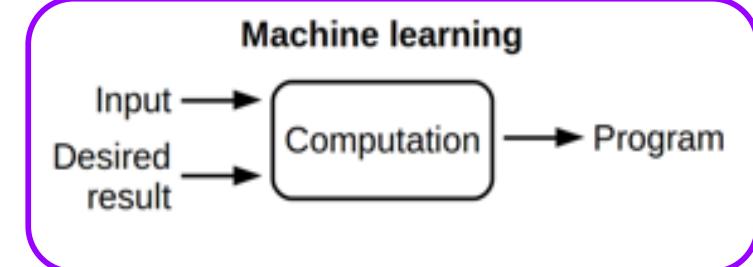
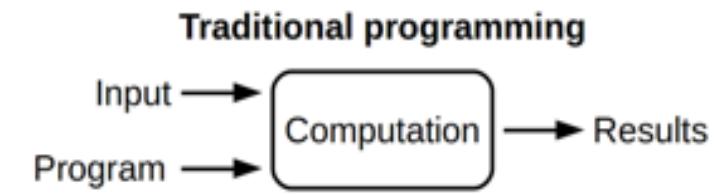


Cosas que no son ML/IA ...

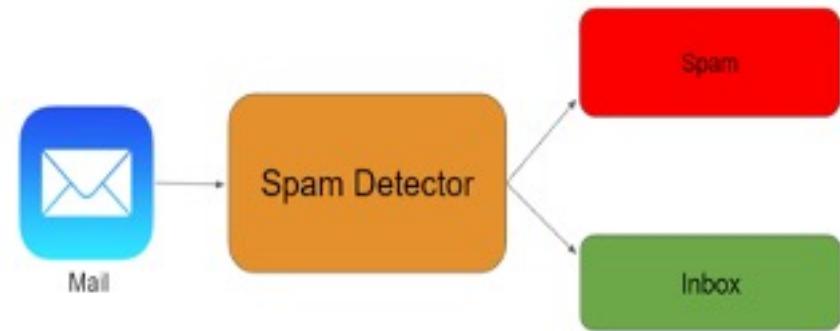


- Buscar en una base de datos
- Aplicación de heurísticas
- Código hardcodeado
- Sistemas basados en reglas

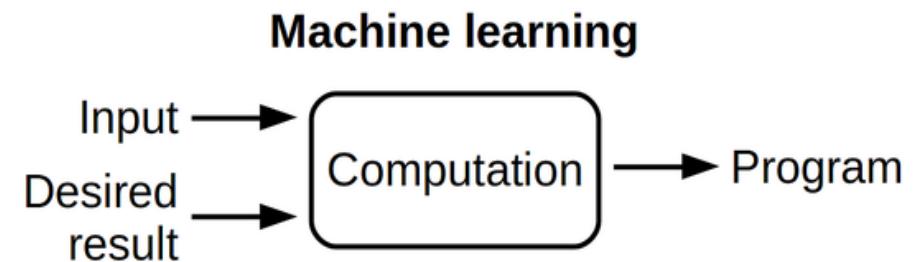
(por mas que pueda parecer “inteligente”
desde la percepción del usuario)



Ejemplo: Clasificación de texto



- La asignación de una etiqueta depende del uso de ciertas palabras en el texto, y en su contexto
- Aplicable también a artefactos textuales de software

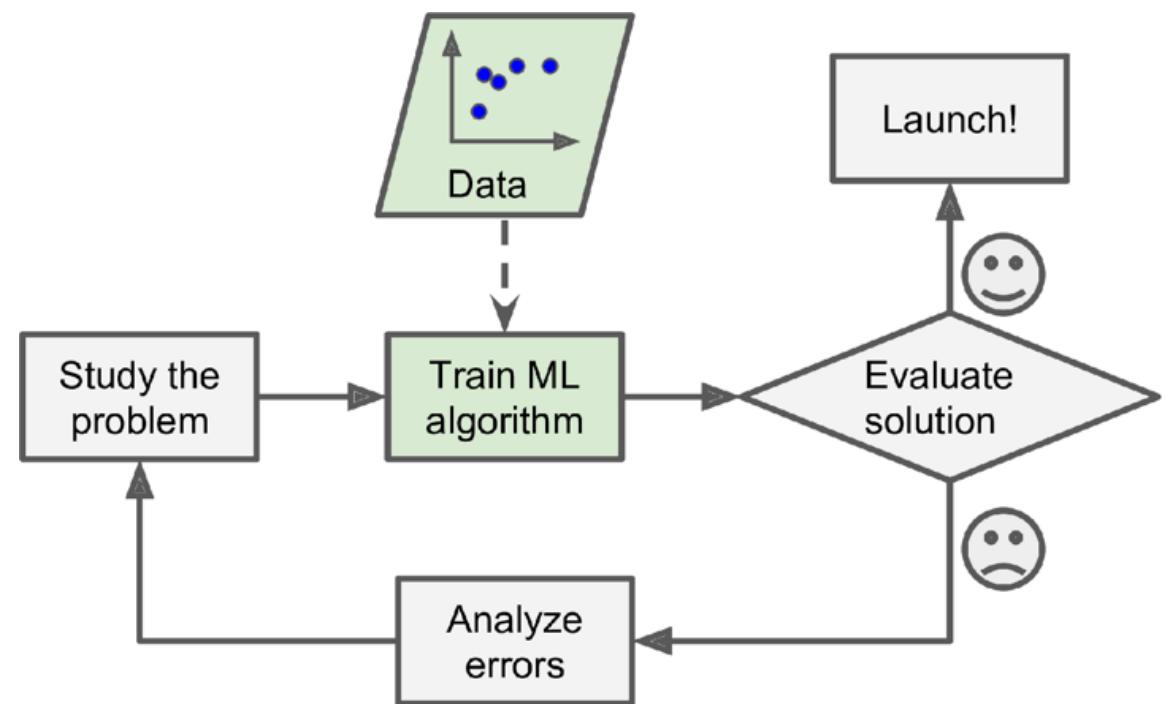
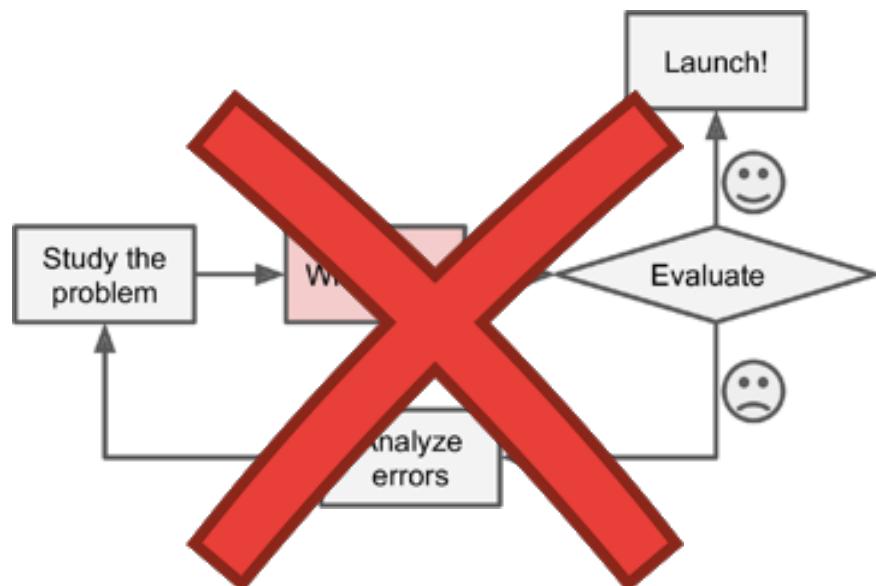


Clasificación de texto en Ing. de Software

- Clasificar requerimientos
- Clasificar decisiones de diseño (ADRs)
- Otras?



Metodología “ad-hoc” y con ML/IA



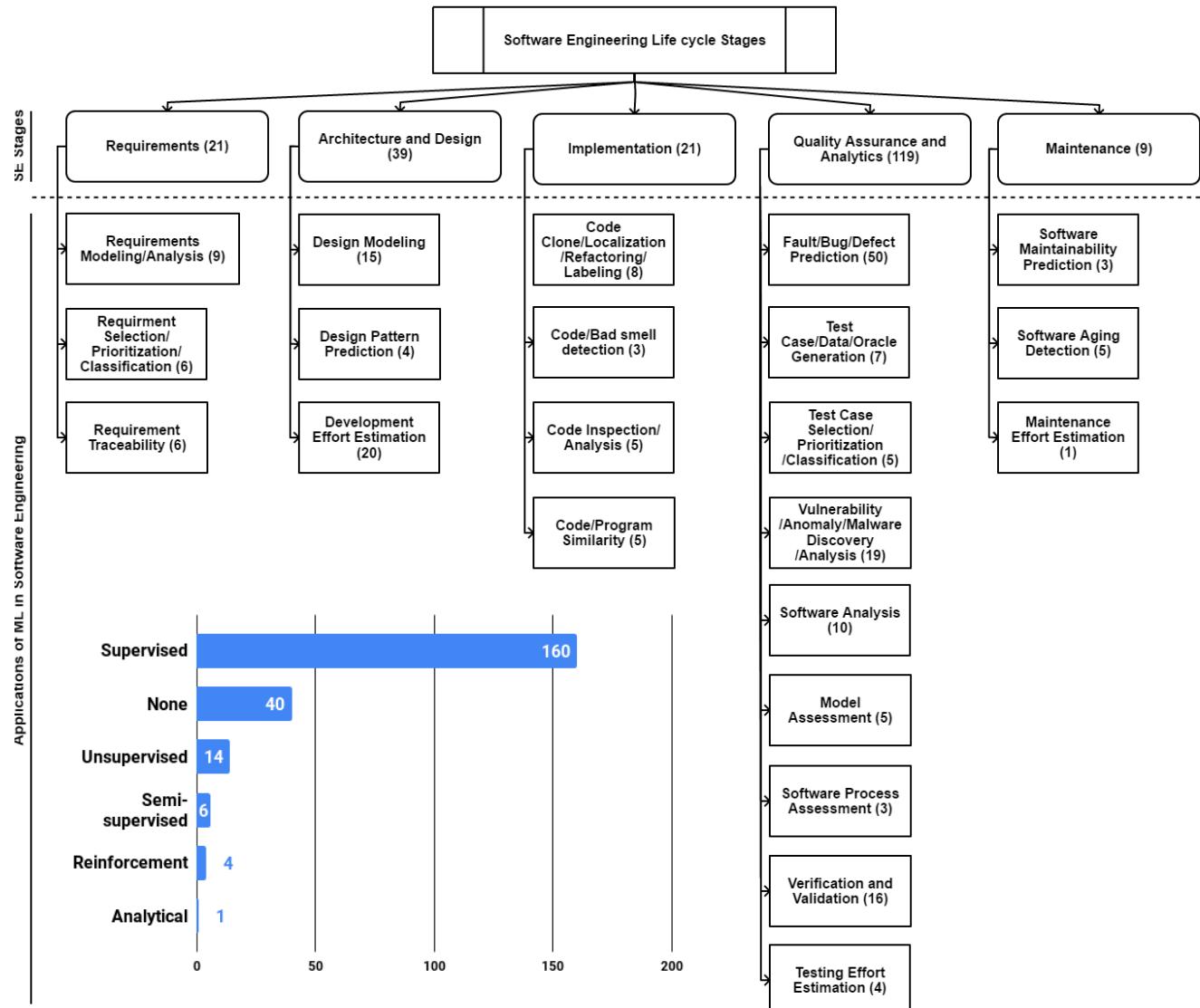
Consideraciones sobre los Datos

- Los datos tienen gran influencia en la “calidad del modelo a desarrollar
- De **dónde provienen** los datos?
- Qué **proceso** se utilizó para extraerlos?
- Hay suficientes datos?
- Son lo suficientemente **heterogéneos**?
- Existe algún tipo de sesgo potencial en los datos?
- Describen la población que quiero caracterizar?
- Hay features faltantes, o con valores inválidos?
- Necesito “combinar” distintos conjuntos de datos?



