.UBAfiuba
FACULTAD DE INGENIERÍA

Procesamiento de Lenguaje Natural III

Docentes:
Esp. Abraham Rodriguez - FIUBA
Mg. Oksana Bokhonok - FIUBA

Programa de la materia

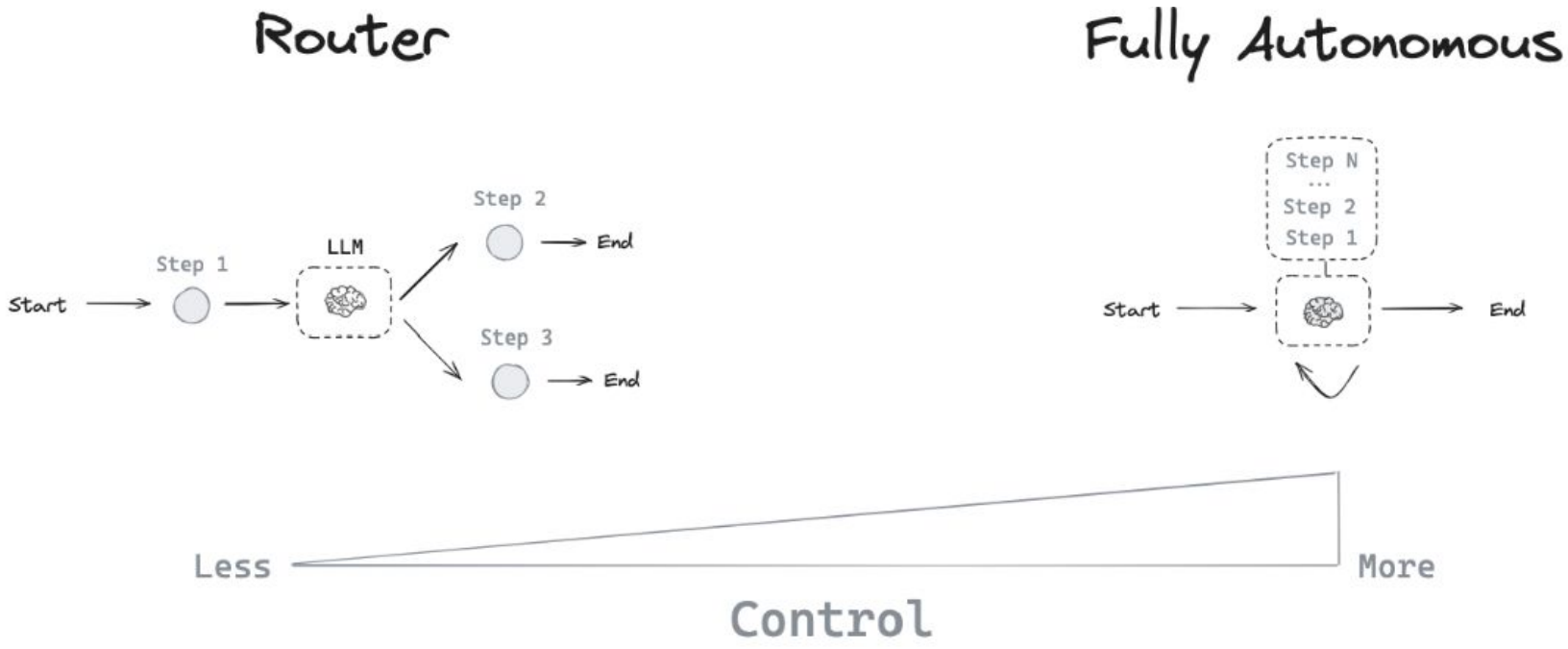
1. RAG avanzado y personalización de soluciones
2. Seguridad. Ética y alineación de modelos con valores humanos.
3. **Sistemas cognitivos y agentes autónomos.**

Arquitecturas de Agentes

Un sistema **Naive RAG** consiste en proveer verdades fundamentadas sobre documentos a una LLM, este es un proceso fijo o hardcoded.

Por otro lado un agente es un sistema que consiste en dar “libertad” a una LLM sobre el control del flujo, en otras palabras es dinámico.

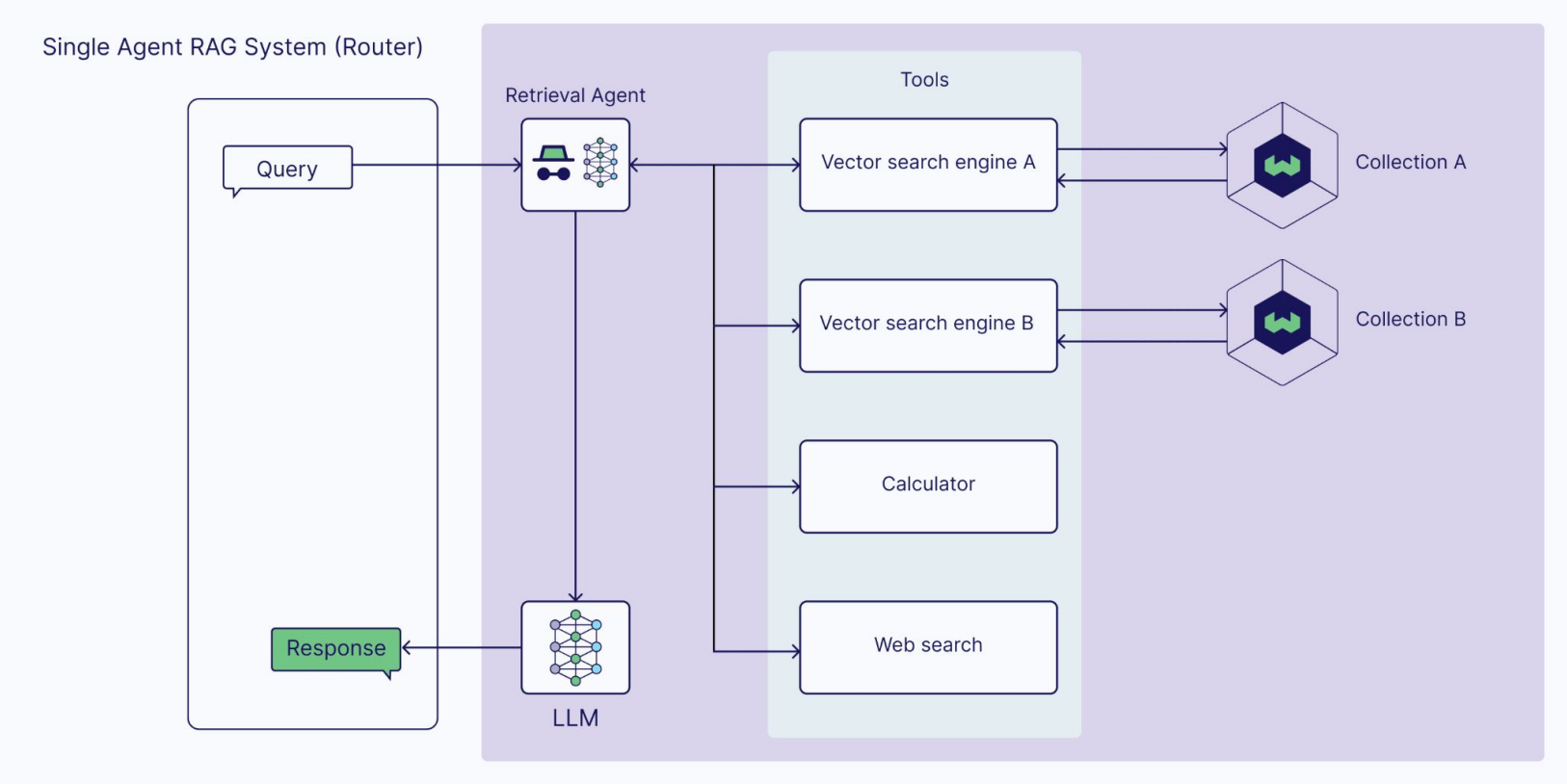
El control proporcionado a un agente es definido mediante la arquitectura utilizada.



Arquitecturas de Agentes (Router)

Un router es realmente un Agentic RAG, el cual consiste en proveer al agente las herramientas necesarias y hacer los llamados on-demand, e.g llamar una REST API, VectorDB, etc.

Útil en sistemas con requerimientos bien definidos y con pocos contextos.



Arquitectura Multi-Agente

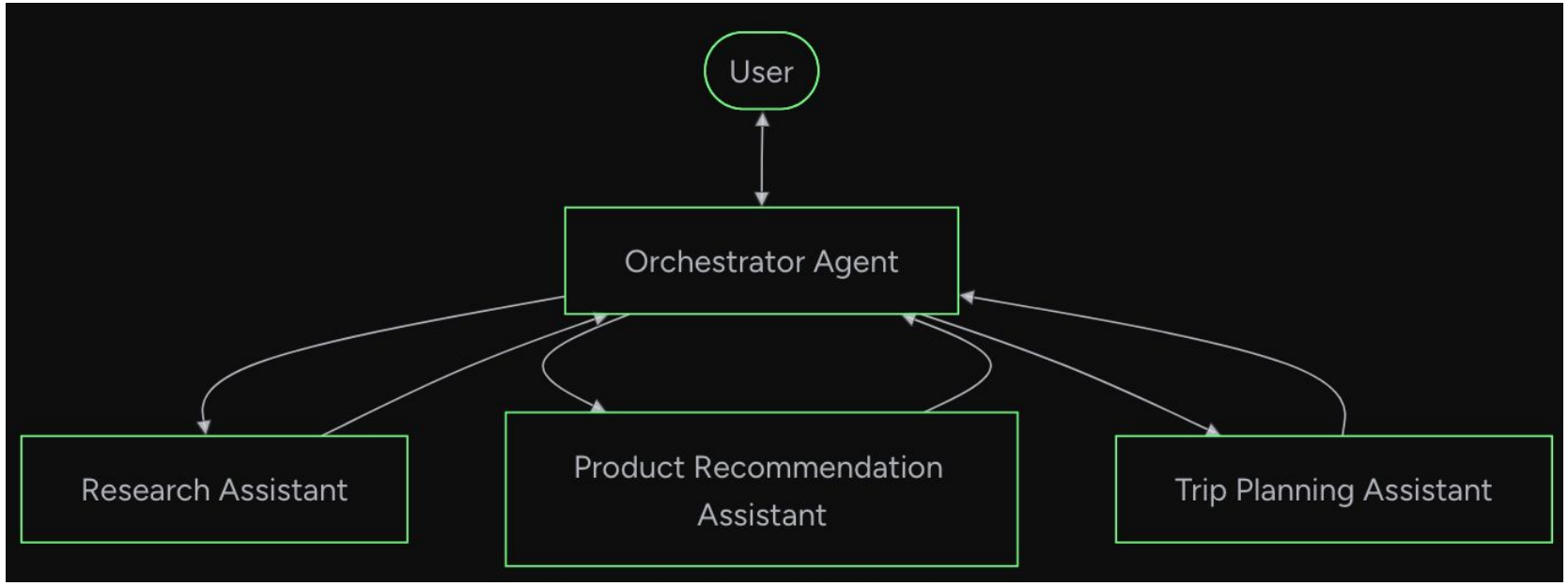
Un **router puede quedarse corto rápidamente** si existen diversos requerimientos o contextos que nuestro sistema debe abarcar. Por ejemplo, un sistema para viajes capaz de:

- Investigar productos
- Recomendar productos
- Planificar viajes

En un enfoque multi-agente, **cada agente tiene un rol más especializado** y puede manejar un subconjunto de contextos o herramientas.

