

Inteligencia Artificial Generativa Para la Ciencia de Datos



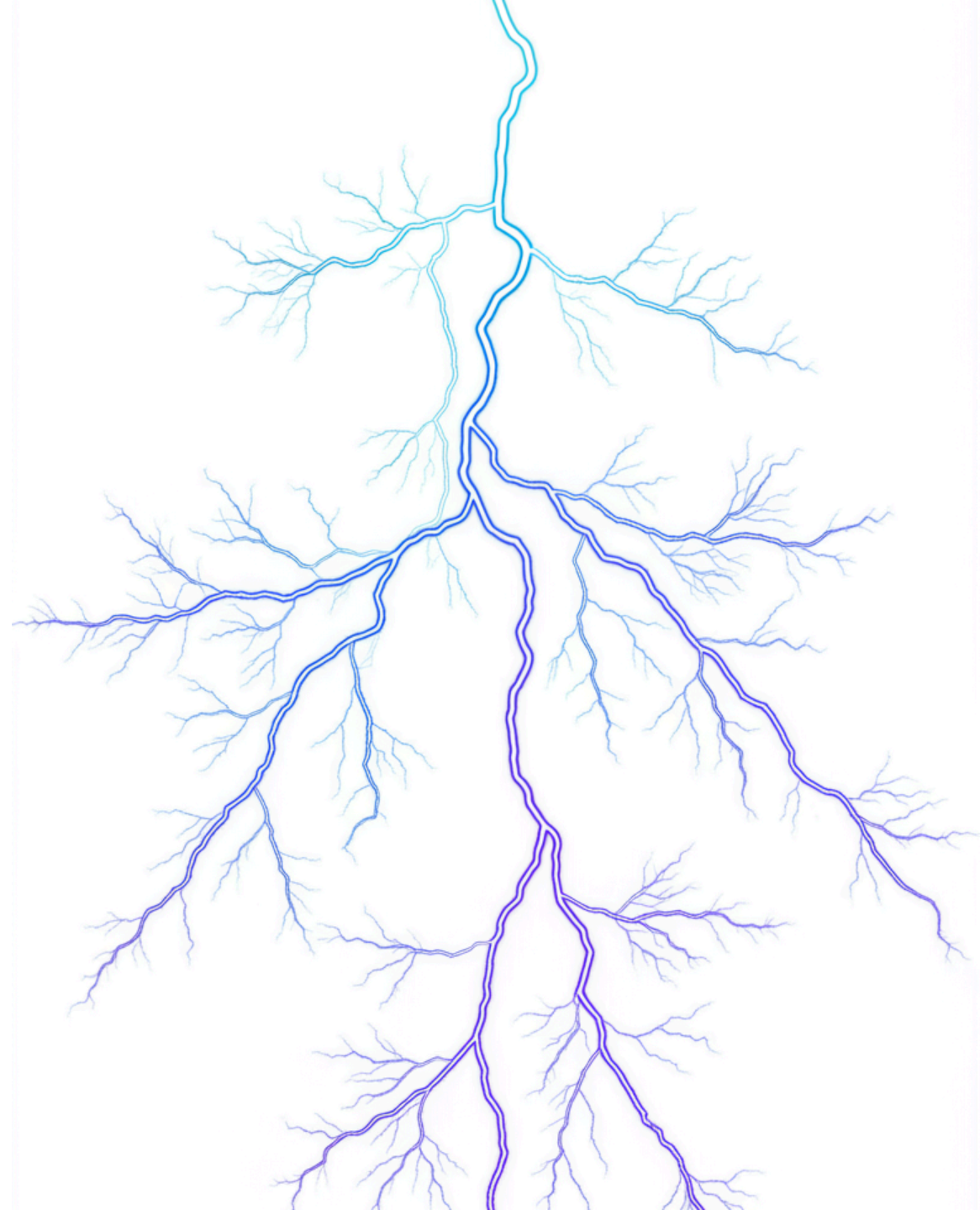
Profesor

Juan Camilo Vega Barbosa

Consultor IA - Ingeniero IA/ML



LinkedIn



Agentes de IA 🤖

Sistemas inteligentes que perciben, razonan y actúan





¿Qué son los Agentes de IA?



Los agentes IA son sistemas de inteligencia artificial diseñados para interactuar con su entorno, tomando decisiones independientes basadas en la percepción de ese entorno y el objetivo que deben cumplir, similar a cómo un asistente humano ejecutaría tareas complejas con mínima supervisión.







Su arquitectura integra percepción, razonamiento y acción, permitiéndoles utilizar herramientas externas como navegadores web, APIs, bases de datos o aplicaciones, ampliando significativamente su capacidad operativa más allá de la simple generación de respuestas textuales.




Mantienen un estado interno o "memoria" que les permite rastrear objetivos, recordar interacciones previas y adaptar estrategias en función de resultados, creando asistentes que evolucionan con el tiempo y se vuelven cada vez más eficientes en sus dominios específicos.