Aplicaciones a SE

- Mejora de prácticas ingenieriles para el desarrollo de software
 - Por ej., en aspecto de automatización, calidad, etc.
- Ciertamente, también aplicable a otras áreas o dominios conducidos por datos
 - Por ej., gestión

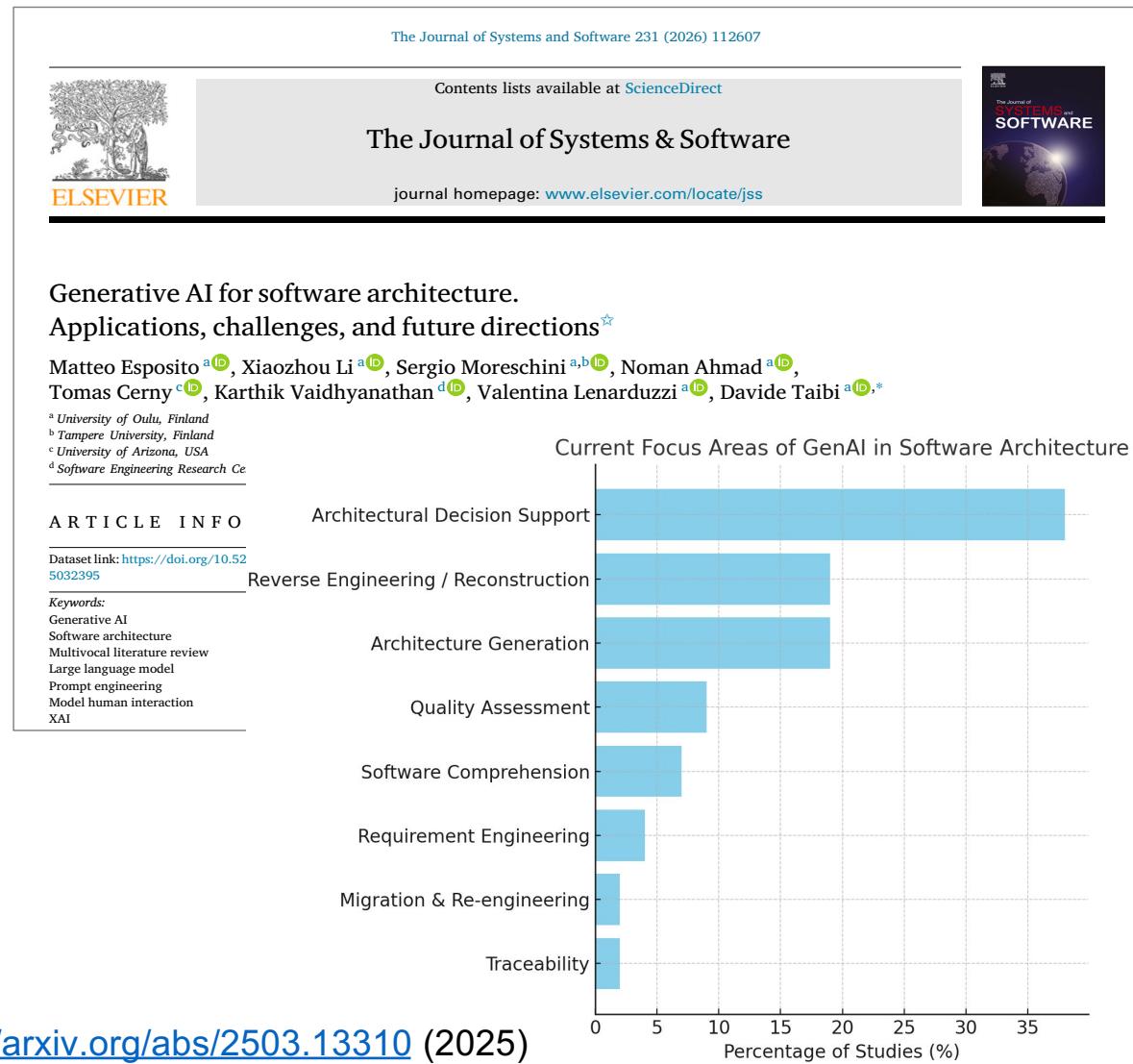


<https://arxiv.org/abs/2005.13299> (2020)

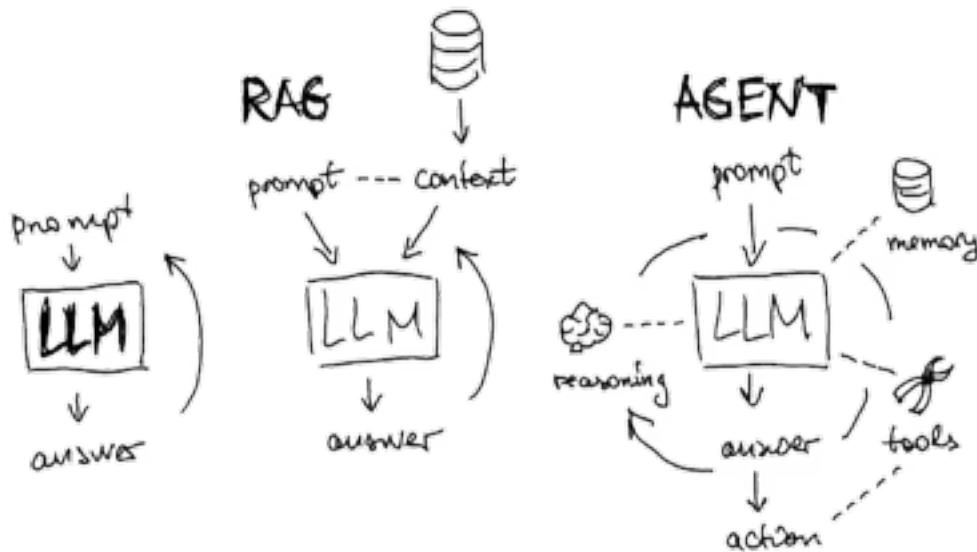
Aplicaciones a SE

- Tipos de aplicaciones:
 - *Requirements to Architecture*
 - *Architecture to Code*
 - *Requirements to Architecture to Code*
 - *Code to Architecture*
 - *Architecture to Architecture*

"LMs are most frequently applied in the Requirement-to-Architecture (40%) and Architecture-to-Code (32%) transitions, while Architecture-to-Architecture (3%) is the least explored"



Nuestra hoja de ruta



- 1

 Fundamentos de LLMs
Comprensión de modelos de lenguaje, prompting y funcionamiento interno
- 2

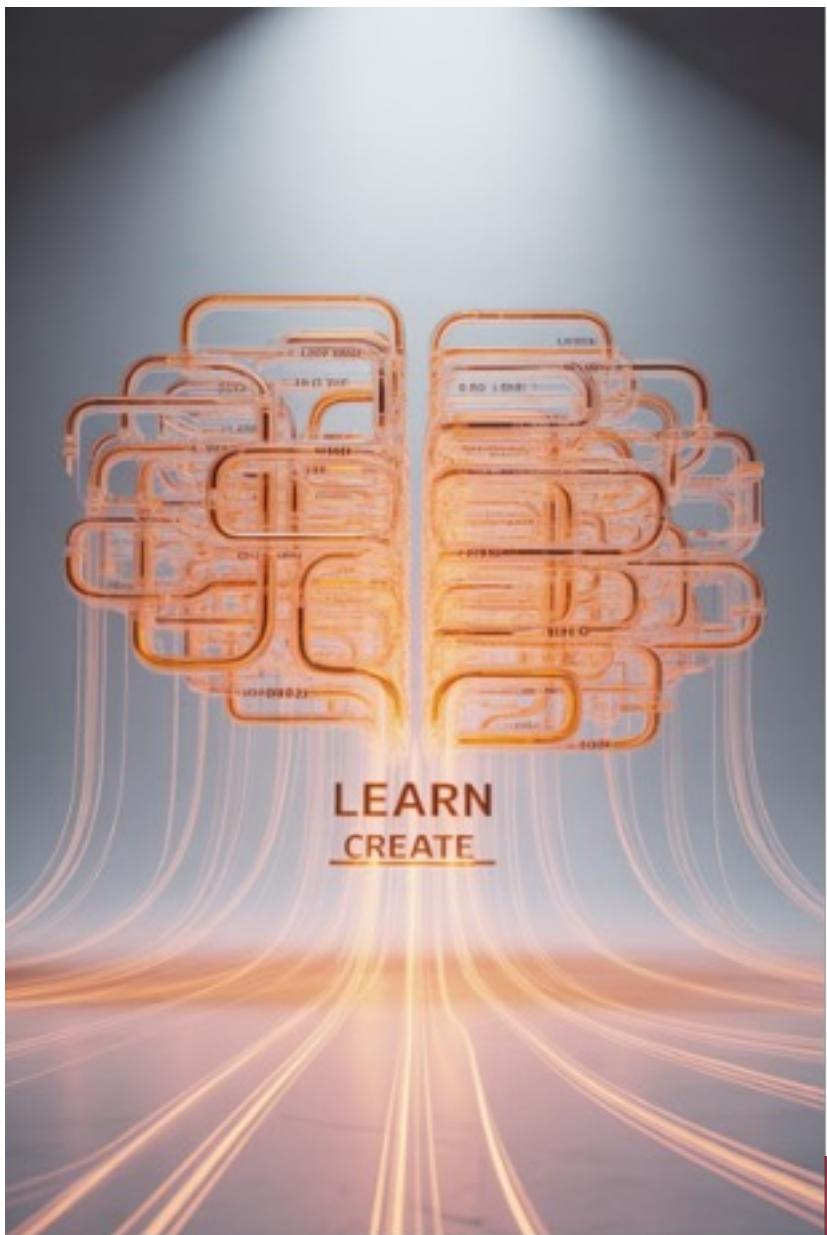
 RAG (Retrieval Augmented Generation)
Técnicas para enriquecer respuestas con info externa
- 3

 Copilots
Desarrollo de sistemas de apoyo para usuarios
- 4

 Agentes
Creación de sistemas con capacidad de decisión y acción
- 5

 Sistemas Multiagente
Integración de múltiples agentes especializados

Fundamentos de LLMs



Conceptos básicos

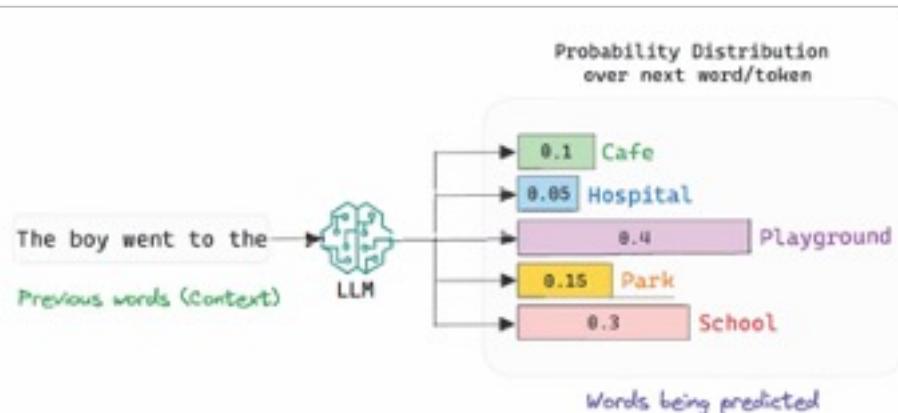
Los Modelos de Lenguaje de Gran Escala (LLMs) representan un avance fundamental en la inteligencia artificial generativa, con capacidades que transforman nuestra interacción con la tecnología.

Modelo probabilístico sofisticado

Un LLM funciona como un sistema que, a partir de una entrada, genera salidas basadas en patrones estadísticos que ha aprendido durante su entrenamiento con enormes volúmenes de texto.

Condicionamiento mediante prompting

El *prompting* es un mecanismo para dirigir las salidas del LLM, proporcionándole instrucciones y contexto específicos que influyen en cómo responde el modelo.





Ingeniería de LLMs

La ingeniería de LLMs se centra en la construcción de aplicaciones prácticas potenciadas por modelos de lenguaje avanzados, integrando estas capacidades en soluciones de software funcionales.

Objetivo principal:

Desarrollar aplicaciones de software que aprovechen las capacidades de los LLMs para resolver problemas reales, como chatbots, generadores de contenido, asistentes virtuales y herramientas de productividad.

"La ingeniería de LLMs no se trata solo de prompting efectivo, sino de diseñar sistemas robustos que integren estos modelos dentro de una arquitectura de software coherente, considerando aspectos como rendimiento, seguridad y mantenimiento."