Los agentes pueden **comunicarse entre sí, coordinar acciones y delegar tareas**, evitando que un único router centralizado se sobrecargue.

Esto permite **escalabilidad y adaptación a múltiples contextos** de manera más flexible que con un router único.



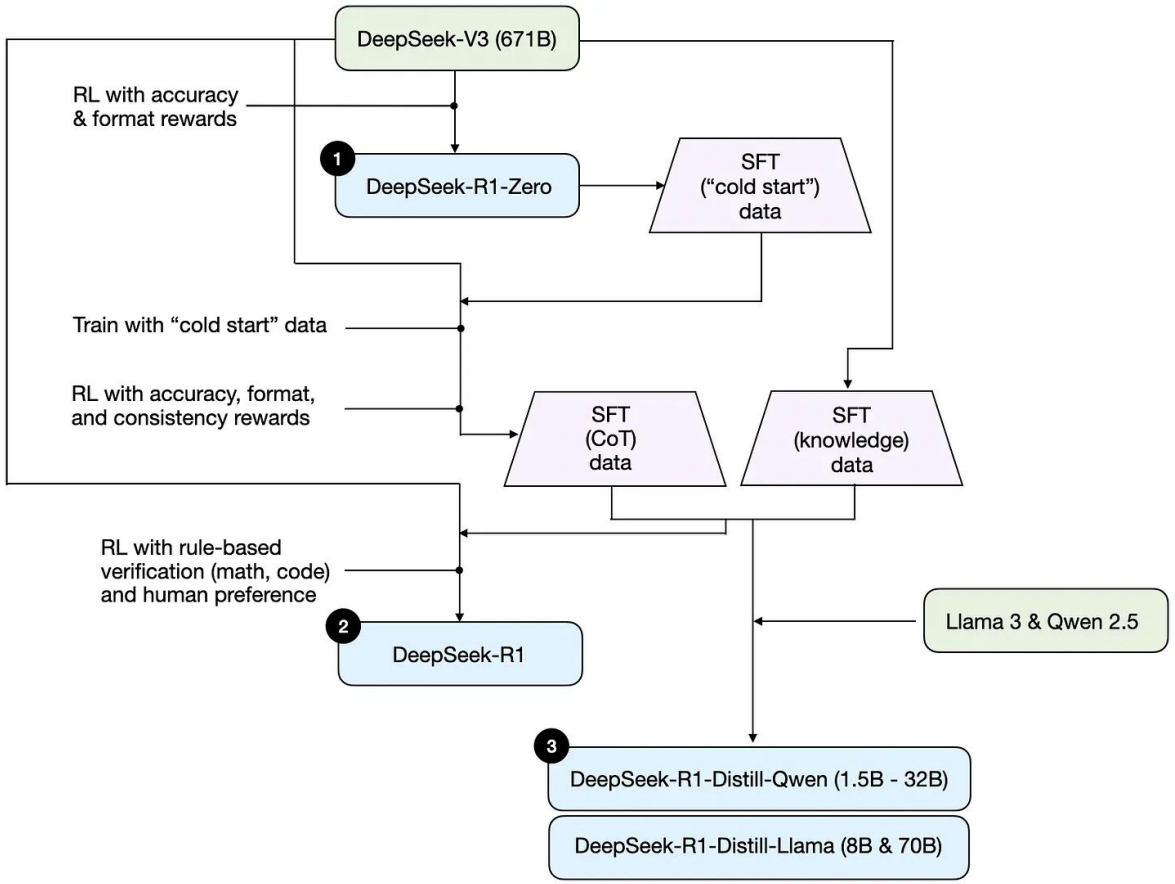
LLMs de razonamiento

Definir qué acción debe realizar un agente LLM tradicionalmente depende del **diseño de prompts**.

Los **modelos de razonamiento** permiten que el agente analice y decida pasos intermedios por **sí mismo.**

Esto **reduce la complejidad de desarrollo** y otorga **mayor independencia y adaptabilidad** al agente frente a múltiples contextos.

En el caso de DeepSeek-R1, el razonamiento surge mediante un proceso de Reinforcement Learning. La data de razonamiento es simplemente chain-of-thought.
Understanding Reasoning LLMs

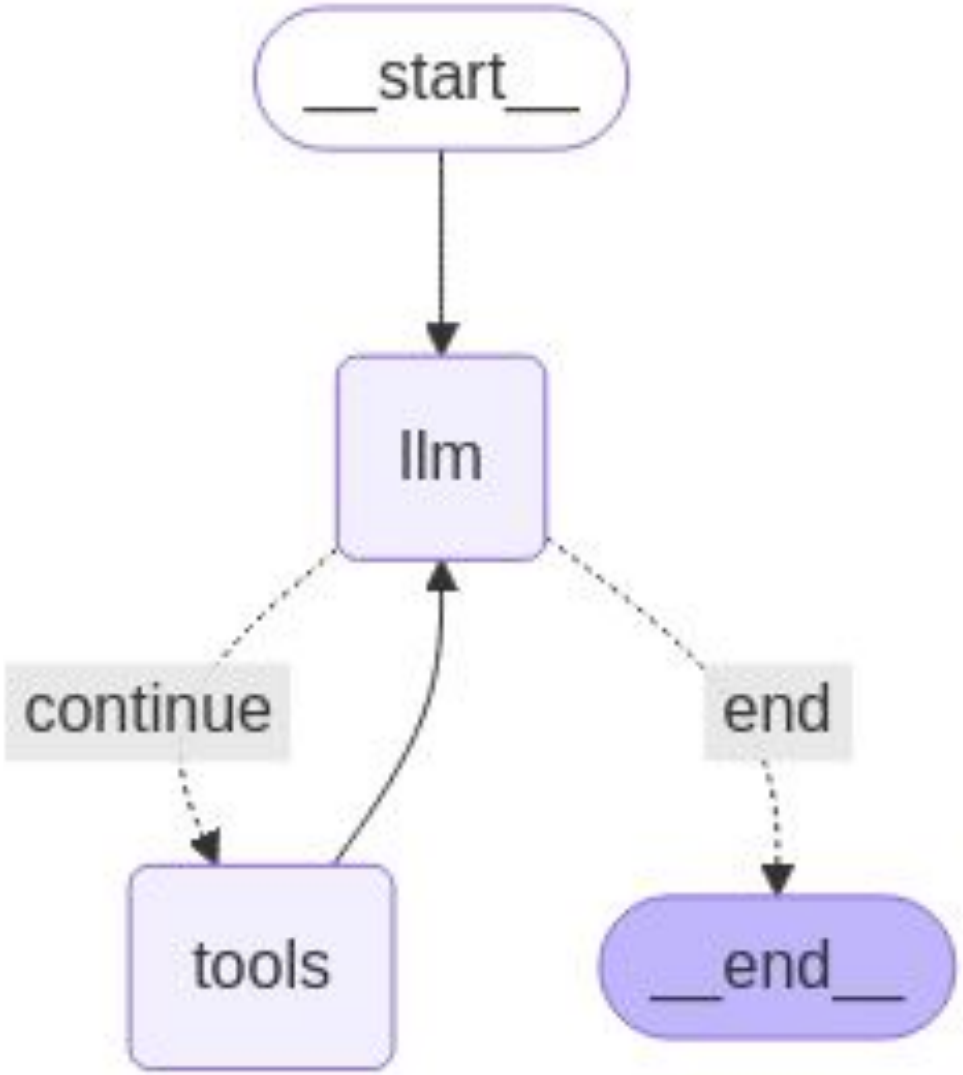


Reason + Act = ReAct

ReAct precede a las LLM de razonamiento, pero consiste en generar razonamiento para generar contexto específico y actuar de acorde.

Surge del Chain-of-Thought el cual que consiste en desglosar un problema en pasos.

CoT es **estático** lo cual carece de la habilidad de interactuar o acceder a contenido externo o simplemente actualizar el conocimiento (contexto) lo cual induce al alucinamiento por carencia de contexto.

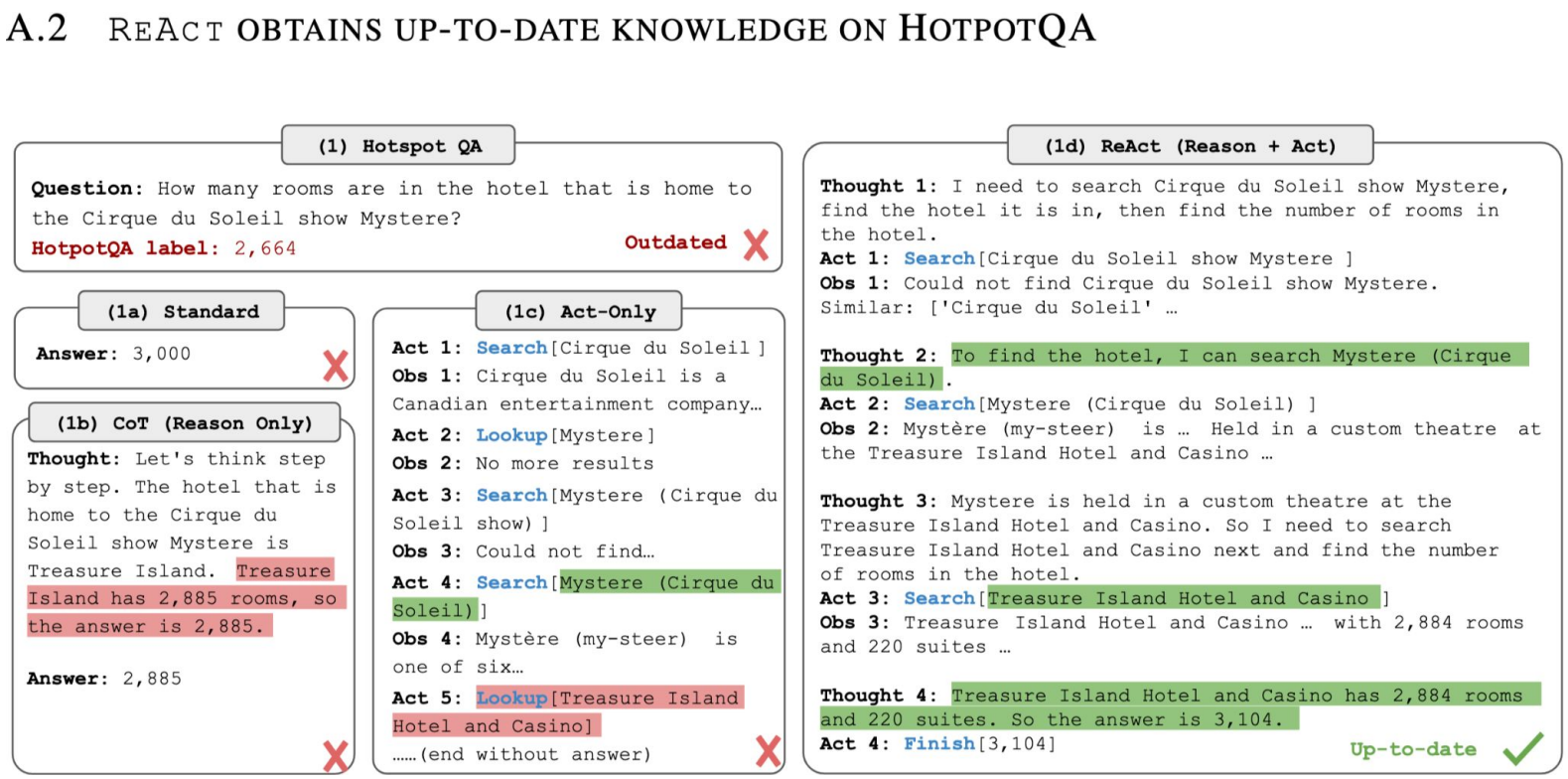


Reason + Act = ReAct

ReAct permite generar razonamiento dinámicamente (a comparación de CoT) además considera la interacción con ambientes externos (**tools**).