RAG: El Primer Agente

RAG (Retrieval-Augmented Generation) fue realmente el **primer agente de IA** ampliamente adoptado:

1.  **Percibe:** Recibe una consulta del usuario
2.  **Razona:** Decide qué información necesita buscar
3.  **Actúa:** Busca información relevante en una base de conocimiento
4.  **Responde:** Genera una respuesta basada en la información recuperada


 **Evolución natural:** La necesidad de conectar agentes a **más entornos** (APIs, bases de datos, herramientas) ha llevado al desarrollo de frameworks especializados para **orquestración compleja**.

La Necesidad de Más Entornos

El éxito de **RAG** reveló una verdad fundamental: la información no vive en un solo lugar. En el mundo empresarial real, los datos críticos están dispersos como islas en un océano digital: algunos en CRMs como **Salesforce**, otros en bases de datos **MySQL**, documentos en **Google Drive**, conversaciones en **Slack**, y métricas en dashboards de **AWS**.

Los usuarios comenzaron a preguntarse: "Si mi agente puede buscar en documentos, ¿por qué no puede también revisar mi calendario, enviar emails, o analizar las métricas de ventas del último trimestre?"

Esta demanda natural llevó a una explosión de innovación en frameworks especializados.

 **El desafío:** Mantener contexto y coherencia mientras se navegan múltiples fuentes de datos y servicios externos.



Frameworks Principales para Agentes

Categorías por especialidad:

Agentes Simples:

-  **LangChain:** Framework maduro para aplicaciones LLM completas

Sistemas Multi-Agente:

-  **OpenAI Swarm:** Experimental, handoffs ligeros
-  **CrewAI:** Equipos colaborativos con roles especializados

Workflows Avanzados:

-  **LangGraph:** Estados, memoria y human-in-the-loop

Exploraremos cada uno en detalle...



LangChain: Agente Simple

El framework más completo para aplicaciones LLM con cientos de integraciones.

 Documentación: python.langchain.com

```
from langchain.agents import create_openai_functions_agent
from langchain.tools import DuckDuckGoSearchRun

# Configurar herramientas disponibles
search = DuckDuckGoSearchRun()
tools = [search]




# Crear agente con herramientas
agent = create_openai_functions_agent(
    llm=ChatOpenAI(),
    tools=tools,
    prompt="You are a helpful research assistant"
)

# El agente decide automáticamente cuándo usar cada herramienta
```





Sistemas Multi-Agente: El Desafío

Los sistemas multi-agente permiten **colaboración especializada**, pero traen nuevos desafíos:

  Lo que funciona bien:

-  **Flujos secuenciales:** Agente A → Agente B → Agente C
-  **Especialización:** Cada agente tiene un rol específico
-  **Handoffs simples:** Transferencia de contexto directa

  Lo que es complejo:

-  **Interacciones multidimensionales:** Múltiples agentes trabajando simultáneamente
-  **Decisiones conflictivas:** ¿Qué pasa si dos agentes dan respuestas diferentes?
-  **Sincronización:** Coordinar timing entre agentes independientes

CrewAI: Equipos Colaborativos - Parte 1

Framework independiente para equipos de agentes con roles especializados.

 Documentación: docs.crewai.com

```
from crewai import Agent, Task, Crew

# Crear agentes especializados con roles claros
researcher = Agent(
    role='Senior Researcher',
    goal='Find and analyze relevant information',
    backstory='Expert in data research with 10 years experience',
    verbose=True
)

writer = Agent(
    role='Content Writer',
    goal='Create compelling and accurate content',
    backstory='Skilled writer specialized in technical content',
    verbose=True
)
```

CrewAI: Equipos Colaborativos - Parte 2

```
# Continúa del slide anterior...

# Definir tareas colaborativas
research_task = Task(
    description='Research the latest AI trends in 2024',
    agent=researcher,
    expected_output='Detailed report with key findings'
)

writing_task = Task(
    description='Write article based on research findings',
    agent=writer,
    expected_output='Well-structured article',
    context=[research_task] # Depende del research
)






# Formar equipo y ejecutar
crew = Crew(
    agents=[researcher, writer],
    tasks=[research_task, writing_task],
    verbose=2
)

result = crew.kickoff() # ¡Trabajo colaborativo!
```

LangGraph: Workflows Avanzados





LangGraph es el framework más sofisticado para agentes que requieren estados persistentes y flujos complejos.

¿Por qué es más avanzado?