Para profundizar en buenas prácticas de ingeniería con LLMs, recomendamos el artículo de Martin Fowler: [Engineering Practices for LLM Applications](#)

Otra visión de un LLM

Token count

27

```
<|im_start|>system<|im_sep|>You are a helpful assistant<|im_end|><|im_start|>user<|im_sep|>Hey! What is the weather in Tailandia?<|im_end|><|im_start|>assistant<|im_sep|>
```

200264, 17360, 200266, 3575, 553, 261, 10297, 29186, 200265, 200264, 1428, 200266, 25216, 0, 4614, 382, 290, 11122, 306, 353, 28387, 535, 30, 200265, 200264, 173781, 200266

Procesamiento basado en tokens

Lo que entra y sale de un LLM se procesa y monetiza en unidades llamadas **tokens**, que representan palabras o partes de palabras en el vocabulario del modelo. Los tokens son la unidad fundamental de procesamiento y determinan:

- El costo de utilizar el modelo
- La longitud máxima de texto que puede procesar
- La eficiencia del procesamiento

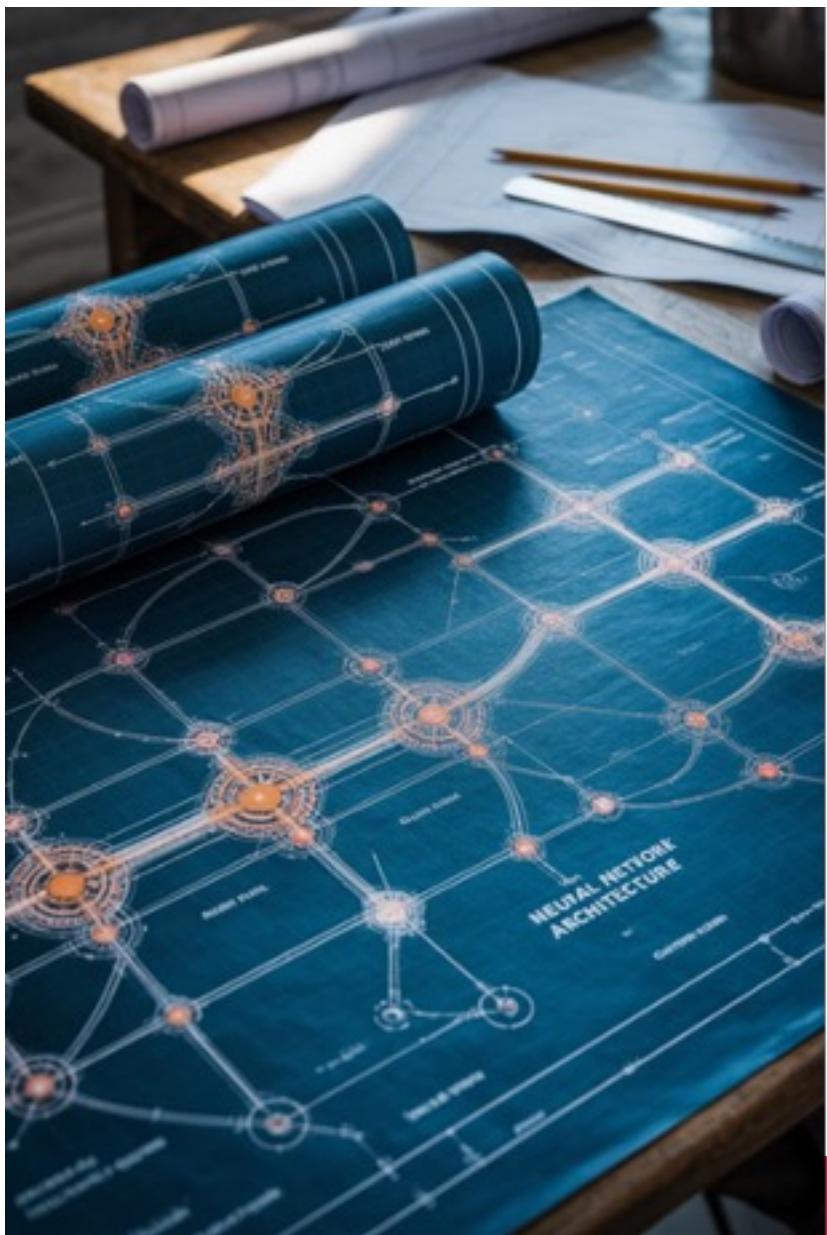
Memoria contextual

La conversación entre el LLM y el usuario mantiene una "ventana" de contexto que funciona como memoria a corto plazo.

Esta ventana de contexto:

- Tiene un límite de tokens (4K, 8K, 16K, 32K o más)
- Permite al modelo recordar partes previas de la conversación
- Influye en la coherencia de respuestas sucesivas
- Representa un desafío para conversaciones largas

Herramienta para explorar tokenización: <https://tiktoktokenizer.vercel.app/>



Algunos patrones para construir con LLMs

La creación de aplicaciones efectivas basadas en LLMs requiere diferentes enfoques según el caso de uso, cada uno con sus propias ventajas y limitaciones.



Prompting

Diseño de instrucciones para obtener respuestas específicas del modelo. Es el método más directo pero limitado por el contexto.

Fine-tuning

Adaptación del modelo a dominios específicos mediante entrenamiento adicional con datos relevantes.

RAG (Retrieval Augmented Generation)

Enriquecimiento de respuestas mediante la recuperación de información externa relevante al contexto.

Workflows

Secuencias de operaciones que combinan LLMs con otros componentes para tareas complejas.

Agentes

Sistemas autónomos que toman decisiones y ejecutan acciones basadas en objetivos definidos.

Prompting



Prompting como direccionador / contexto

El prompting actúa como un conjunto de instrucciones que orientan al LLM sobre cómo interpretar una solicitud y qué tipo de respuesta generar.

Ejemplos de diferentes contextos

Prompt 1: Tell me about: Apple

Prompt 2: Tell me about: Apple fruit

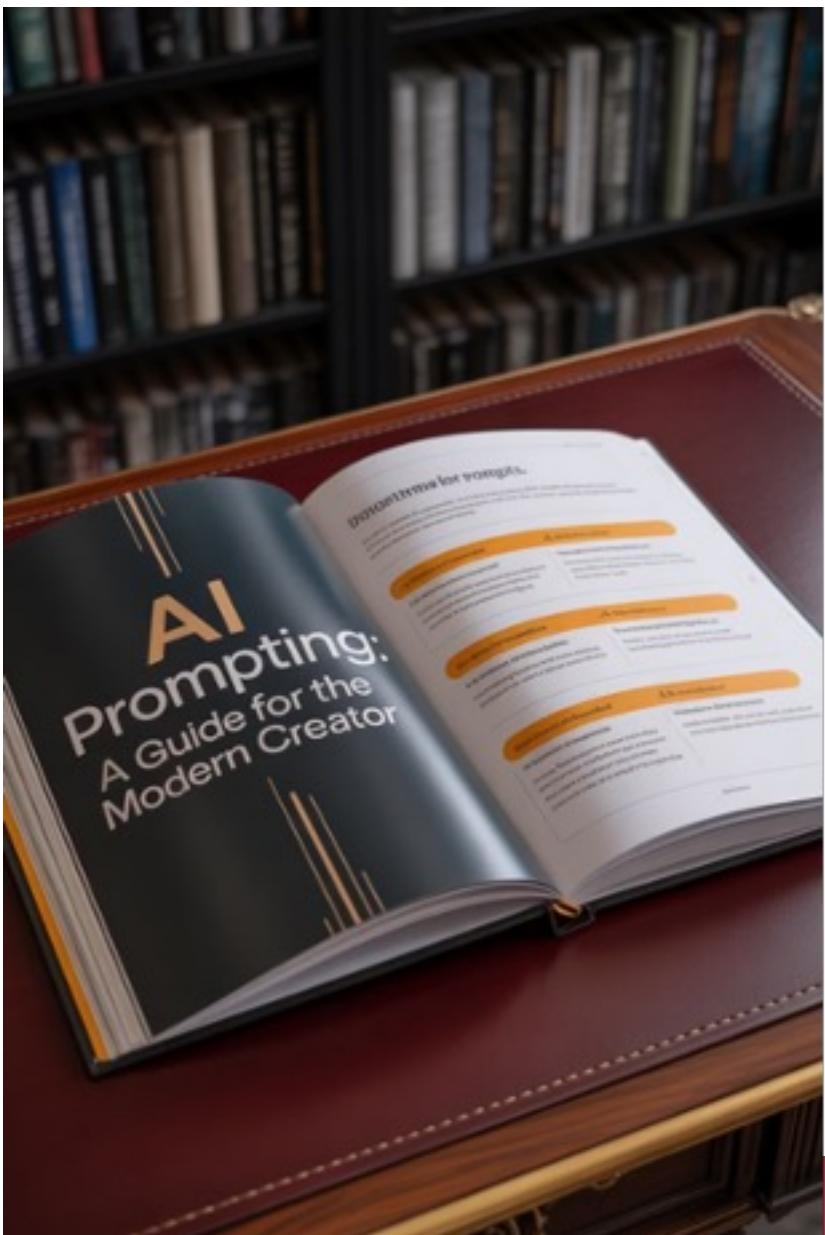
Prompt 3: Tell me about: Apple of my eye

Adaptación a diferentes roles

Prompt 4: You are a preschool teacher. Explain how attention in LLMs works.

Prompt 5: You are an NLP professor. Explain how attention in LLMs works.

Estos ejemplos demuestran cómo pequeños cambios en el prompt pueden alterar radicalmente la respuesta del modelo, permitiéndonos obtener información con el enfoque y nivel de detalle deseados.



Prompting: Guía básica

Un prompt efectivo es la base para obtener resultados precisos y útiles de un LLM. La estructura adecuada puede marcar la diferencia entre una respuesta genérica y una solución adaptada a tus necesidades.

Definición del rol

Especifica qué papel debe asumir el modelo (experto en marketing, profesor, programador...)

Planteamiento del problema

Describe claramente el contexto y el objetivo que se busca resolver

Descripción de la tarea

Detalla qué acción específica debe realizar el modelo

Instrucciones detalladas

Proporciona pautas sobre el formato, tono, extensión y otros requisitos

Ejemplos ilustrativos

Incluye casos de muestra para guiar el estilo y enfoque de la respuesta

Consideraciones adicionales

Añade restricciones, preferencias o cualquier información contextual relevante

Uso de LLMs para Clasificación y Regresión

Clasificación

La **clasificación** consiste en asignar una etiqueta predefinida a un texto o entrada, categorizando la información según criterios establecidos.

Ejemplos de aplicación:

- Clasificación de gastos por categoría (viáticos, oficina, transporte)
- Categorización de tickets de soporte por prioridad
- Análisis de sentimiento en comentarios (positivo, negativo, neutral)
- Clasificación de correos por departamento destinatario

Regresión

La **regresión** implica predecir un valor numérico basado en datos de entrada, extrapolando información para obtener estimaciones cuantitativas.

Ejemplos de aplicación:

- Predicción de precios de productos o servicios
- Estimación de tiempos de entrega
- Cálculo de presupuestos aproximados
- Predicción de ventas futuras

Enfoques de prompting para tareas analíticas



Zero-shot

El modelo realiza la tarea sin ejemplos previos, basándose únicamente en las instrucciones



One-shot

Se proporciona un único ejemplo para guiar al modelo en la tarea solicitada



Few-shots

Se incluyen varios ejemplos que establecen un patrón claro para que el modelo lo siga

Una ventaja clave de los LLMs es que pueden complementar sus clasificaciones o predicciones con explicaciones detalladas del razonamiento utilizado.



Ejemplo: Clasificación de Gastos

Un caso práctico de clasificación utilizando LLMs es la categorización automática de gastos empresariales, que simplifica la gestión contable y facilita el análisis financiero.

Prompt: Clasifica los siguientes gastos según su tipo: viáticos, oficina, transporte u otros.

Concepto	Clasificación
Cena con cliente en restaurante	Viáticos
Taxi a reunión en otra ciudad	Transporte
Compra de hojas A4	Oficina
Estacionamiento en el aeropuerto	Transporte
Pasaje aéreo a Córdoba	Transporte
Toner para impresora	Oficina
Alojamiento en hotel por viaje de negocios	Viáticos

Este tipo de clasificación puede integrarse en sistemas contables para automatizar la categorización de facturas y recibos, ahorrando tiempo y reduciendo errores humanos.



Ejemplo: Estimar costo mensual

La regresión mediante LLMs permite estimar valores numéricos basados en patrones identificados en datos de ejemplo, como en este caso de costos de auditoría.

Datos de referencia proporcionados:

Industria	Empleados	Facturación Anual	Ubicación	Costo Auditoría
Comercio	25	\$120.000.000	Buenos Aires	\$220.000
Manufactura	100	\$500.000.000	Córdoba	\$600.000
Servicios	10	\$80.000.000	Rosario	\$180.000
Servicios	30	\$250.000.000	Buenos Aires	\$320.000
Comercio	50	\$300.000.000	Mendoza	\$400.000

Estimaciones solicitadas:

Caso 1: Manufactura – 70 empleados – \$400.000.000 – Rosario **Estimación:** Aproximadamente \$520.000

Caso 2: Servicios – 15 empleados – \$150.000.000 – Córdoba **Estimación:** Aproximadamente \$240.000

El LLM no solo proporciona la estimación, sino que puede explicar los factores considerados en su cálculo, como la correlación entre tamaño de empresa, industria y ubicación.

Ejemplo multi-modal: Estimar bien mueble

Datos de referencia proporcionados:

Marca/Modelo	Año	Kilómetros	Estado	Valor estimado
Toyota Yaris	2020	45.000	Muy bueno	\$7.200.000
Ford Focus	2018	80.000	Bueno	\$5.500.000
Peugeot 208	2021	25.000	Excelente	\$8.100.000
Renault Sandero	2019	60.000	Regular	\$4.200.000

Caso a evaluar:



Vehículo: Volkswagen Gol Trend **Año:** 2020 **Kilómetros:** 50.000 **Estado:** Bueno **Ubicación:** Córdoba

Proceso de estimación:

1. Análisis de vehículos comparables en antigüedad
2. Ajuste por kilometraje (penalización por mayor uso)
3. Evaluación del estado general
4. Consideración del mercado local (Córdoba)
5. Comparación con valores de referencia del modelo

Resultado:

Estimación: \$5.800.000



Más sobre Prompting

Las técnicas avanzadas de prompting permiten obtener respuestas más precisas y razonadas de los LLMs, dirigiendo su proceso de pensamiento de manera específica.