Hoy en día, gracias a las LLMs de razonamiento, el concepto de ReAct es **repotenciado** gracias a que el CoT es intrínseco.

[ReAct Paper]



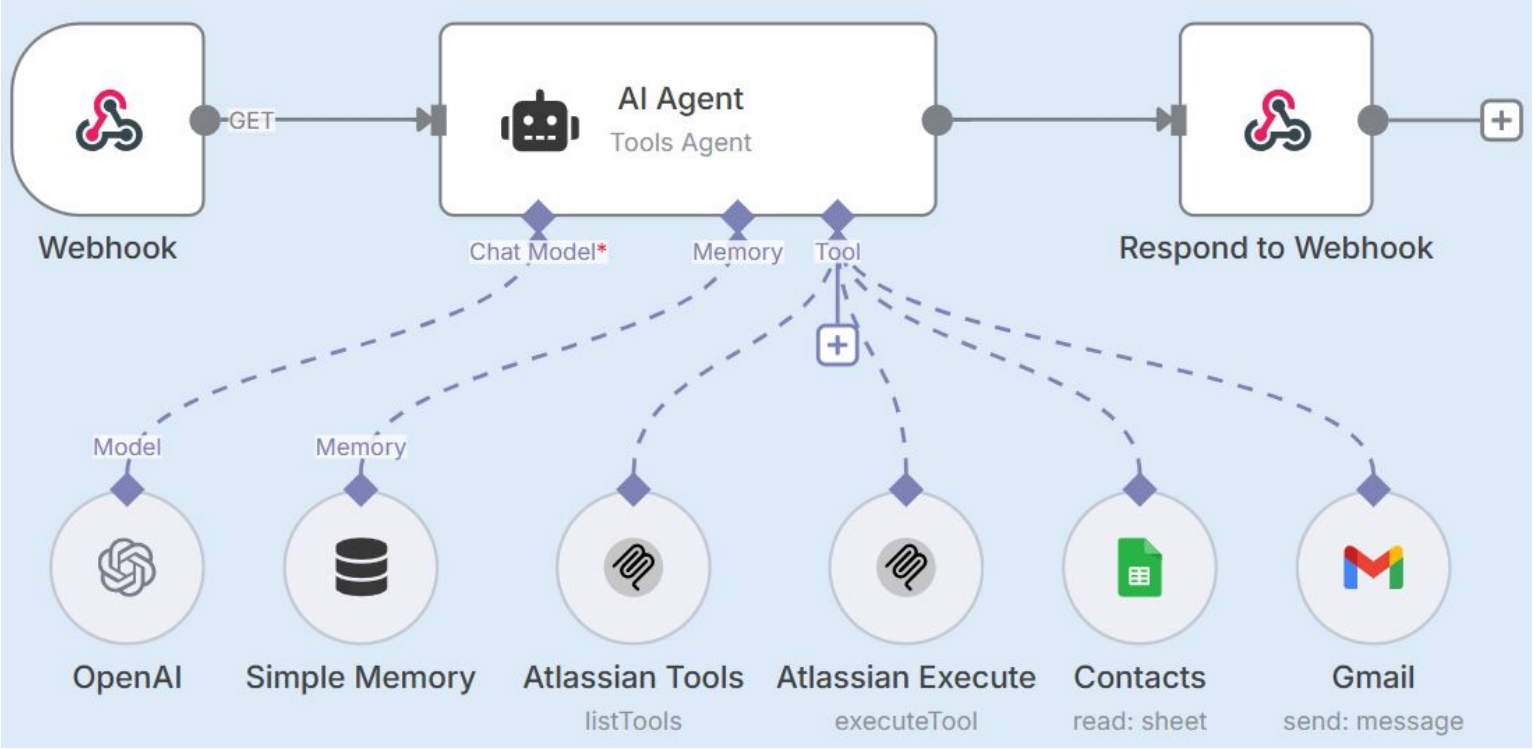
Tooling

Las herramientas son **esenciales** para los agentes.

Permite extender el razonamiento interno al proveer contexto **ajeno** a las capacidades de la LLM para realizar accione (demostrado en ReAct).

Una tool consiste en:

- **Nombre del tool**
- **Descripción/función**
- **Schema**



Tooling

El tool es **inyectado** al contexto para otorgar al agente la labor de **decidir** llamar una tool o no según sea conveniente.

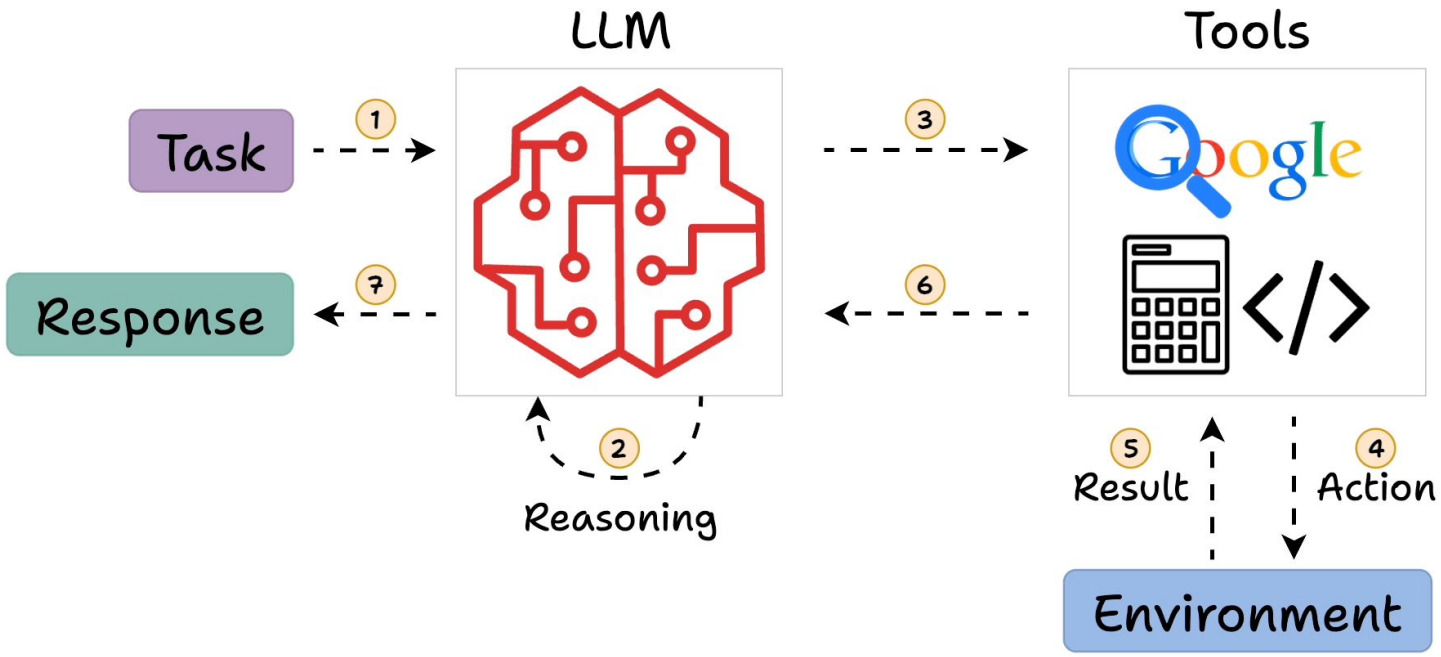
Esto significa que:

SysPrompt = BaseSysPrompt + Tooling

donde:

BaseSysPrompt = Instructions + Persona/Role, Communication structure, etc.

Tooling = available tool description, schema, etc.

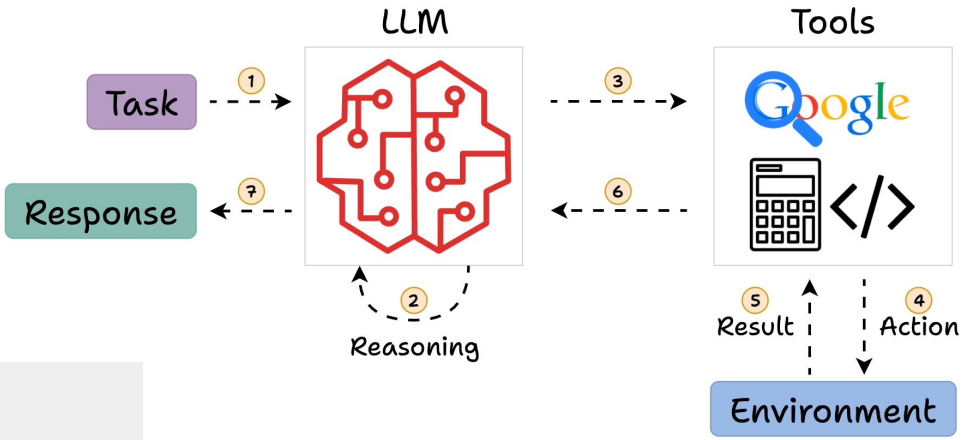


Tooling Prompt Ejemplo

systemPrompt: "**You are an expert in stock market analysis with *access to the "get_stock_price" tool to fetch the current stock price of a company."***

get_stock_price_schema:

```
"input_schema": {
        "type": "object",
        "properties": {
            "company": {
                "type": "string",
                "description": "CompanyName to fetch stock data"
            }
        },
        "required": ["company"]
    }
```

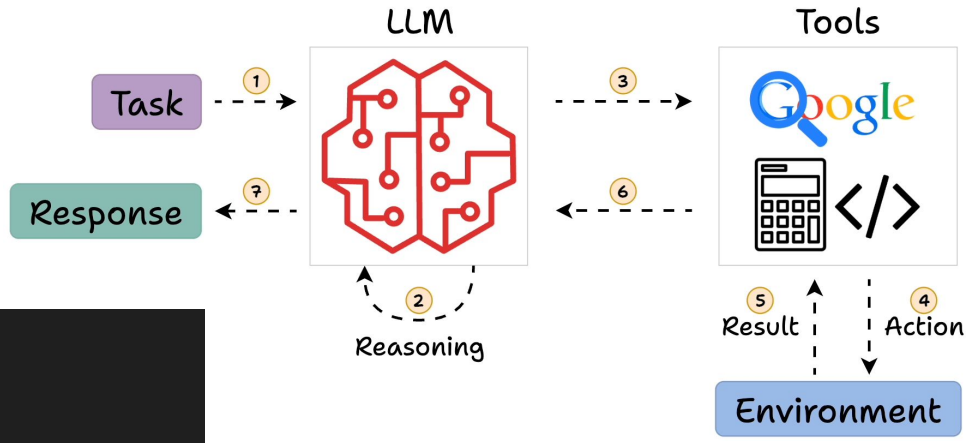


ToolName	*get_stock_price*
Description	Retrieves the current stock price for a given company.
Input	company: string
Output	stockPrice: float

