# RAG Portfolio

## 10 Proyectos de Retrieval-Augmented Generation

*De básico a avanzado — PostgreSQL + pgvector + NVIDIA NIM*

**Python 3.9** | **FastAPI** | **PostgreSQL 17** | **pgvector**

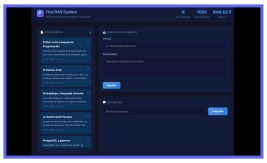
**NVIDIA NIM** | **LangChain** | **SSE Streaming** | **BeautifulSoup4**


## Manuel Argüelles

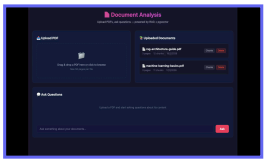
Data Engineer / Analytics Engineer

Lima, Perú

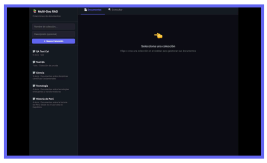

### Los 10 Proyectos



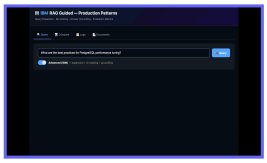
01 Basic



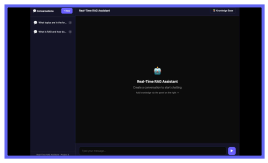
02 PDF



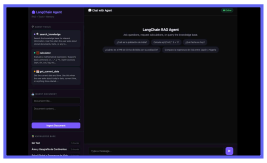
03 Multi-doc



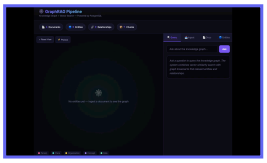
04 IBM



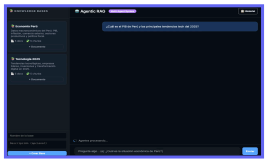
05 Real-time



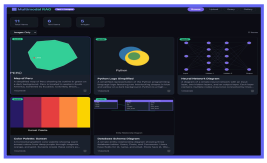
06 LangChain



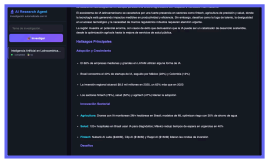
07 GraphRAG



08 Agentic



09 Multimodal



10 Research

Febrero 2026

**github.com/manuelarguelles/rag-portfolio**

# Contenido

# ¿Qué es RAG?

RAG (Retrieval-Augmented Generation) combina búsqueda de información relevante en una base de conocimiento con generación de respuestas por un LLM. Esto permite responder con información específica y actualizada, reduciendo alucinaciones significativamente.

**Pipeline RAG Fundamental:**

```
Pregunta → Embedding (1024d) → Vector Search (pgvector)
       → Top-K Chunks → LLM (Kimi K2.5) → Respuesta Grounded
```

## Stack Tecnológico

### PostgreSQL 17 + pgvector

Base de datos vectorial con índice HNSW.
Búsqueda eficiente por similitud coseno.

### NVIDIA NIM API (gratis)

nv-embedqa-e5-v5 (1024d) para embeddings.
Kimi K2.5 como LLM de generación.

### Python 3.9 + FastAPI

Backend ligero con async. P06 usa Flask.
Sin frameworks pesados.