Objetivo principal

Conseguir respuestas más efectivas o acertadas mediante instrucciones estratégicamente diseñadas que guíen el razonamiento del modelo.

Enfoques según necesidad

Zero-shot: Sin ejemplos, solo instrucciones

Few-shot: Con ejemplos que establezcan el patrón deseado

Chain-of-Thought (CoT): Guiar al modelo para que rione paso a paso

Técnicas específicas

Instrucciones como "think step by step" o "let's solve this systematically" pueden mejorar significativamente la calidad de respuestas en problemas complejos.



Nota importante: Algunas técnicas avanzadas de prompting requieren mecanismos adicionales como memoria para mantener el contexto a lo largo de múltiples intercambios, especialmente en razonamientos extensos.

Tipos de LLMs (modelos)

Los modelos de lenguaje actuales se han especializado para destacar en diferentes tipos de tareas, cada uno con fortalezas particulares.



Generación

Especializados en crear contenido como texto, código o ideas creativas. Ejemplos:

GPT-4, Claude, Llama 2.

- Creación de contenido creativo
- Redacción de documentos
- Generación de código
- Traducción y parafraseo



Razonamiento

Optimizados para resolver problemas complejos que requieren análisis lógico.

Ejemplos: DeepMind's Chinchilla, PaLM 2.

- Resolución de problemas matemáticos
- Análisis lógico
- Toma de decisiones estructuradas
- Inferencia causal



Multimodal

Capaces de procesar y generar contenido en múltiples formatos.

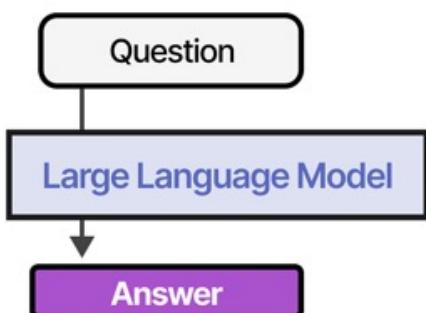
Ejemplos: GPT-4V, Gemini, Claude 3.

- Análisis de imágenes
- Comprensión de gráficos
- Interpretación de datos visuales
- Generación de contenido visual

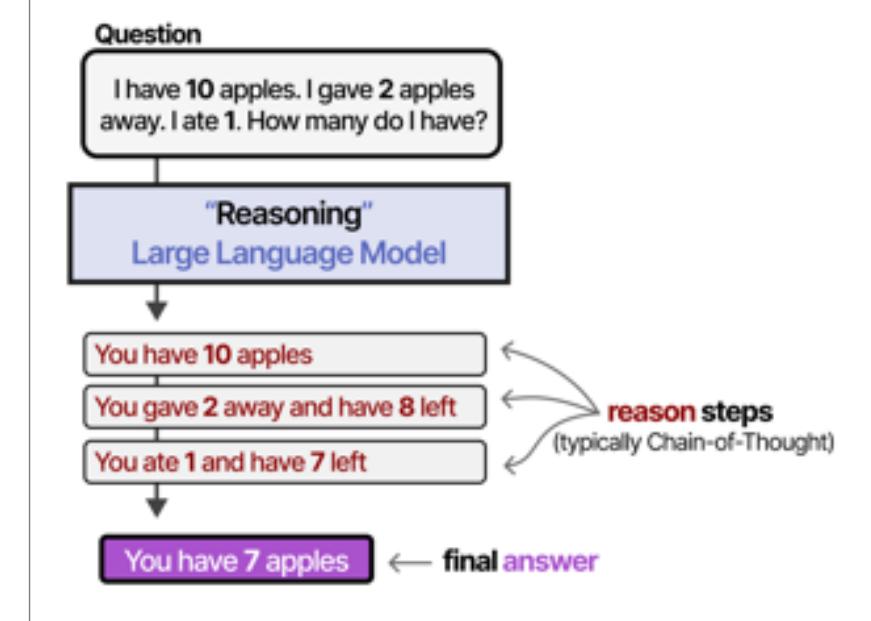
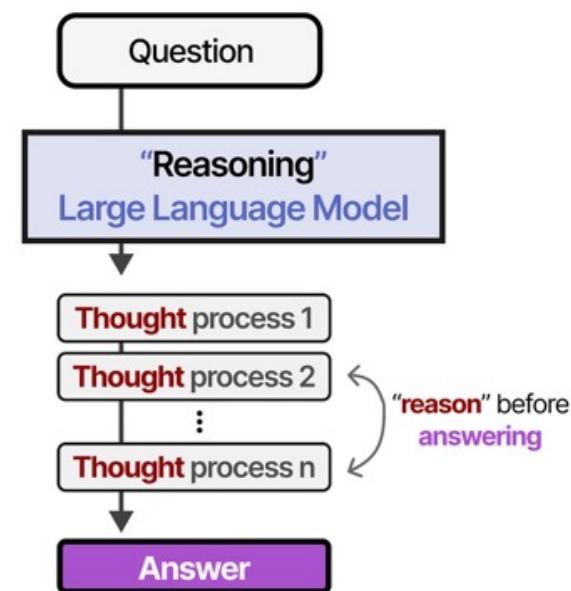
Para profundizar en los modelos de razonamiento, recomendamos esta guía visual: [A Visual Guide to Reasoning LLMs](#)

Modelos de “razonamiento”

“Regular” LLMs



“Reasoning” LLMs



<https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-langs>

Implementación

Modelos

Abstracciones para interactuar con diferentes LLMs (OpenAI, Hugging Face, Anthropic, etc.) a través de una interfaz unificada.

Callbacks

Sistema de eventos para monitorizar, registrar y depurar el funcionamiento de aplicaciones complejas.

Agentes

Sistemas que combinan LLMs con herramientas y razonamiento para resolver problemas de forma autónoma.



Prompts

Plantillas y herramientas para crear, gestionar y optimizar prompts, incluyendo técnicas avanzadas como few-shot learning.

Cadenas (Chains)

Secuencias de operaciones que combinan modelos, prompts y herramientas para realizar tareas complejas mediante pasos intermedios.

Memoria

Mecanismos para mantener el estado y el contexto a lo largo de múltiples interacciones en una conversación.

Vectorstores

Integración con bases de datos vectoriales para implementar sistemas RAG eficientes.

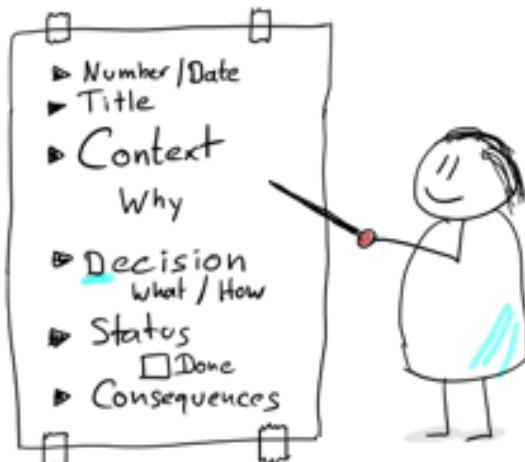
Notebooks



- *Configurar/uso de un LLM (Python)*
- *Templates de prompting*
- *Salida estructurada*

Aplicación a Ingeniería de Software

- Clasificación de ADRs
 - Taxonomía de Kruchten
 - Atributo de calidad preponderante?
- Técnica: Prompting one-shot (por categoría)



Existence
(ontocrisis)

These decisions declare that an element or artifact will exist in the design or implementation. Includes structural decisions (e.g., layers, components) and behavioral decisions (e.g., connectors, interactions). Structural decisions lead to the creation of subsystems, layers, partitions, or components. Behavioral decisions are more related to how the elements interact together to provide functionality or to satisfy some non-functional requirement (quality attribute), or connectors.

Ban/Non-existence
(anticrisis)

These decisions declare that an element will not exist in the design or implementation and often used to rule out alternatives.

Property
(diacrisis)

These decisions state a general, enduring quality or constraint of the system. These are often cross-cutting concerns or design rules (positive) or constraints (negative).

Executive
(pericrisis)

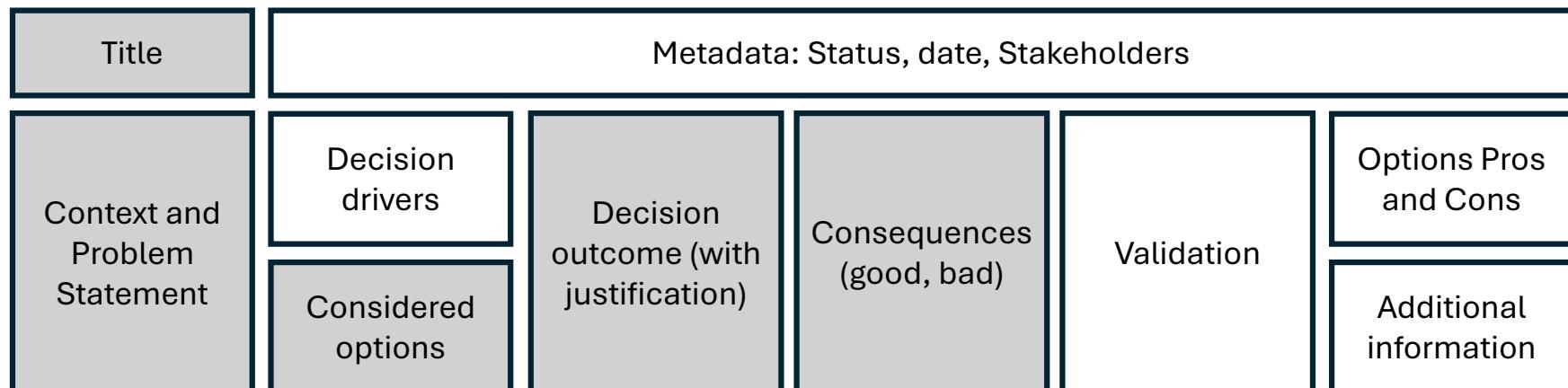
These decisions refer to decisions that do not relate directly to the design elements or their qualities but are driven more by the business environment (financial), and affect the development process (methodological), the people (education and training), the organization, and to a large extend the choices of technologies and tools.

<https://github.com/joelparkerhenderson/architecture-decision-record/tree/main>



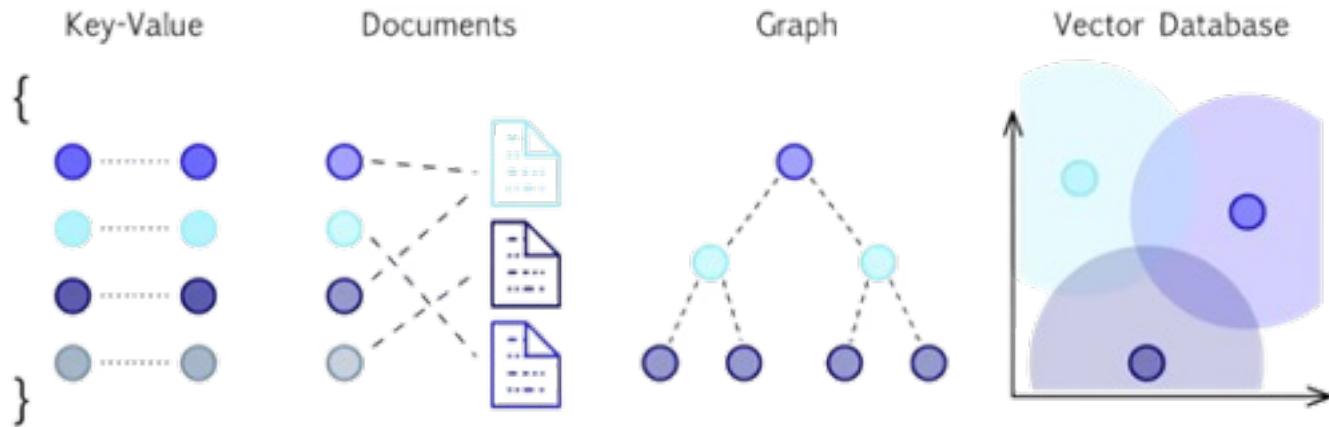
Aplicación a Ingeniería de Software

- Actividad: Escribir un prompt que permita chequear la calidad de una ADR, en el sentido de que tenga las secciones necesarias (obligatorias, opcionales) de MADR, y si es posible, los contenidos esperados por sección
- La evaluación debiera retornar un score de 1-100% y una explicación



<https://adr.github.io/madr/>

Búsqueda Semántica



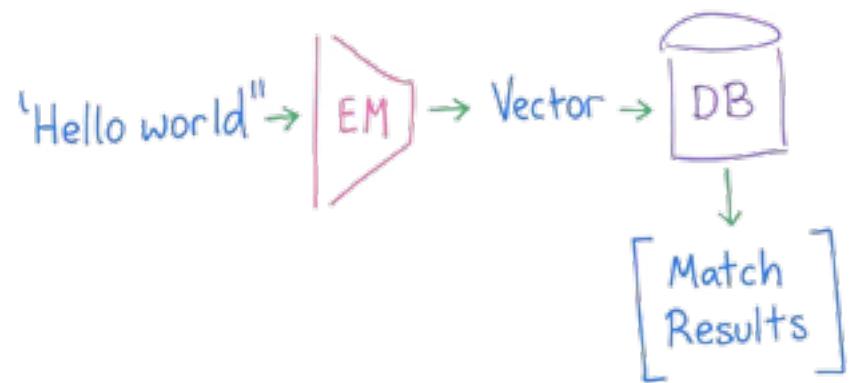
La búsqueda semántica va más allá de la coincidencia de palabras clave, utilizando el significado contextual de las consultas para devolver resultados relevantes incluso cuando no hay coincidencias exactas de términos.