[An introduction to function calling and tool use]

Tooling Patrón

Este patrón es trivial y común por lo tanto, cualquier librería de agents y workflows incluye abstracciones, en el caso de LangGraph/LangChain:




```python
from typing import Optional
from httpx import request
from langchain_core.tools import tool
from pydantic import BaseModel
from .models import Summary

class wikipediaSummarySearch(BaseModel):
    query: str

@tool(args_schema=wikipediaSummarySearch)
def search_wikipedia_summary(query: str) -> Optional[Summary]:
    """Search Wikipedia and return the summary of the top result."""
    url = f"https://en.wikipedia.org/api/rest_v1/page/summary/{query}?redirect=false"
    try:
        response = request("GET",url)
        response.raise_for_status()
    except Exception as e:
        return {"error": str(e)}
    return Summary(**response.json())
```

En este caso la librería se encarga de inyectar al agente el contexto del tool siendo: **el input y output schema y la descripción.**

Tooling Patrón

Se debe ser explícito con respecto a los **types** esperados por el tool.

El typing es contexto para la LLM.

Comúnmente se utiliza Pydantic para definir los types.

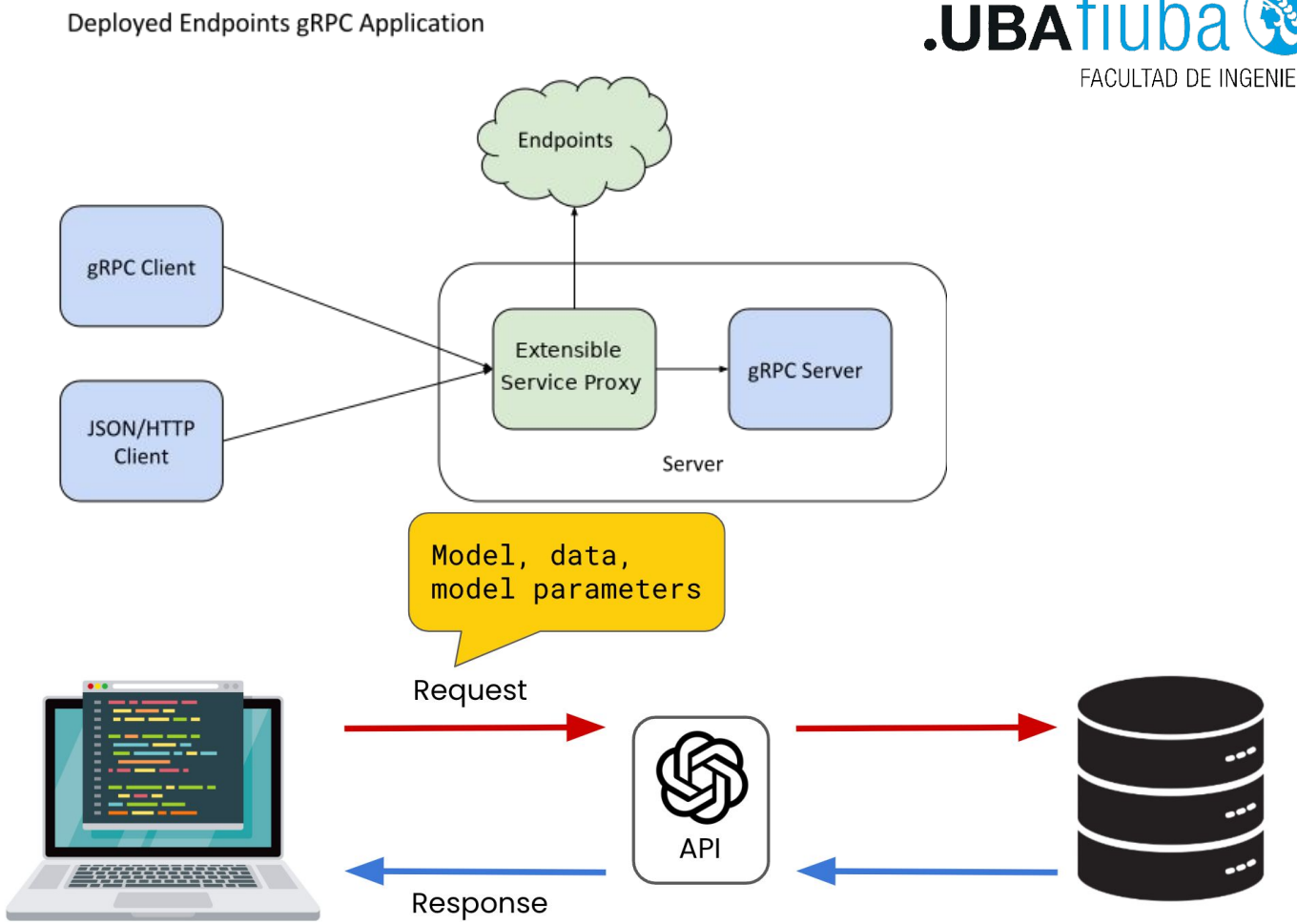
```python
from typing import Optional
from httpx import request
from langchain_core.tools import tool
from pydantic import BaseModel
from .models import Summary

class wikipediaSummarySearch(BaseModel):
    query: str

@tool(args_schema=wikipediaSummarySearch)
def search_wikipedia_summary(query: str) -> Optional[Summary]:
    """Search Wikipedia and return the summary of the top result."""
    url = f"https://en.wikipedia.org/api/rest_v1/page/summary/{query}?redirect=false"
    try:
        response = request("GET",url)
        response.raise_for_status()
    except Exception as e:
        return {"error": str(e)}
    return Summary(**response.json())
```

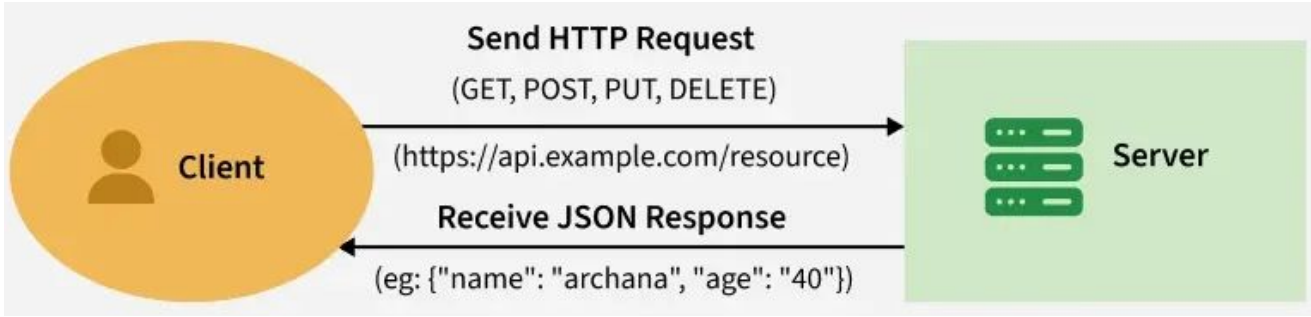
REST APIs y gRPC

La base de los agentes es el consumo de APIs, lo cual es un tema de **ingeniería de software**, en pocas palabras los agentes es en su mayoría sobre integraciones con APIs.

La razón es porque a través de la web, son las APIs las que **proveen contexto**.

Además, lo más común es consumir proveedores de LLMs, e.g OpenAI.