-  **Memoria persistente:** Los agentes recuerdan conversaciones y decisiones anteriores
-  **Estados complejos:** Maneja múltiples variables que evolucionan durante el workflow
-  **Human-in-the-loop:** Pausa para obtener aprobación humana antes de acciones críticas
-  **Grafos de decisión:** Flujos condicionales basados en resultados anteriores
-  **Streaming nativo:** Muestra el proceso de razonamiento en tiempo real

LangGraph: Workflows Avanzados

Componentes principales:

-  **Nodes:** Funciones que procesan y transforman el estado
-  **Edges:** Conexiones que definen el flujo entre nodos
-  **State:** Objeto persistente que mantiene toda la información
-  **Conditional Edges:** Decisiones dinámicas basadas en el estado actual

 Documentación: langchain.com/langgraph



LangGraph: Configuración y Estado

```
from langgraph.graph import StateGraph
from typing import TypedDict

# Definir el estado que persiste entre nodos
class AgentState(TypedDict):
    messages: list
    research_data: str
    analysis_result: str
    user_approved: bool

# Función de investigación
def research_node(state: AgentState):
    # Simular investigación
    research_data = "AI market growing 40% yearly"

    return {
        **state,
        "research_data": research_data,
        "messages": state["messages"] + ["Research completed"]
    }

# Función de análisis que usa datos del estado
def analysis_node(state: AgentState):
    research = state["research_data"] # Usa estado anterior
    analysis = f"Based on: {research}, recommend investment"

    return {
        **state,
        "analysis_result": analysis,
        "messages": state["messages"] + ["Analysis completed"]
    }
```



LangGraph: Workflow y Ejecución

```
# Continúa del slide anterior...

# Función condicional para decisiones dinámicas
def should_continue(state: AgentState):
    if state.get("user_approved"):
        return "execute"
    else:
        return "wait_approval"

# Construir el grafo de workflow
workflow = StateGraph(AgentState)

# Agregar nodos al grafo
workflow.add_node("research", research_node)
workflow.add_node("analysis", analysis_node)
workflow.add_node("wait_approval", lambda s: s) # Pausa humana




# Definir flujo con decisiones condicionales
workflow.add_edge("research", "analysis")
workflow.add_conditional_edges(
    "analysis",
    should_continue, # Función que decide el próximo paso
    {
        "execute": "END", # Terminar workflow
        "wait_approval": "analysis" # Esperar aprobación
    }
)

# Compilar y ejecutar
app = workflow.compile()
result = app.invoke({"messages": [], "user_approved": False})
```

El Problema de Consenso

Cuando **múltiples agentes** analizan la misma información, pueden llegar a **conclusiones diferentes**:

Ejemplo práctico:

-  Agente de Investigación: "La empresa debería invertir en IA"
-  Agente Financiero: "Los costos son demasiado altos"
-  Agente de Datos: "Los datos son insuficientes para decidir"

? ¿Quién tiene razón? ¿Cómo decidir?

 Solución tradicional: Un agente árbitro que evalúa y decide.

 Problema: ¿Quién valida al árbitro? ¿Qué pasa si el árbitro está mal configurado?



Sistemas de Arbitraje





Los sistemas de arbitraje son como jueces que resuelven conflictos entre agentes:



Arquitectura típica:








Estrategias de arbitraje:

-  **Votación por mayoría:** La respuesta más común gana
-  **Ponderación por confianza:** Agentes más "confiables" tienen mayor peso
-  **Especialización contextual:** El agente más experto en el tema decide
-  **Agregación estadística:** Promedios, medianas de respuestas numéricas

Sistemas de Disputas: Inspiración Blockchain






Debido a las limitaciones del arbitraje tradicional, han tomado fuerza los **sistemas de disputas descentralizados**. Son inspirados en conceptos de **blockchain**.


Principios clave de blockchain:

-  **Descentralización:** Múltiples validadores independientes
-  **Proof-of-Stake:** Los validadores tienen "stake" (incentivos económicos)
-  **Consenso:** Acuerdo de la mayoría para validar transacciones
-  **Disputas:** Mecanismo para desafiar decisiones incorrectas
-  **Penalización:** Validadores maliciosos pierden su stake

 **Aplicado a agentes:** Múltiples agentes-validadores verifican decisiones, con incentivos para actuar correctamente y penalizaciones para comportamiento malicioso.

Visión 2025-2030: Agentes Descentralizados

-  **Redes de agentes:** Miles de agentes colaborando globalmente
-  **Consenso algorítmico:** Sistemas de disputas automatizados
-  **Economías de agentes:** Tokens e incentivos reales
-  **Resistencia bizantina:** Tolerancia a agentes maliciosos
-  **Auditoría transparente:** Todas las decisiones son verificables

 **Meta final:** Agentes que pueden **auto-organizarse**, **auto-validarse** y **auto-corregirse** sin intervención humana centralizada, manteniendo alta confiabilidad y transparencia.

Recursos Adicionales

Frameworks de Agentes

Agentes Simples:

- LangChain → python.langchain.com
- LlamaIndex → docs.llamaindex.ai

Multi-Agente:

- CrewAI → docs.crewai.com
- OpenAI Swarm → github.com/openai/swarm
- AutoGen → microsoft.github.io/autogen

Recursos Adicionales

Workflows Avanzados:

- LangGraph → langchain.com/langgraph
- Agent-to-Agent (A2A) → [Blog Google](#)

Herramientas:

- AgentOps → agentops.ai
- LangSmith → langchain.com/langsmith