- ⓘ A diferencia de la búsqueda tradicional que se basa en coincidencias léxicas, la búsqueda semántica entiende la intención y el contexto de la consulta del usuario.

Arquitectura de un sistema de búsqueda semántica

Los sistemas de búsqueda semántica combinan técnicas de procesamiento de lenguaje natural, embeddings vectoriales y algoritmos de recuperación para proporcionar resultados relevantes basados en el significado.

El proceso típicamente incluye la conversión de documentos y consultas a vectores, el almacenamiento eficiente en bases de datos vectoriales y la recuperación basada en similitud.



Embeddings: Representaciones vectoriales

Concepto fundamental para agentes de IA

Los modelos de embedding son algoritmos entrenados para encapsular información en representaciones densas dentro de un espacio multidimensional.



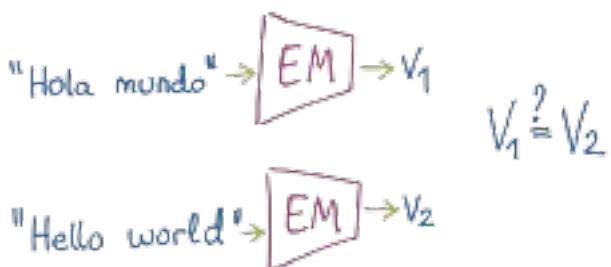
- ① Los embeddings permiten a las máquinas entender relaciones semánticas entre palabras, frases, imágenes y otros tipos de datos.

Estas representaciones vectoriales capturan el significado y contexto, permitiendo:

- Búsquedas semánticas
- Recomendaciones
- Clasificación de contenido
- Agrupación de datos similares
- Detección de anomalías

Matemática intuitiva detrás de los vectores

La similitud coseno mide el coseno del ángulo entre dos vectores, proporcionando un valor entre -1 y 1. Cuanto más cercano a 1, más similares son los vectores en dirección, independientemente de su magnitud.



$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

Aplicaciones de la búsqueda semántica

Motores de búsqueda avanzados

Mejora de resultados basados en la intención del usuario y no solo en palabras clave

Sistemas RAG

Recuperación aumentada de generación para LLMs, proporcionando contexto relevante desde fuentes externas

Bases de conocimiento

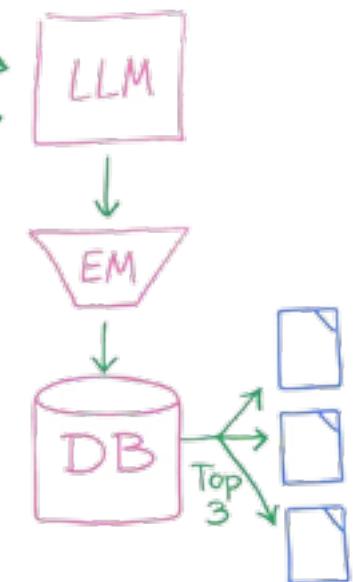
Organización inteligente de documentación técnica y recursos empresariales

Chatbots y asistentes

Mejora de la comprensión contextual en sistemas conversacionales

"Optimiza la siguiente pregunta para:...."

"Como cambio el alias de mi cuenta?"



Articulación de búsqueda semántica con LLM

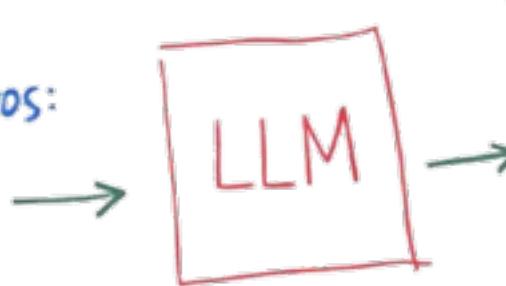
"Dado la siguiente Pregunta:



y los siguientes fragmentos:



Respuesta:



"Debes ir a:
Cuento, luego..."

Pausa



RAG (Retrieval Augmented Generation)



Retrieval Augmented Generation (RAG)

RAG es una técnica que permite a los modelos de lenguaje acceder a información externa y específica, superando las limitaciones de su conocimiento entrenado.

"Habla con tus documentos"

Ingestión de documentos

Procesamiento de documentos específicos de un tema o dominio particular para su posterior consulta

Conversión en contexto

Transformación de los documentos en información estructurada que puede ser recuperada eficientemente

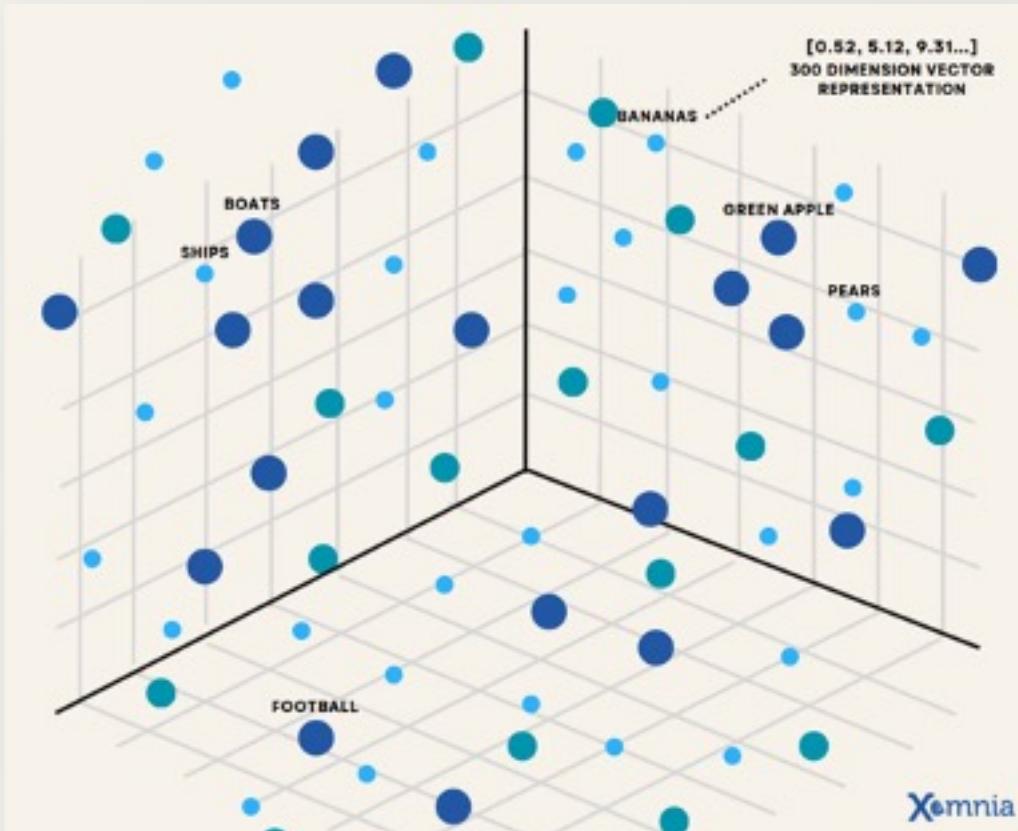
Recuperación contextual

Selección de fragmentos relevantes en función de la consulta del usuario

Generación fundamentada

Uso de los fragmentos recuperados como contexto adicional para que el LLM genere respuestas precisas

El efecto principal de RAG es que el LLM se **"enfoca"** (grounding) en los documentos proporcionados para generar respuestas más precisas y relevantes, expandiendo y especializando su conocimiento de manera efectiva.



DB Vectoriales

Funcionamiento básico

Convierten contenido a vectores numéricos multidimensionales donde la cercanía en el espacio vectorial representa similitud semántica

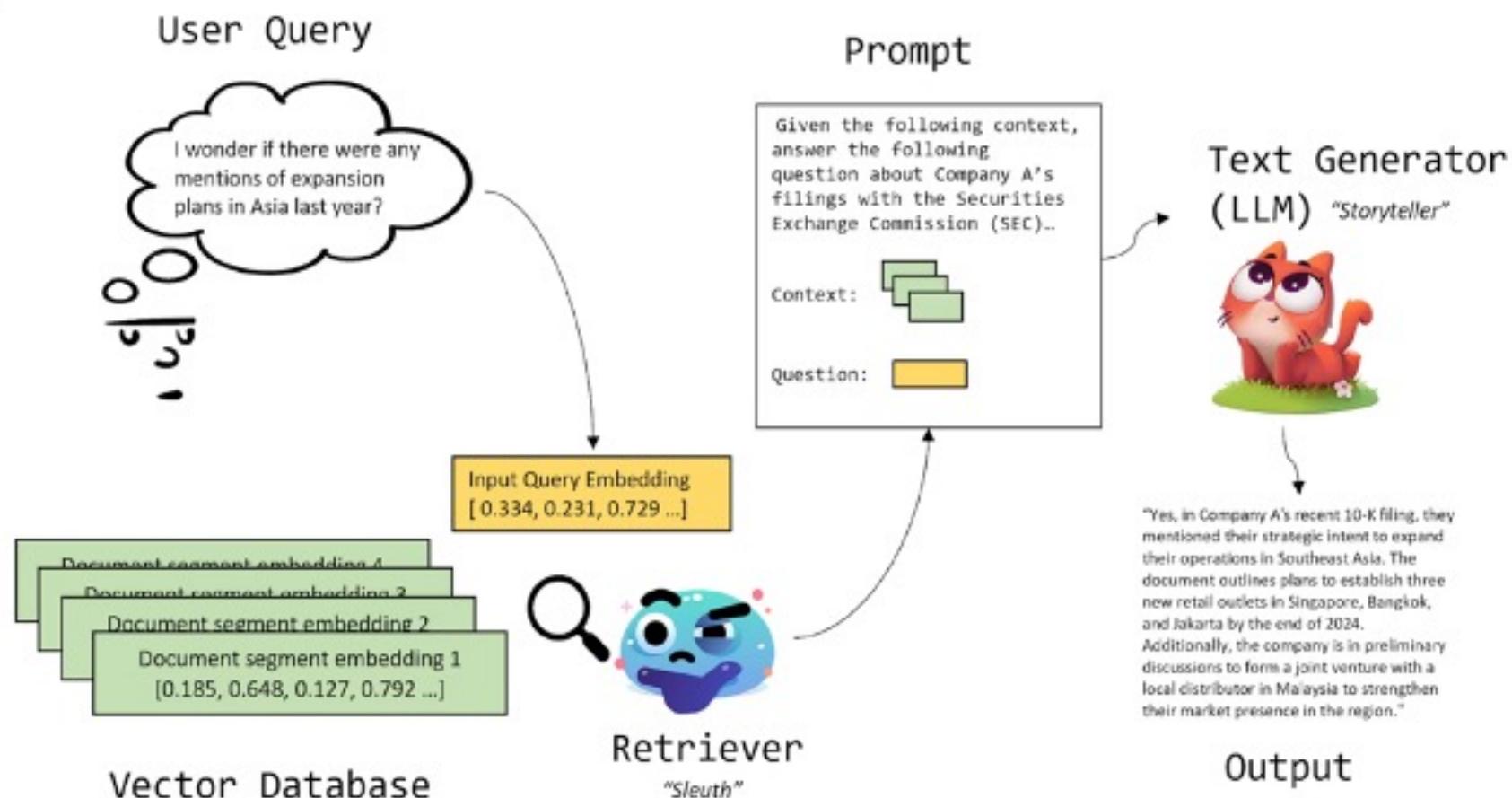
Ventajas clave

- Búsqueda por similitud conceptual
- Comprensión de sinónimos y relaciones
- Recuperación eficiente en grandes volúmenes
- Soporte para consultas multimodales

Aplicaciones en RAG

Permiten recuperar rápidamente los fragmentos más relevantes para una consulta, incluso cuando no hay coincidencia literal de términos

Funcionamiento de un RAG



Esquema de LLM con RAG

El flujo de información en un sistema RAG sigue un proceso estructurado que enriquece las consultas del usuario con conocimiento externo relevante.



La inyección de contexto externo hace que la respuesta sea más factual y basada en la información específica de la base de datos, reduciendo las alucinaciones y aumentando la precisión.