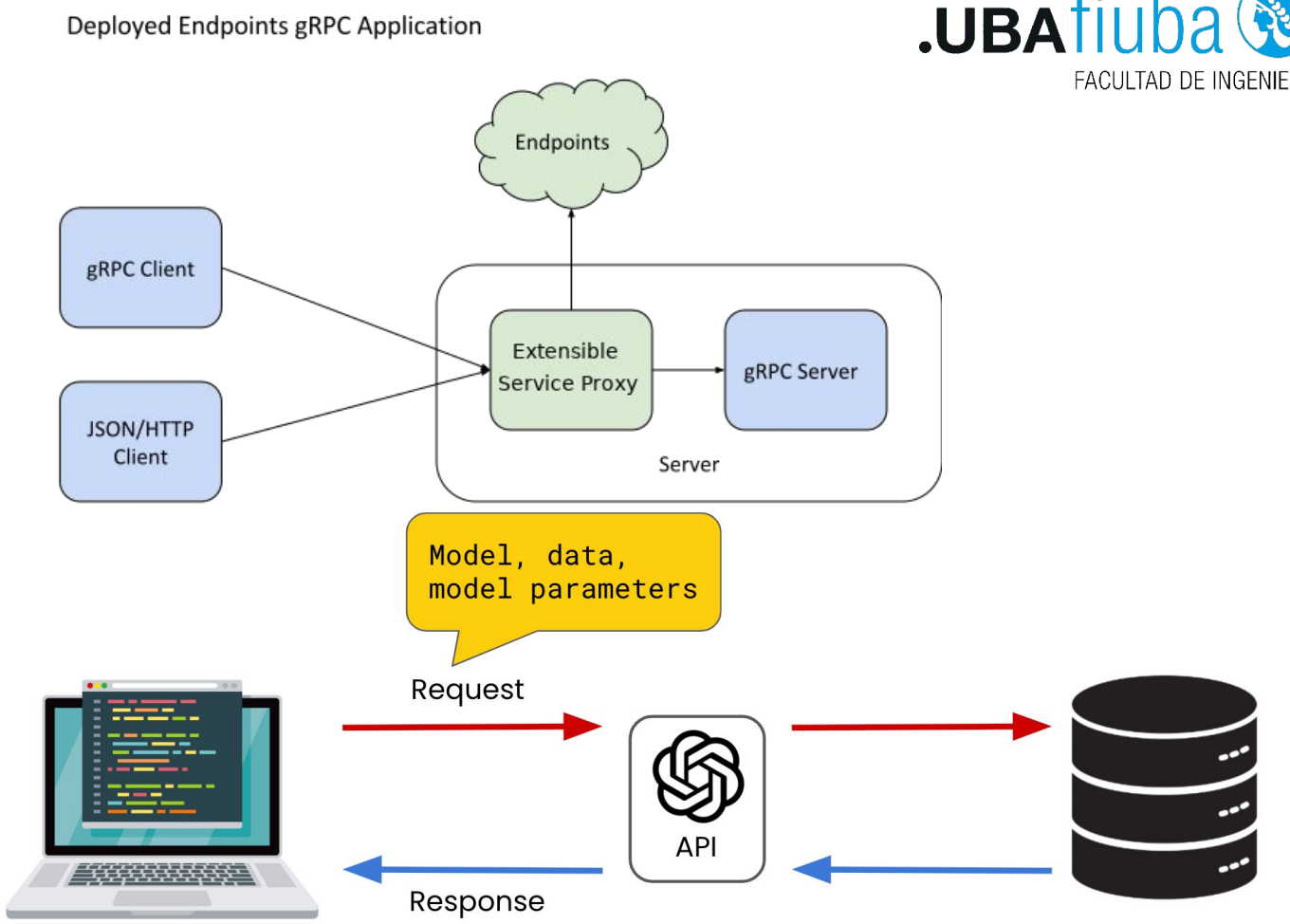


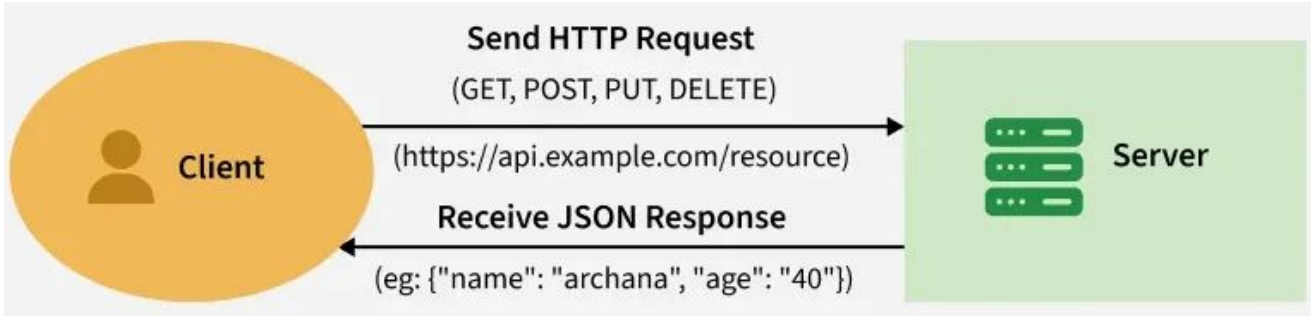


REST APIs y gRPC

Los agentes son **agnósticos** a los medios de comunicación (REST, gRPC, etc.) ya que existen sobre una capa arriba como aplicación, esto permite crear integraciones de tools híbridas e.g:

- getItemAvailability (REST)
- putItemInCart (gRPC)





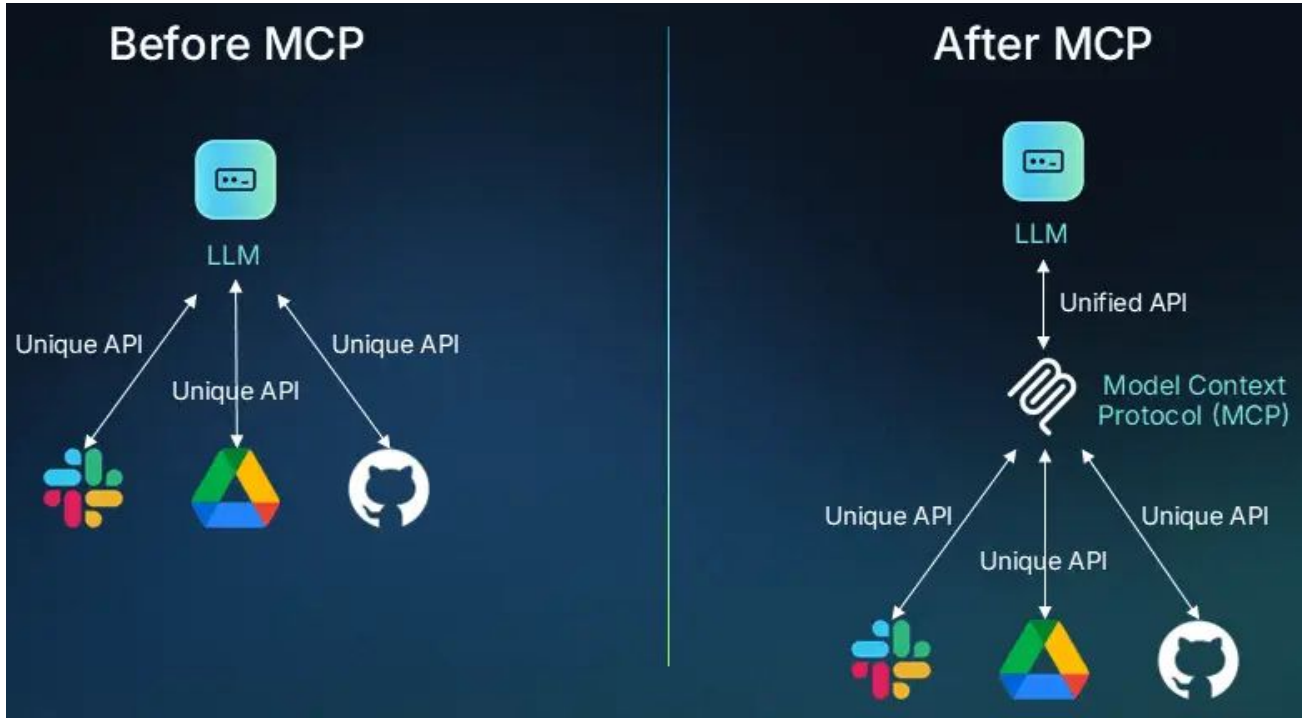
Model Context Protocol (MCP)

El **tooling** en agentes es difícil de mantener porque **carece de un estándar** y depende del protocolo específico (REST, gRPC, etc.) de cada herramienta.

Esto genera un problema de escalamiento tipo **N×M** entre **agentes** y **tools**.

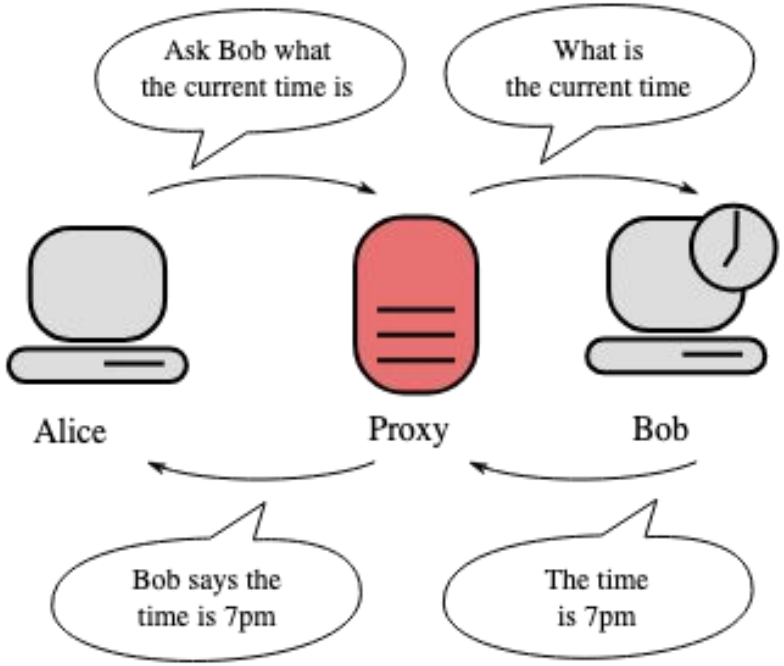
MCP propone un **estándar** que desacopla al agente de las conexiones y protocolos, centralizando la integración.

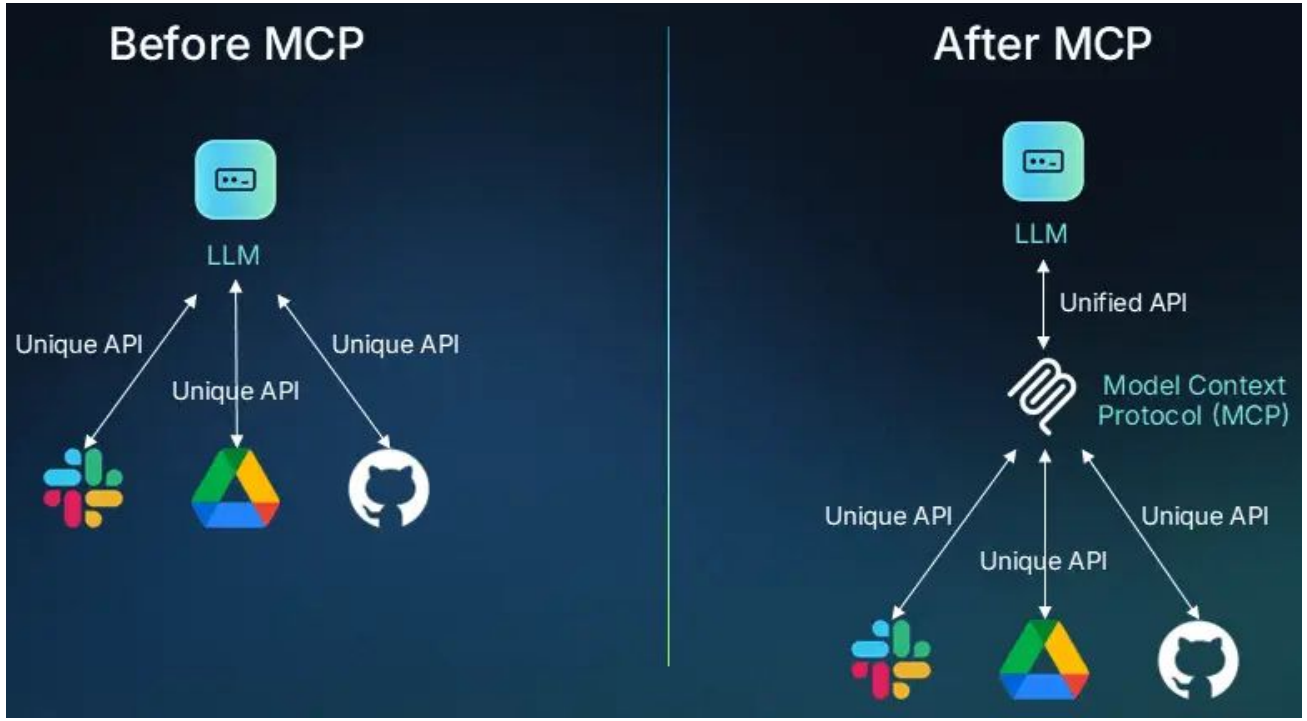
No se elimina la complejidad de integrar tools, **simplemente se delega**, el agente en lugar de llamar **N tools,** llama a un MCP central.