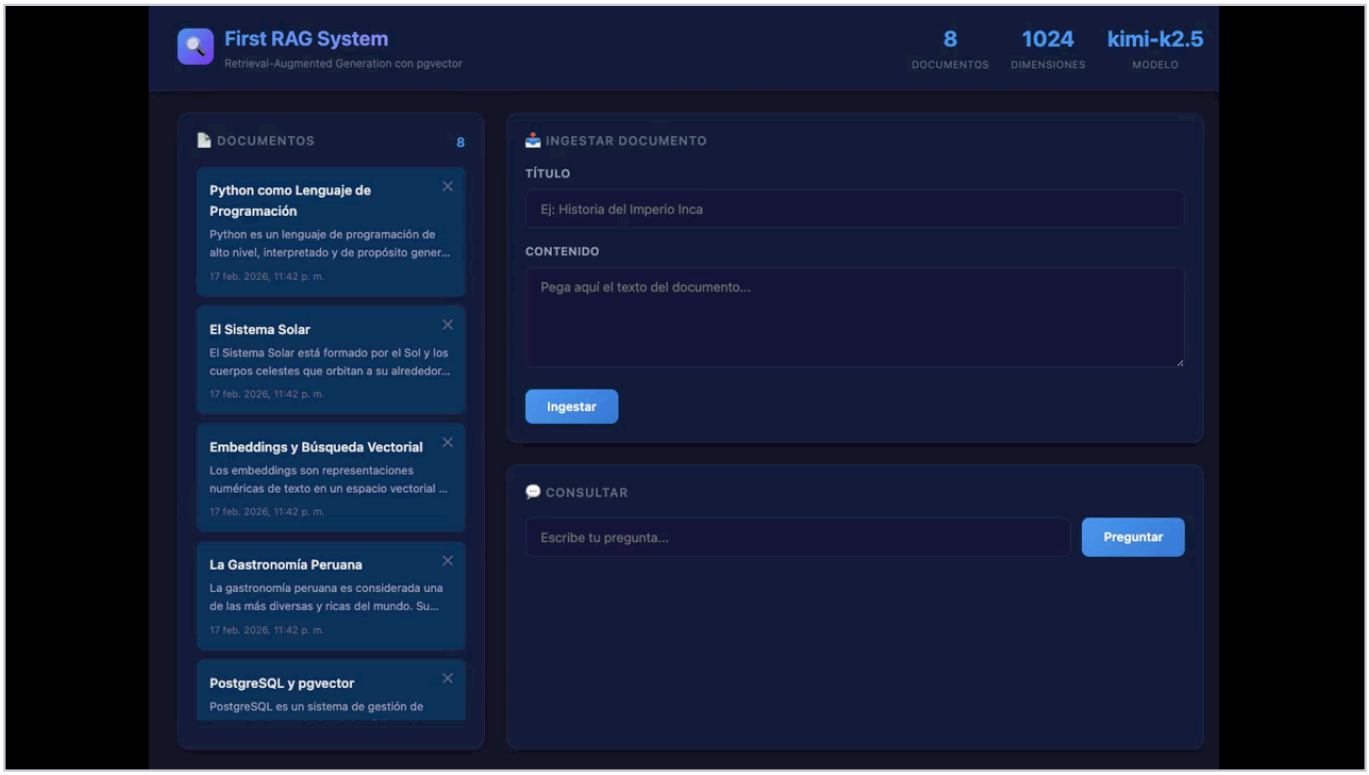
### HTML/CSS/JS vanilla

Frontend responsive con dark theme.
Cada proyecto independiente.

## Números Clave

| 10 | 1024 | 100% | 0 |
|---|---|---|---|
| Proyectos RAG | Dimensiones embedding | Open source | Costo de API |

# 01 — First RAG System




## ¿Qué es?

RAG básico construido desde cero, sin frameworks. Implementa el patrón fundamental: embedding de la pregunta → búsqueda por similitud coseno en pgvector → contexto al LLM → respuesta grounded con citas. Punto de partida ideal para entender RAG.

## Implementación

- Ingestión de 8 temas (Transformers, pgvector, RAG, SSE, etc.)
- Búsqueda vectorial con similarity scores visibles
- Generación con citas al chunk fuente
- UI tipo chat con panel de resultados

```
Stack: FastAPI + psycopg2 + pgvector  |  Puerto: 8000
```