Visualización de un RAG

Las herramientas de visualización como RAGxplorer permiten comprender mejor cómo funcionan internamente los sistemas RAG, facilitando su optimización y depuración.

Características principales de RAGxplorer

- Visualización interactiva de espacios vectoriales en 3D
- Exploración de relaciones entre documentos y consultas
- Identificación de clusters temáticos en la base de conocimiento
- Análisis de la efectividad de diferentes estrategias de chunking
- Evaluación visual de la calidad de recuperación

Esta herramienta de código abierto es especialmente útil para desarrolladores de sistemas RAG que necesitan ajustar y optimizar sus implementaciones.

<https://github.com/gabrielchua/RAGxplorer>

Visualise which chunks are most relevant to your query.

Enter your query:

What are the top revenue drivers for Microsoft?



Estrategias de Chunking e Indexado

El rendimiento de un sistema RAG depende en gran medida de cómo se dividen e indexan los documentos originales, un proceso que requiere equilibrar diversos factores.

El desafío del chunking

El problema central consiste en encontrar el particionamiento "correcto" que:

- Preserve el contexto y la estructura lógica del documento
- Facilite la búsqueda eficiente de fragmentos relevantes
- Aproveche la información semántica para mejorar la recuperación

Chunking por estructura lingüística

División basada en oraciones, párrafos o secciones naturales del texto

Particionamiento recursivo

División jerárquica que mantiene la relación entre fragmentos de diferentes niveles

Enriquecimiento por contexto

Inclusión de metadatos o fragmentos de contexto circundante para mejorar la comprensión

Chunking por modalidad

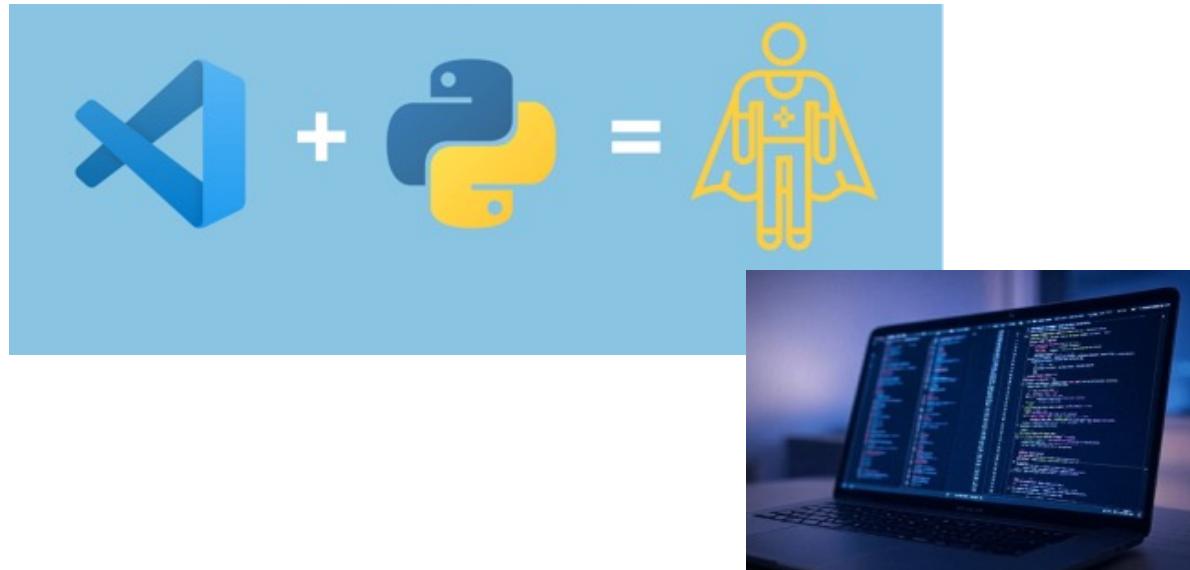
Tratamiento específico según el tipo de contenido: texto, tablas, imágenes, etc.

Chunking semántico

División basada en unidades de significado o temas, independientemente de la estructura formal



Notebooks



- *RAG simple con base vectorial*

RAGs en Ing. de Software

- Generación de documentación (por ej., ADRs)
- Q&A sobre arquitecturas de referencia (por ej., AWS)
- Otros?

<https://awslabs.github.io/mcp/servers/aws-knowledge-mcp-server>



Técnicas de RAG avanzadas

Más allá del RAG básico, existen numerosas técnicas avanzadas que mejoran la calidad, precisión y eficiencia de la recuperación y generación de información.

Mejoras en la recuperación

Reranking

Reordenamiento de resultados mediante un segundo modelo más preciso

Hybrid search

Combinación de búsqueda semántica y por palabras clave

Query expansion

Generación de múltiples versiones de la consulta para ampliar la cobertura

Mejoras en la generación

Self-query

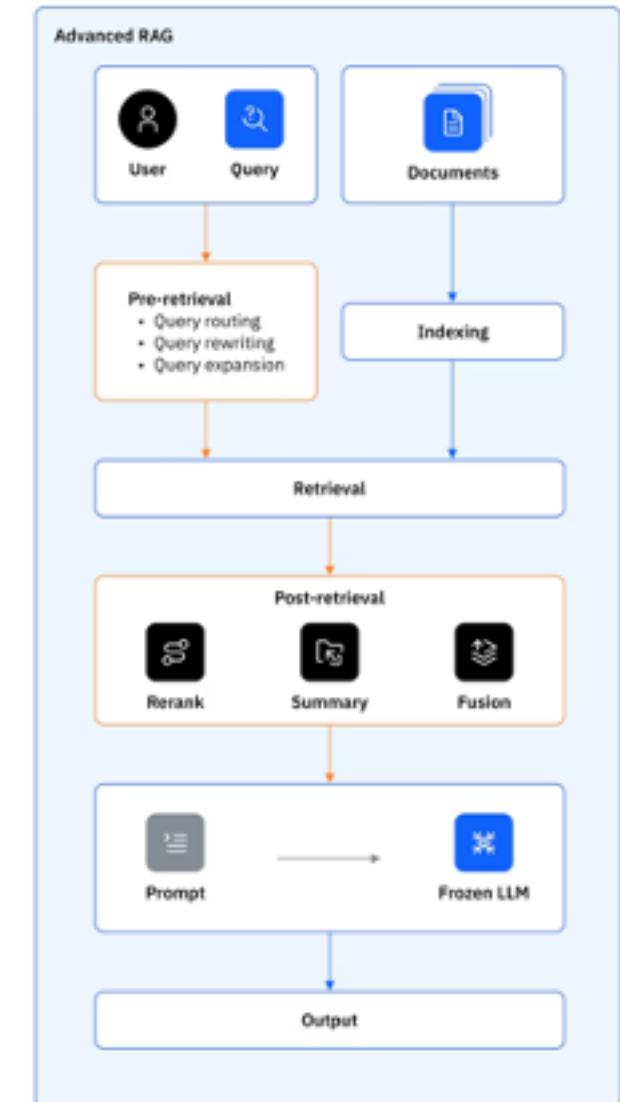
El LLM reformula la consulta original para optimizar la recuperación

Generación con citaciones

Inclusión de referencias a las fuentes de información utilizadas

Post-procesamiento

Verificación de la respuesta generada contra las fuentes recuperadas



Más información: <https://www.ibm.com/think/topics/rag-techniques>

Técnicas de RAG avanzadas

Arquitecturas avanzadas

RAG recursivo

Múltiples ciclos de recuperación y generación para refinar progresivamente la respuesta

Chunking adaptativo

Fragmentación dinámica basada en la estructura semántica del documento y el tipo de consulta

RAG multi-vectorial

Uso de diferentes representaciones vectoriales para el mismo contenido, optimizadas para distintos tipos de consultas

Knowledge graphs

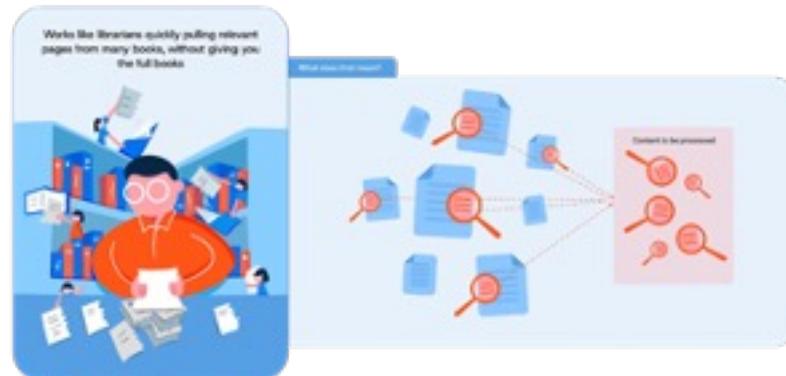
Integración de grafos de conocimiento para representar relaciones entre entidades y conceptos

¿Y si no usara RAG sino la ventana de contexto?

RAG (Retrieval Augmented Generation)



CAG (Context-aware Generation)



Características:

- Recuperación dinámica basada en la consulta
- Selección precisa de información relevante
- Menos dependiente del tamaño de contexto
- Mayor eficiencia para bases de conocimiento grandes
- Requiere infraestructura de búsqueda

Más información: <https://pair.gov.sg/articles/beyond-rag>

Características:

- Carga directa de documentos completos en el contexto
- Aprovecha ventanas de contexto extensas (32K, 100K+ tokens)
- Implementación más simple
- Mejor comprensión del documento completo
- Limitado por el tamaño máximo de contexto

Aplicación a Ingeniería de Software

- *Archmind*
 - Un conjunto de chatbots para practicar toma de decisiones de arquitectura
- Técnica: RAG con prompts especializados y bases de conocimiento de patterns



<https://archmindv2-test.streamlit.app/>

<https://github.com/tommantonela/archmind>

The screenshot shows a user interface for a software engineering tool. At the top, there's a navigation bar with tabs: Q&A, Pattern Suggestion (which is active), Pattern Assessment, ADR Generator, and Consistency Check. To the right of the navigation is a stylized icon of a robot head with speech bubbles and gears.

Below the navigation is a section titled "System context:" containing a detailed text about a food company's migration from a monolithic system to microservices. The text describes the current state with two SQL databases (Customers, Orders) and the planned transition using HTTP/REST protocols through a Gateway component. It mentions critical modules for Customers, Delivery & Routing, and Payments, as well as non-critical modules for Orders, Statistics, and Incidents. The new architecture will require an OrderManager component to serve as an intermediary between key functionalities.

Underneath this is a "Enter requirement:" input field with a right-pointing arrow icon.

The main content area contains a message from the AI bot:

The system must provide a module to optimize delivery and order routing depending on the delay. Two optimization algorithms that assign the best route should be implemented.

For your requirement, I have the following patterns:

- Strategy Pattern
- Factory Method Pattern
- Chain of Responsibility Pattern

Please, edit the list of patterns to be compared and ranked and press ctrl + enter.

At the bottom, there's a "Patterns to select:" section with a list: Strategy Pattern, Factory Method Pattern, Chain of Responsibility Pattern. There's also a small circular icon with a question mark.

Aplicación a Ingeniería de Software

- Actividad: Realizar un RAG que permita contestar preguntas sobre una fuente de conocimiento específica, relacionada con Ingeniería de Software



Copilots

Copilots

Los asistentes o copilots representan un nivel intermedio entre un simple chatbot y un agente autónomo, ofreciendo apoyo contextualizado sin tomar decisiones por el usuario.

Características principales

- Memoria persistente de conversaciones previas
- Implementación frecuente sobre arquitectura RAG
- Capacidad para entender el contexto del usuario
- Interfaz conversacional natural

Límites de autonomía

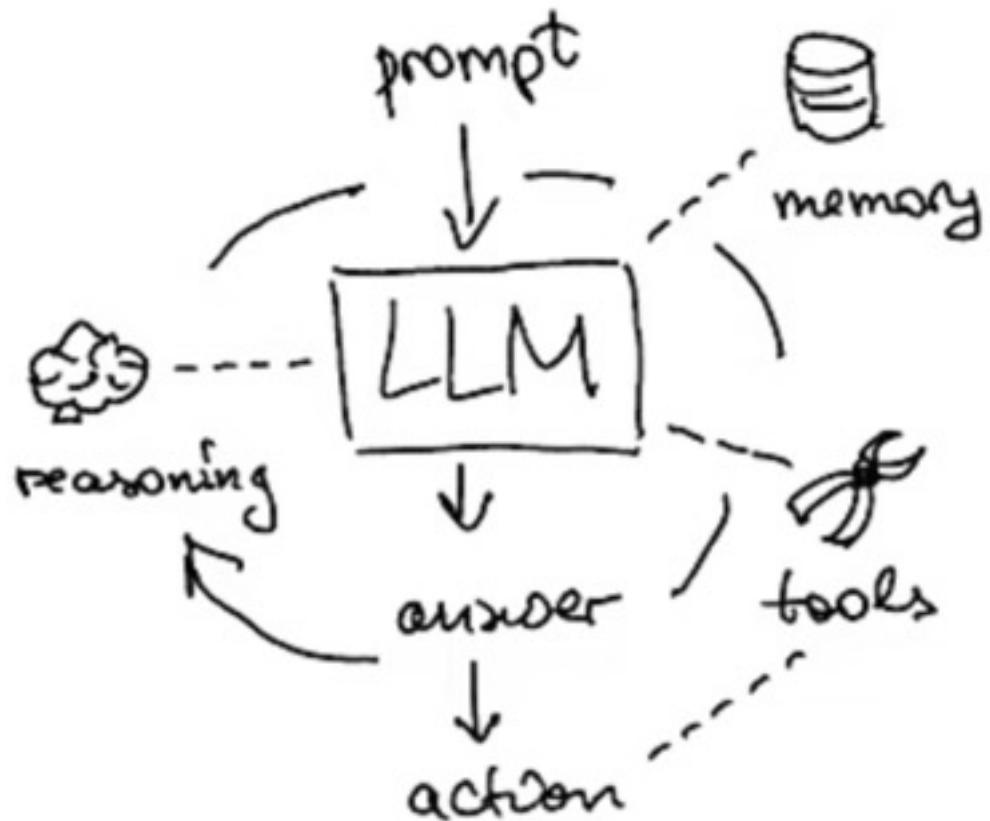
A diferencia de los agentes, los asistentes no tienen autonomía para decidir o ejecutar acciones, sino que se limitan a sugerir, recomendar o generar información que el usuario puede aceptar, rechazar o modificar.