Model Context Protocol (MCP)

Para capturar bien la idea, podríamos tomar el concepto de proxy





MCP Layers

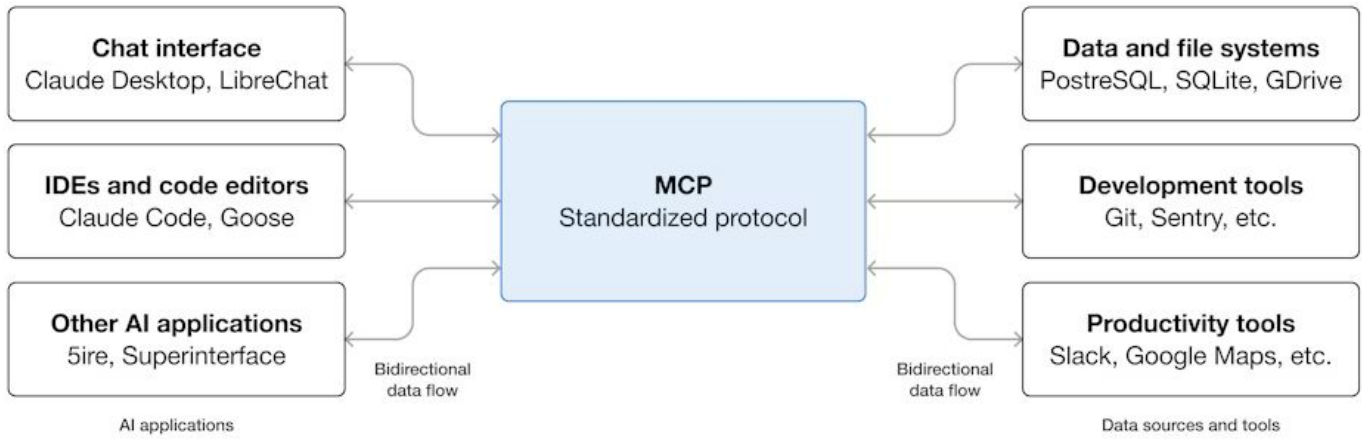
La arquitectura de MCP consiste en dos capas:

- Data
- Transport

La capa de data utiliza JSON-RPC 2.0 para definir las estructuras y semánticas.

La capa de transporte soporta Standard I/O (stdio) como pipes, Streamable HTTP y Websockets.

Capa	Mecanismo
Data	JSON-RPC 2.0
Transport	HTTP, etc.



MCP Server

Los MCP Servers exponen tres funcionalidades:

- Tools
- Recursos
- Prompts

Tools es exactamente igual a lo visto anteriormente.

Son controlados por el LLM sin embargo, MCP enfatiza la supervisión humana.

```
{
  name: "searchFlights",
  description: "Search for available flights",
  inputSchema: {
    type: "object",
    properties: {
      origin: { type: "string", description: "Departure city" },
      destination: { type: "string", description: "Arrival city" },
      date: { type: "string", format: "date", description: "Travel date" }
    },
    required: ["origin", "destination", "date"]
  }
}
```

Funcion	Descripción	Ejemplo	Quién lo controla
Tools	Escribir en bases de datos, llamar APIs, modificar archivos, etc.	• Buscar viajes. • Crear eventos en calendario	LLM
Recursos	Fuentes de datos de lectura siendo: ***archivos, table schemas o APIs.***	• Knowledge Base • Document retriever	Aplicación
Prompts	Instrucciones sobre cómo utilizar los tools y recursos.	-	User

MCP Server

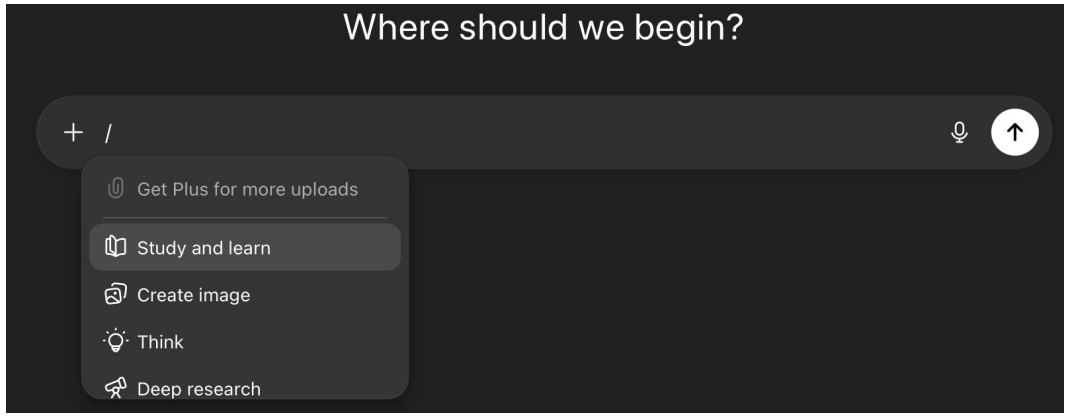
Los recursos son contexto únicamente de lectura, las aplicaciones determinan cómo acceder a esa información, en el caso de **planificación de viajes, los recursos proporcionan a la aplicación de IA acceso a información relevante:**

- **Datos del calendario (calendar://events/2024)** - Verifica la disponibilidad del usuario
- **Documentos de viaje (file:///Documents/Travel/passport.pdf)** - Accede a documentos importantes
- **Itinerarios anteriores (trips://history/barcelona-2023)** - Consulta viajes y preferencias pasadas

Funcion	**Descripción**	**Ejemplo**	**Quién lo controla**
Tools	Escribir en bases de datos, llamar APIs, modificar archivos, etc.	• Buscar viajes. • Crear eventos en calendario	LLM
Recursos	Fuentes de datos de lectura siendo: ***archivos, table schemas o APIs.***	• Knowledge Base • Document retriever	Aplicación
Prompts	Instrucciones sobre cómo utilizar los tools y recursos.	-	User

MCP Server

Los prompts son simplemente templates para definir inputs esperados y patrones de interacción.
Son invocados por los users al ser expuestos por la UI de la aplicación.




```
{
  "name": "plan-vacation",
  "title": "Plan a vacation",
  "description": "Guide through vacation planning process",
  "arguments": [
    { "name": "destination", "type": "string", "required": true },
    { "name": "duration", "type": "number", "description": "days" },
    { "name": "budget", "type": "number", "required": false },
    { "name": "interests", "type": "array", "items": { "type": "string" } }
  ]
}
```

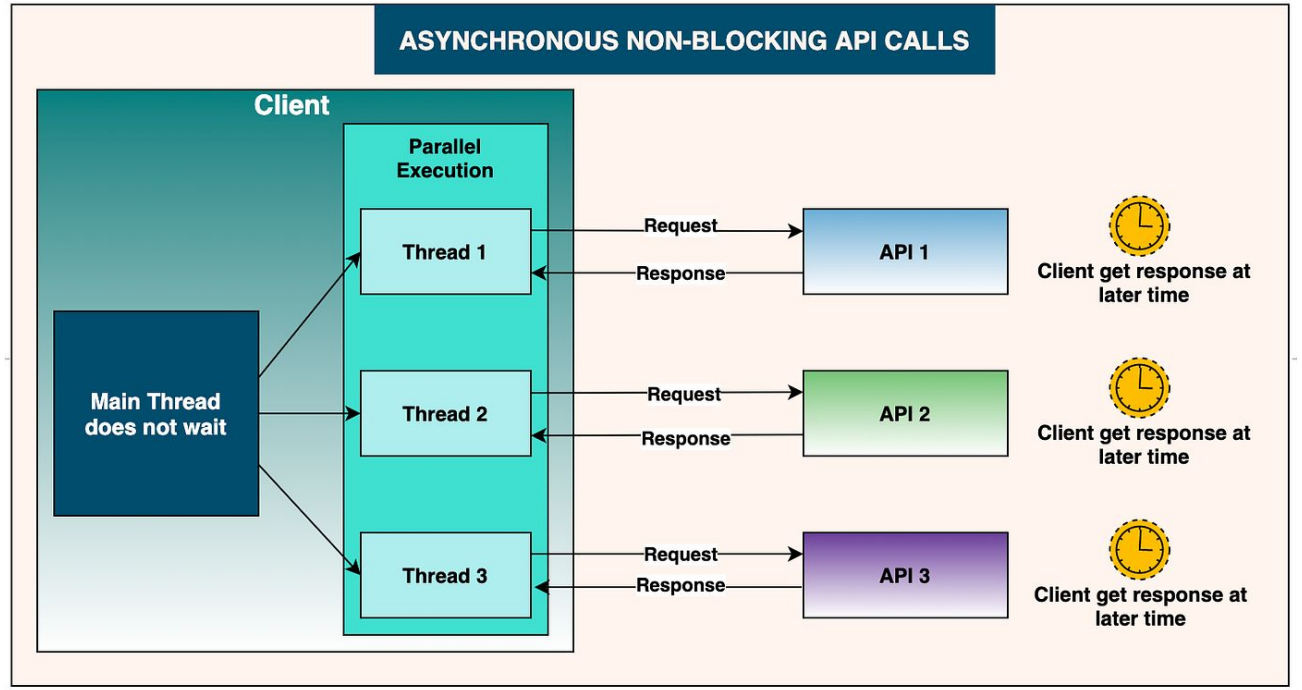
Funcion	Descripción	Ejemplo	Quién lo controla
Tools	Escribir en bases de datos, llamar APIs, modificar archivos, etc.	• Buscar viajes. • Crear eventos en calendario	LLM
Recursos	Fuentes de datos de lectura siendo: ***archivos, table schemas o APIs.***	• Knowledge Base • Document retriever	Aplicación
Prompts	Instrucciones sobre cómo utilizar los tools y recursos.	-	User

MCP/Tools Latencia y complejidad

Como MCP es una capa intermedia puede agregar un poco de latencia pero en su mayoría **los tools pueden ser el cuello de botella** del sistema e.g un endpoint puede tomar 5 segundos.

Para esto la solución no es más que async, threadpools o concurrencia, workerpools, etc.

[FastMCP Async tools]

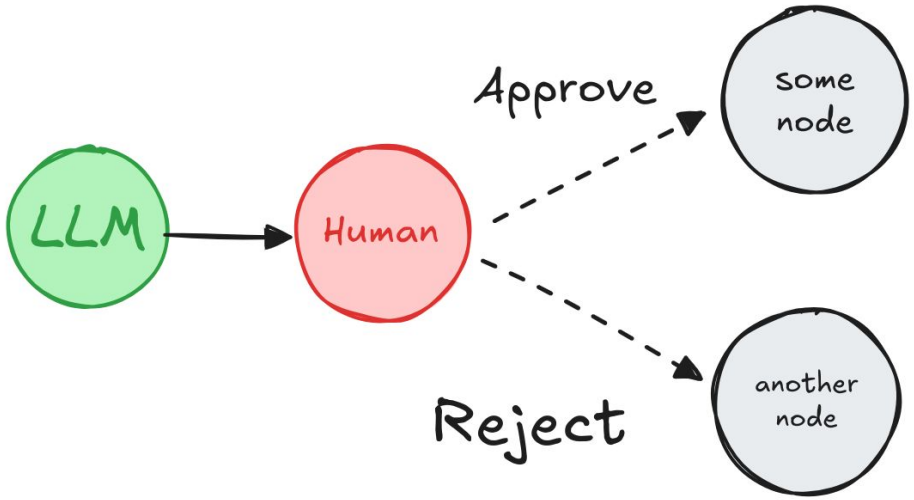


Interrupciones y Human-in-the-loop

Es muy común que los agentes no sean autónomos y requieren de interacciones humanas, en sistemas **customer-facing** ejemplo:

- Reservaciones de cuartos de hotel.
- Compras de stocks en mercado de valores.
- Transacciones y aprobaciones.
- Copiloto de desarrollo

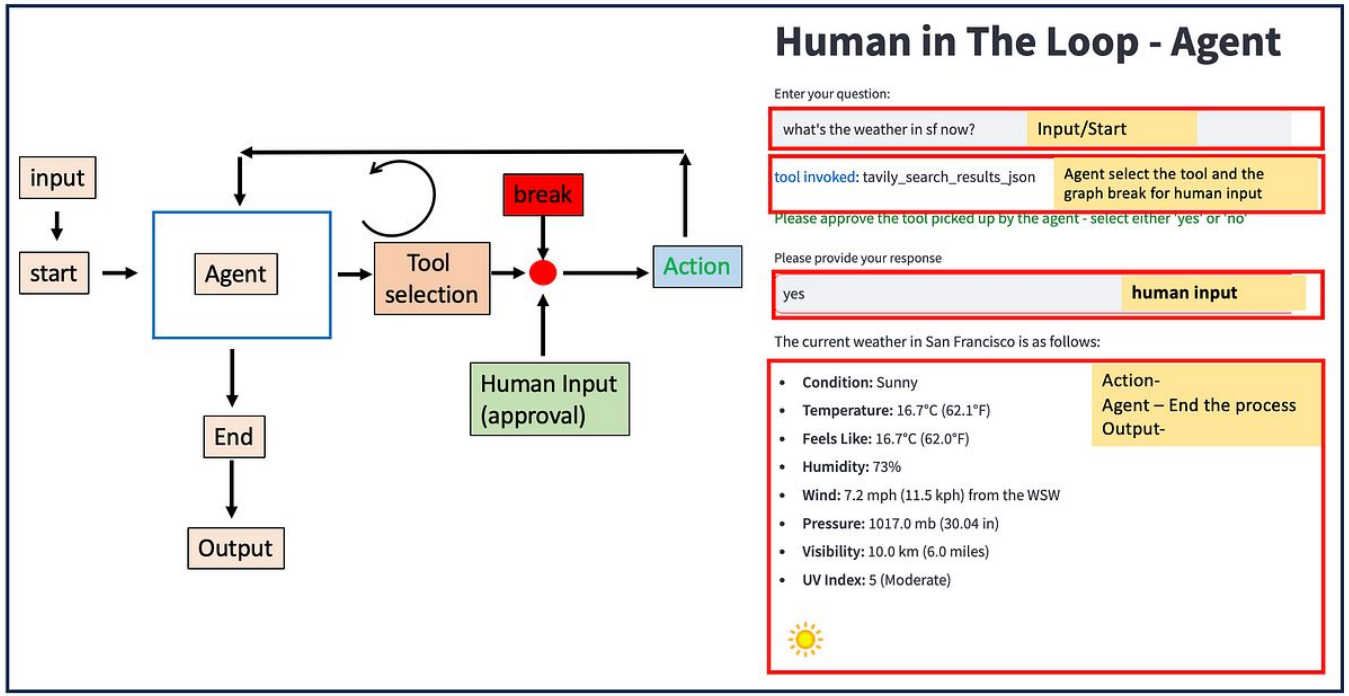
Human-in-the-loop (HITL) es utilizado para confirmar, revisar y editar una **acción critica** a realizar.



Human-in-the-loop

[LangGraph HITL]

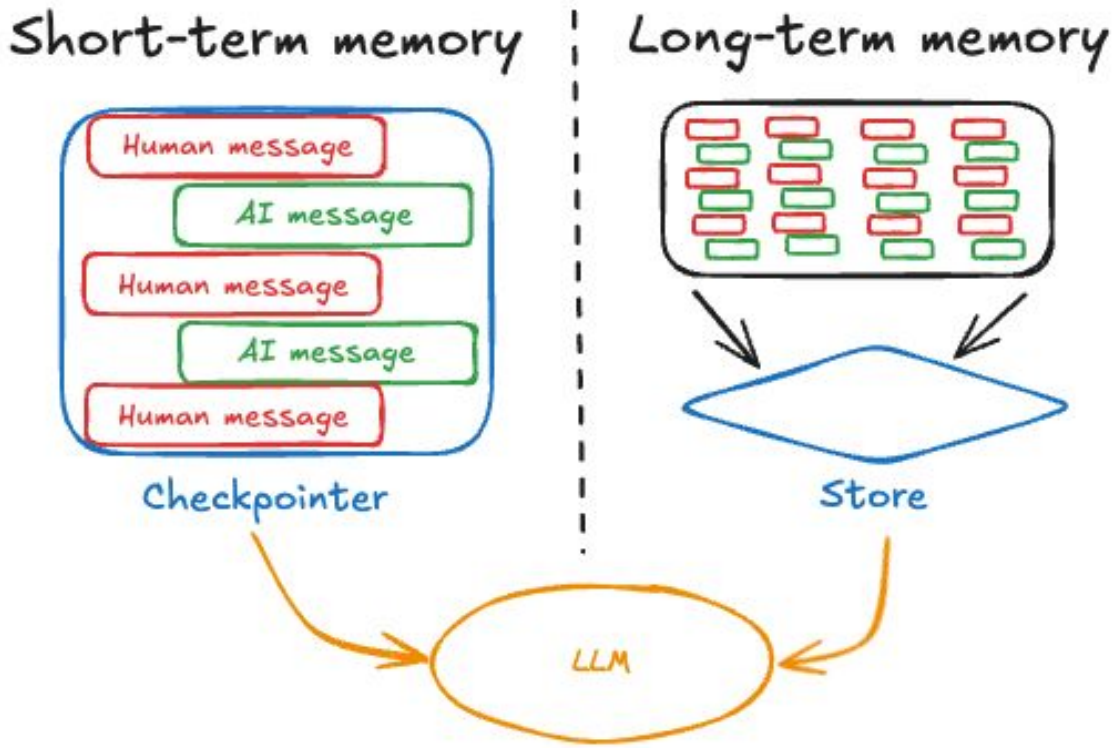
[LlamaIndex HITL]



Manejo de memoria y sesiones

La memoria es **crucial** en agentes para recordar las interacciones pasadas “aprender” mediante interacciones y adaptarse al usuario.

Al escalar un agente a múltiples usuarios, la memoria se vuelve aún más importante al lidiar con la **eficiencia, latencia, throughput y satisfacción de los usuarios.**

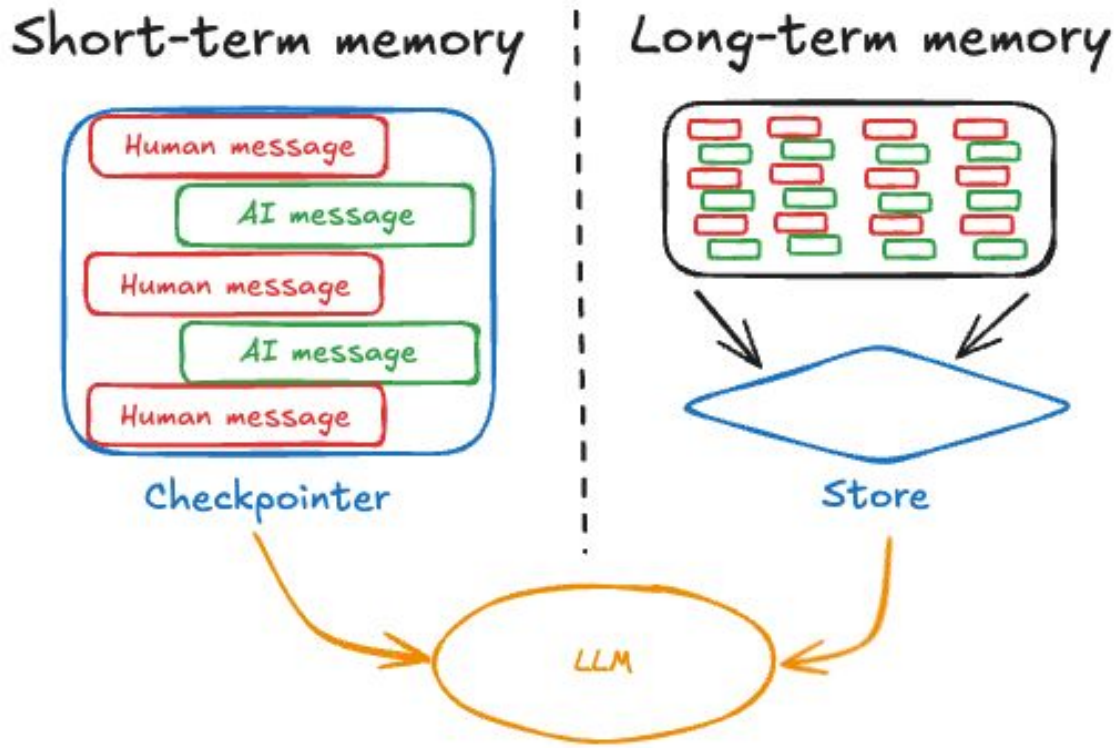


Memoria a corto plazo

A nivel de conversación o sesión es el tipo de memoria más común en agentes y servicios de LLMs.

Está fuertemente asociado al contexto length que la LLM permita.

Se debe tener consideración que si bien el context length de LLMs del estado del arte rondan en 128k+ entre mas grande, mas computo es necesario y la latencia reduce.



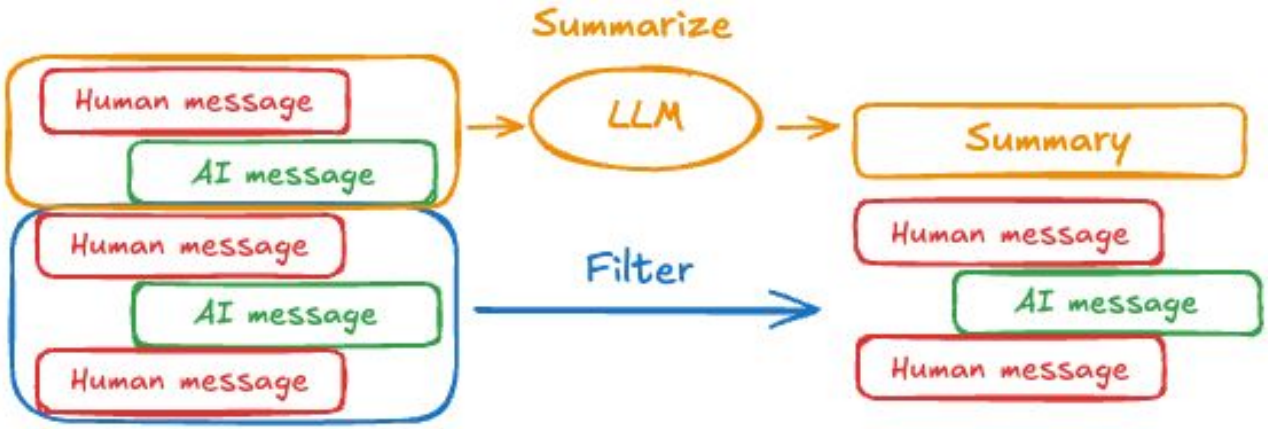
Para lidiar con memoria a corto plazo existen diversas estrategias sobre la conversación como:

- Recorte
- Eliminación
- Resumen/compresión

Memoria a corto plazo

El problema de recortar o eliminar mensajes es que se pierde el contexto.