Aplicaciones comunes

- Asistentes de programación (GitHub Copilot)
- Ayudantes de redacción y edición
- Soporte para análisis de datos
- Asistentes de productividad integrados en aplicaciones

Agentes

AGENT



Herramientas para
construir Agentes de IA

Frameworks



LangChain

LangChain

LangChain is a framework for building LLM-powered applications.



LlamaIndex

LlamaIndex

Build agentic workflows to extract information, synthesize insights, and take actions over the most complex enterprise documents.



Agno

Agno is a python framework for building multi-agent systems with shared memory, knowledge and reasoning.



Semantic Kernel

Semantic Kernel is a lightweight, open-source development kit that lets you easily build AI agents and integrate the latest AI models into your C#, Python, or Java codebase. It serves as an efficient middleware that enables rapid delivery of enterprise-grade solutions.

Frameworks



HayStack

Haystack is an open-source AI orchestration framework built by deepset that empowers Python developers to build real-world, compound, agentic LLM applications.



CrewAI

CrewAI is an open source multiagent orchestration framework.



AutoGen

AutoGen is a framework for creating multi-agent AI applications that can act autonomously or work alongside humans.



Azure AI Foundry

This foundation combines production-grade infrastructure with friendly interfaces, enabling developers to focus on building applications rather than managing infrastructure.

Alternativa: Flowise

Flowise es una plataforma **low-code** que democratiza el desarrollo de aplicaciones basadas en IA, permitiendo a usuarios sin experiencia en programación crear sistemas sofisticados.

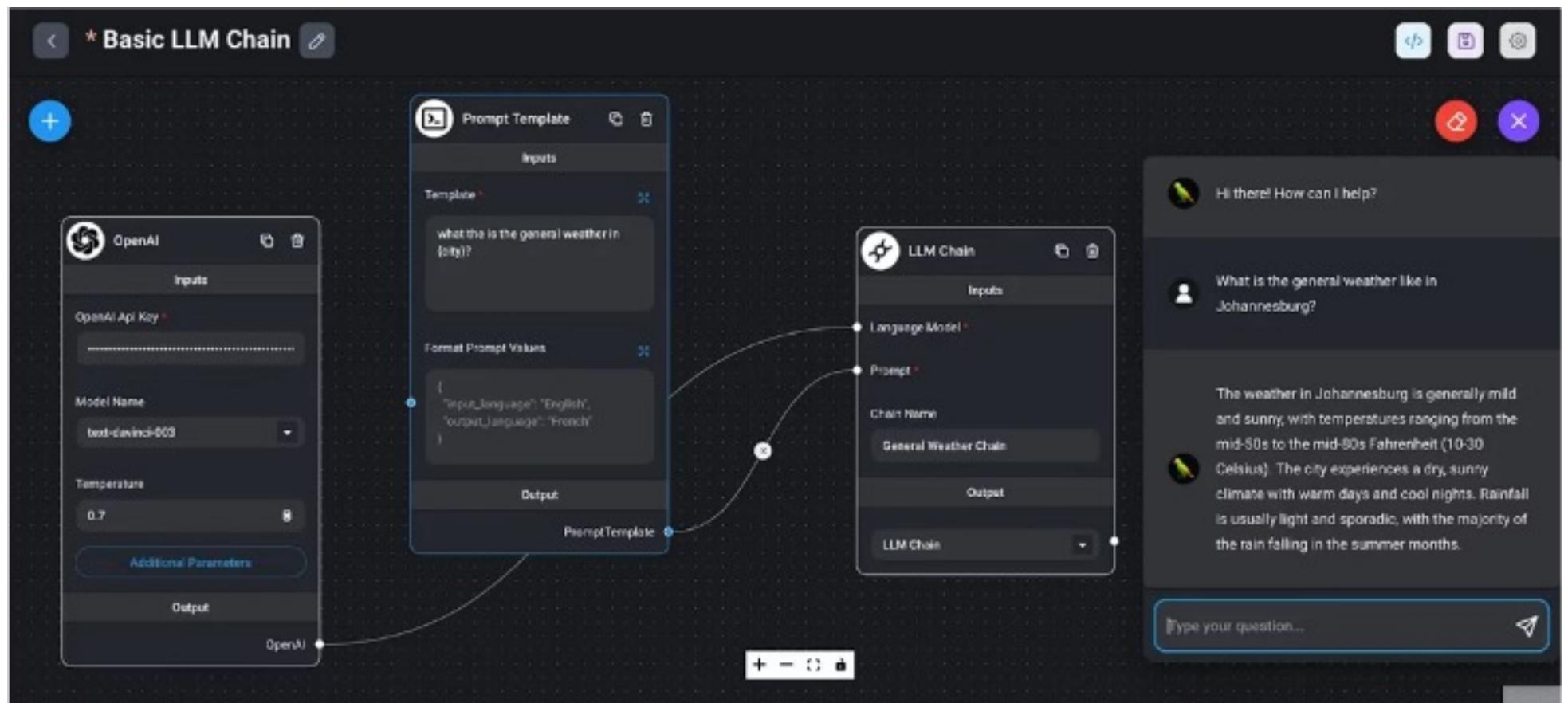
Características principales

- Interfaz visual de "arrastrar y soltar" para diseñar flujos
- Configuración intuitiva de bloques y conexiones
- Integración con múltiples frameworks y herramientas de IA
- Compatibilidad con diversas bases de datos y fuentes de información
- Plantillas prediseñadas para casos de uso comunes
- Despliegue simplificado en la nube

Ventajas para desarrolladores

- Prototipado rápido de soluciones de IA
- Reducción significativa del tiempo de desarrollo
- Visualización clara de flujos de trabajo complejos
- Fácil experimentación con diferentes enfoques
- Depuración visual de problemas
- Colaboración mejorada con equipos multidisciplinarios

Alternativa: Flowise



Alternativa: Flowise



Chatbots personalizados

Asistentes conversacionales adaptados a dominios específicos



Automatizaciones

Flujos de trabajo que integran IA para tareas repetitivas



Análisis documental

Sistemas para extraer y procesar información de documentos

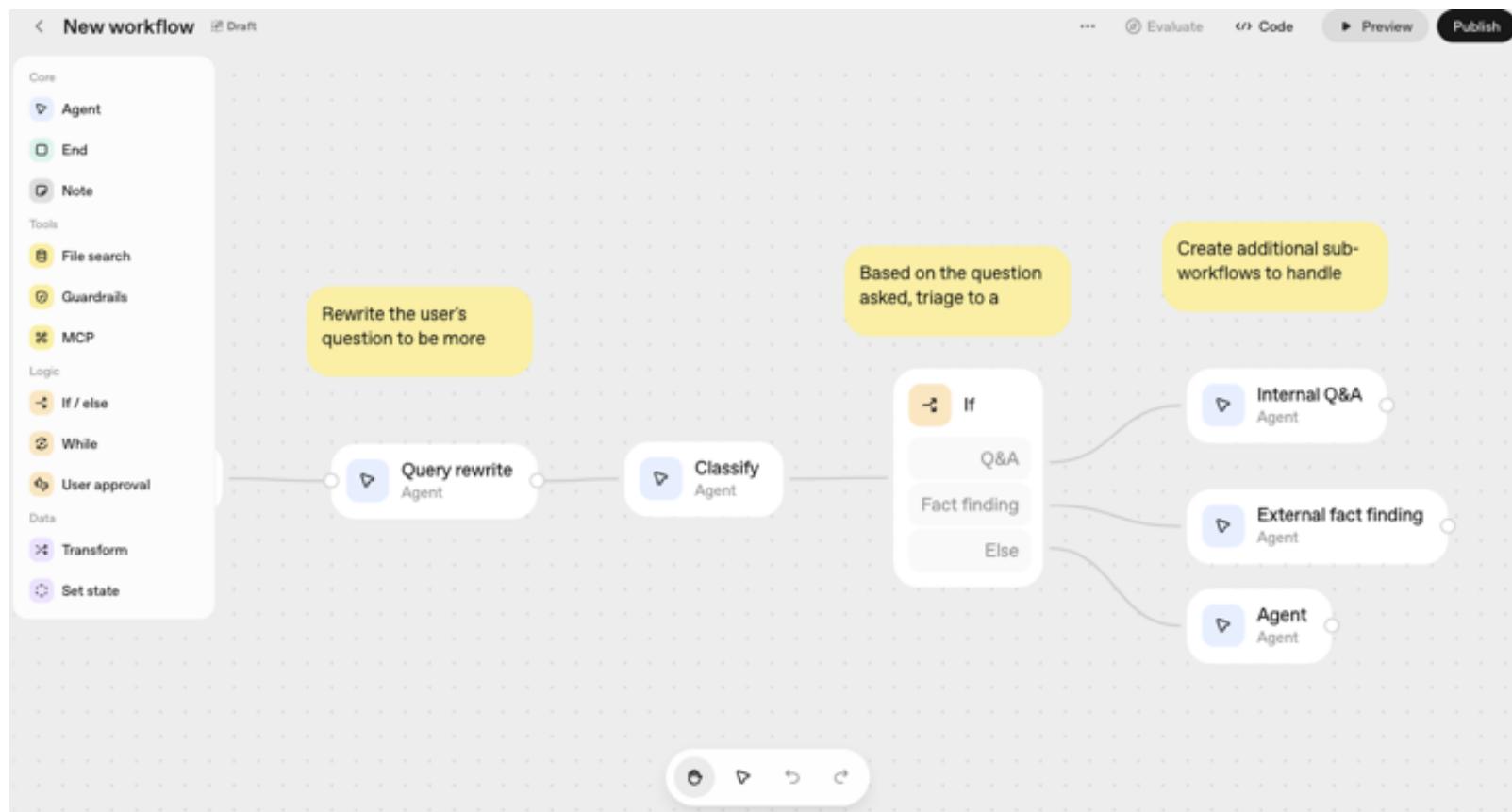


Sistemas RAG

Soluciones de búsqueda y respuesta basadas en conocimiento específico

Curso recomendado: [Curso de Flowise por Gabriel Merlo](#)

Otra alternativa: AgentKit / AgentBuilder (OpenAI)