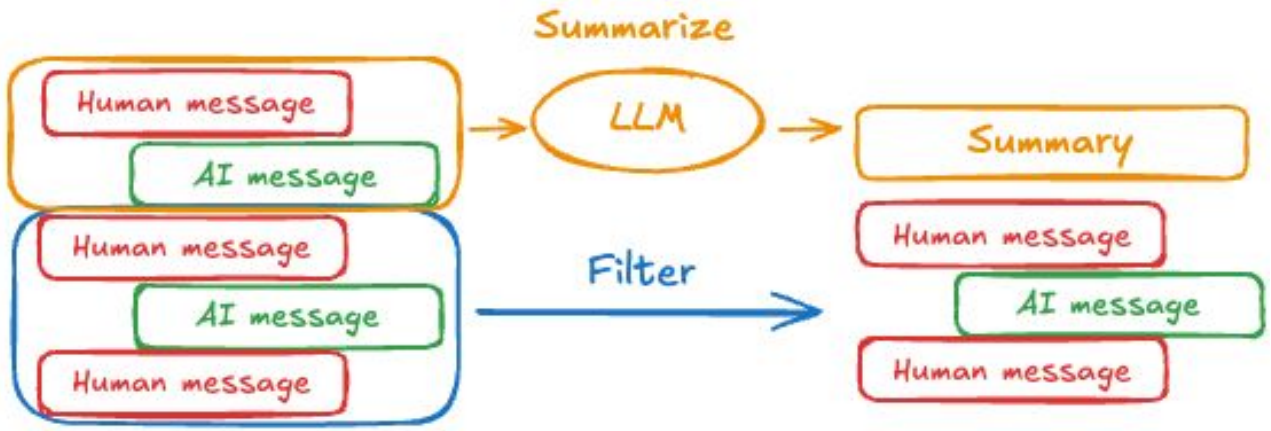
Lo ideal es realizar una síntesis del historial conversacional.



Manejo de memoria a corto plazo

En LangGraph se utilizan ***checkpoints*** para la memoria a corto plazo.

Checkpoints es una de las capas core de LangGraph para persistencia.

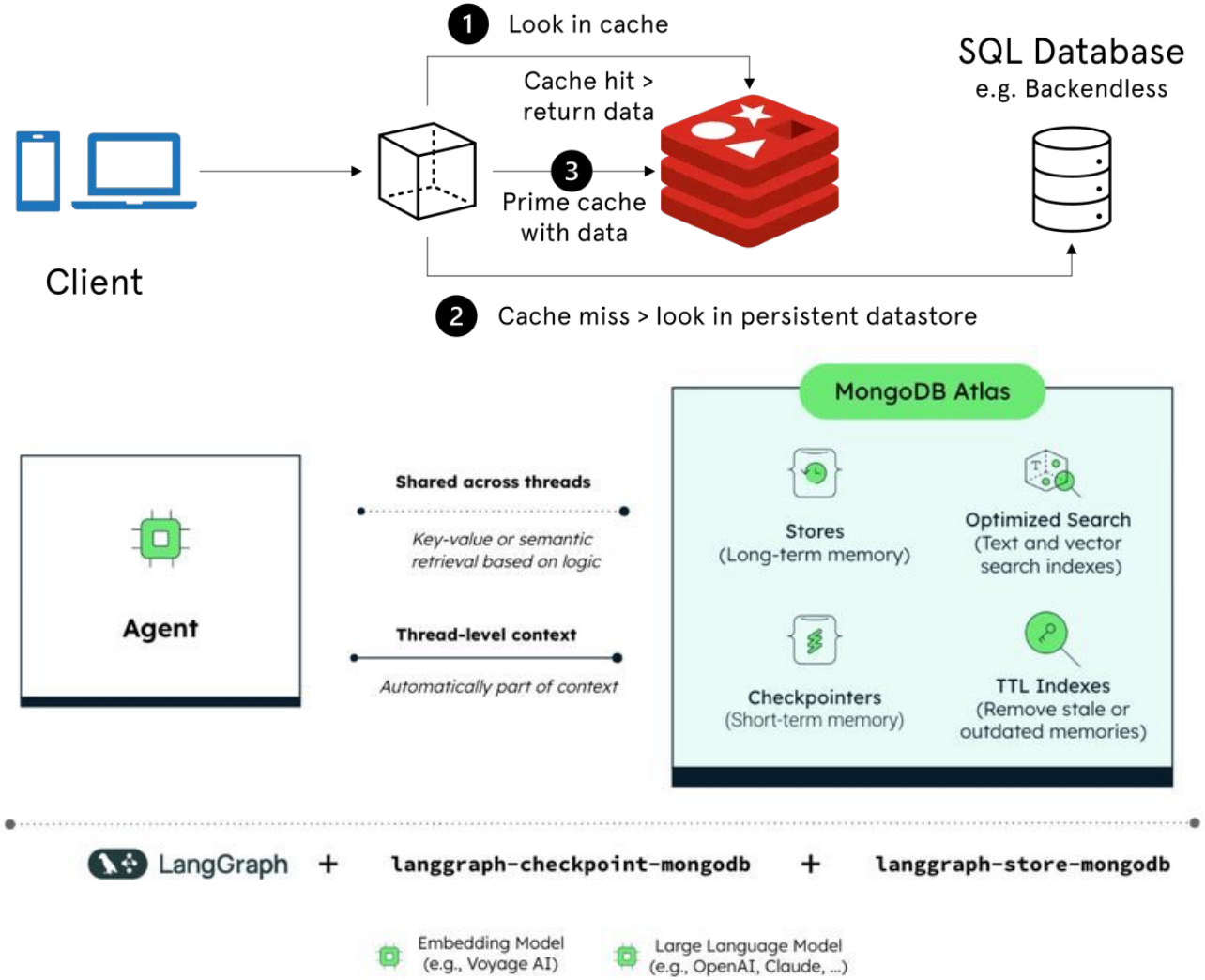


Manejo de memoria y sesiones

Checkpoint es equiparable a cache y persistencia en un sistema backend tradicional.

[LangGraph-Redis]

[LangGraph-Redis Example]

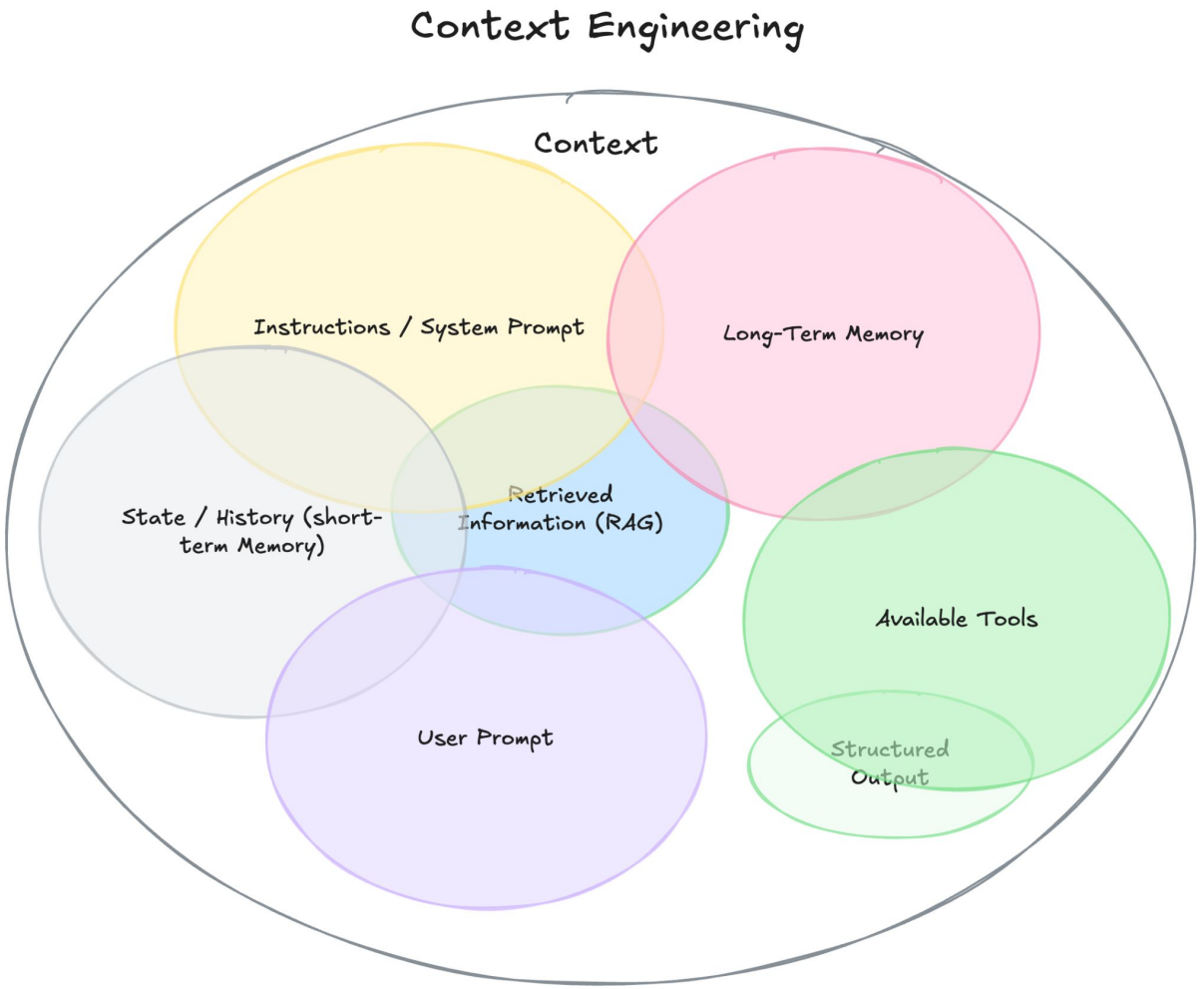


Context Engineering

ReACT, LLMs de razonamiento y MCP/Tools, demuestran que **el contexto no es estático** para cumplir labores de manera precisa y reducir las alucinaciones.

Prompt engineering **puede no ser** suficiente especialmente en sistemas reactivos o adaptativos, un prompt es un conjunto de reglas estáticas las cuales es **virtualmente imposible tener en cubrir cada caso.**

Context Engineering es simplemente proveer el contexto necesario para resolver una tarea on-demand. En otras palabras, proveer al sistema la información y herramientas correctas en el momento correcto



Herramientas de Workflows

Workflows

Los agentes no son necesarios todo el tiempo, si existen labores asíncronas un workflow puede satisfacer la necesidad ejem:

- Leer archivos de google y generar correos.
- Schedule tasks para enviar correos a diario.
- Generacion de newsletters mediante RSS.
- Análisis de formularios.

n8n

n8n es una herramienta para crear workflows popularizada por la integración con LLMs, dando la capacidad de crear workflows con capacidad cognitiva al ser capaz de integrarse a las APIs de proveedores de modelos e incluso modelos locales.

Código y demo

n8n + ollama (local)

Utilizar docker

Ollama docker

n8n docker

Supervisor + ReAct

LangGraph Supervisor

LangGraph ReAct

Preguntas?