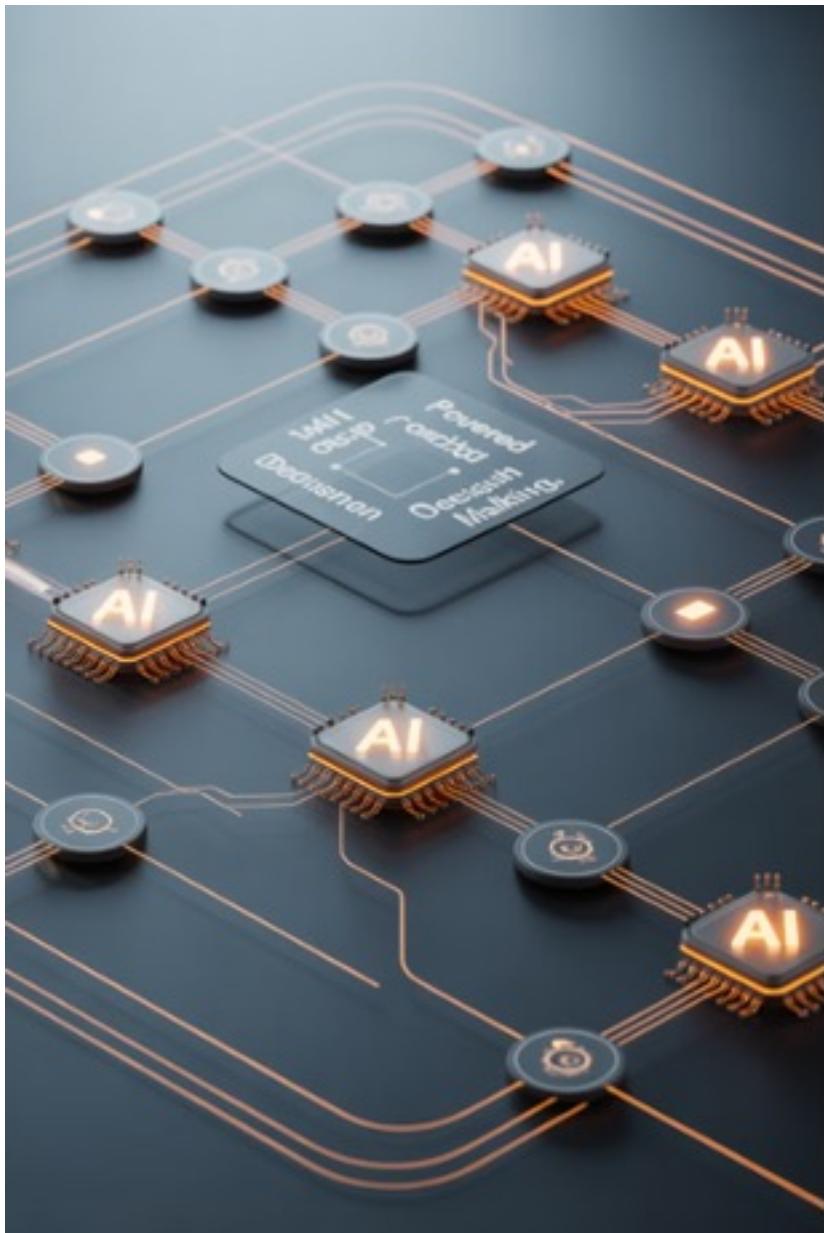


Aplicación a Ingeniería de Software

- Actividad (opcional): Tanto la actividad de prompting para clasificación como la actividad de RAG podrían realizarse con un editor no/low-code



Sistemas Multiagente



Volviendo al LLM: Un punto de quiebre

A medida que los proyectos de IA se vuelven más complejos, los enfoques basados únicamente en prompts empiezan a mostrar limitaciones significativas.

Limitaciones del super-prompt

Un prompt único, por muy sofisticado que sea, no siempre resulta efectivo para resolver tareas complejas que requieren múltiples pasos o tipos de razonamiento diferentes.

Complejidad de los flujos

Al desarrollar aplicaciones con múltiples invocaciones a LLMs, el proceso se vuelve cada vez más difícil de implementar, evaluar y ajustar de manera coherente.

Desafíos de orquestación

La coordinación de decisiones secuenciales en flujos de trabajo complejos (por ejemplo, selección de candidatos para un perfil laboral) introduce nuevas capas de complejidad.

La transición de prompts individuales a sistemas modularizados representa un punto de inflexión en el desarrollo de aplicaciones de IA, similar al paso de scripts monolíticos a arquitecturas de software bien estructuradas.

Modularización: Divide, Conquista y Autonomía

La evolución natural de la modularización es el desarrollo de sistemas multiagente, donde cada componente



Tiene razonamiento propio

Cada agente aplica capacidades específicas de razonamiento a su dominio de especialización



Se enfoca en objetivos concretos

Los agentes se concentran en resolver aspectos específicos del problema global



Se comunica con otros agentes

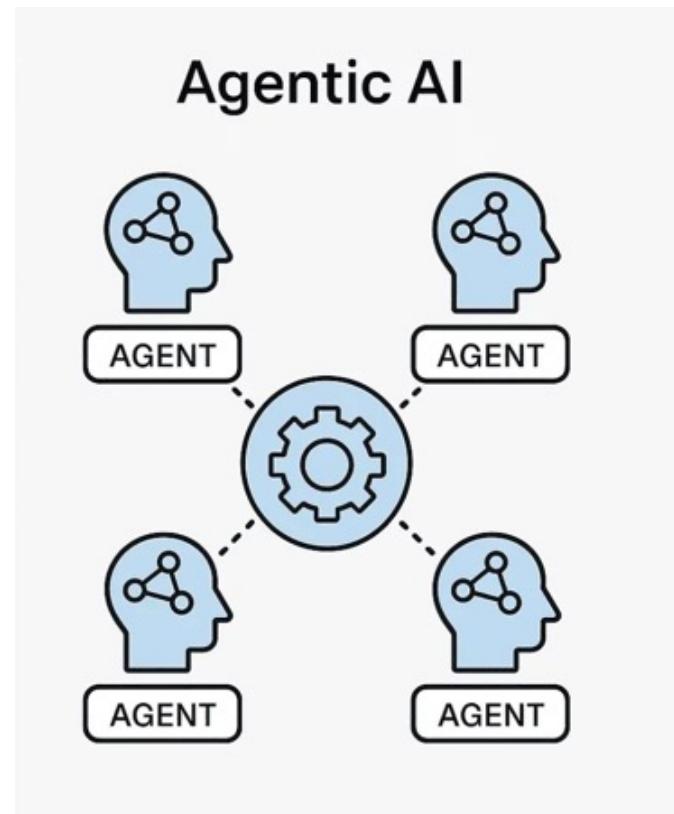
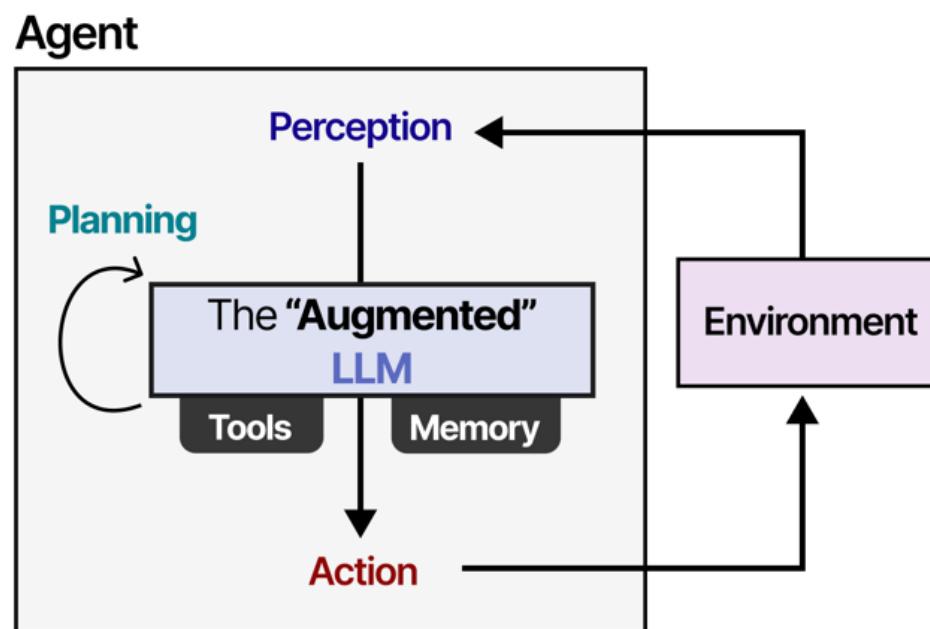
Los agentes intercambian información y colaboran para alcanzar objetivos comunes



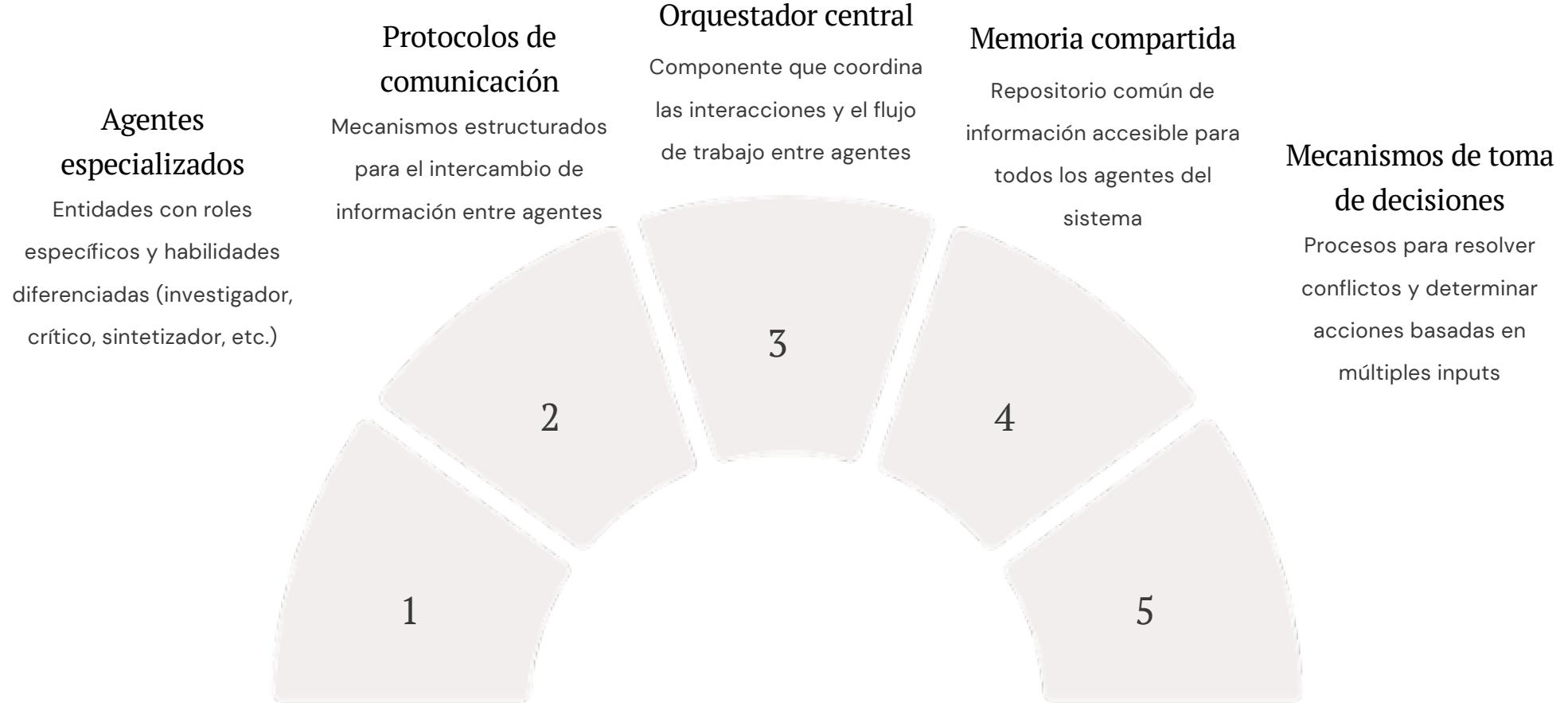
Toma decisiones autónomas

Dentro de su ámbito, cada agente puede decidir cómo abordar mejor los problemas

Agente(s) con tool(s)



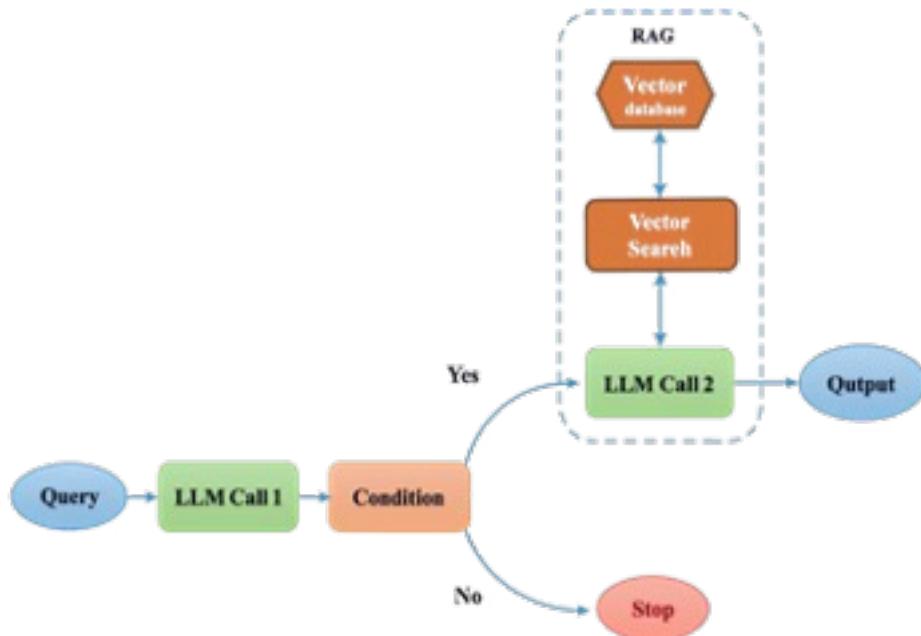
Sistemas Multiagente



Próximos pasos



LLM Workflow



Intercambio



Andres Diaz Pace

andres.diazpace@isistan.unicen.edu.ar