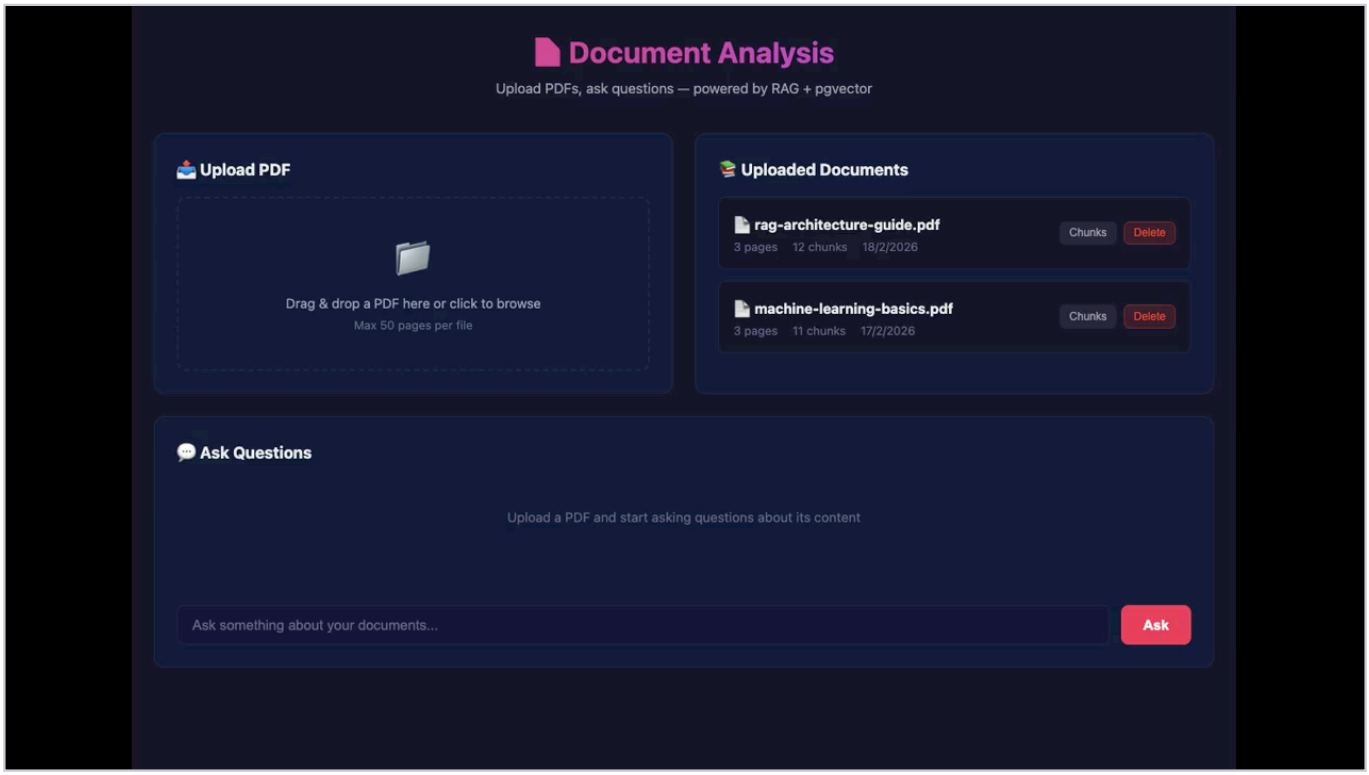
### ✓ Ventajas

+ Simple y didáctico
+ Sin frameworks
+ Código transparente

### ✗ Limitaciones

- Sin chunking inteligente
- Sin reranking
- Sin memoria

# 02 — Document Analysis




## ¿Qué es?

RAG con procesamiento real de PDFs. Upload → extracción con PyMuPDF → chunking 500 chars / 100 overlap → embedding → búsqueda con citas a la página exacta del PDF.

## Implementación

- Pipeline: drag & drop → extracción → chunking → embedding
- Trazabilidad hasta página exacta del PDF
- Chunking con overlap preserva contexto
- 3 PDFs de ejemplo precargados

```
Stack: FastAPI + PyMuPDF + pgvector  |  Puerto: 8002
```

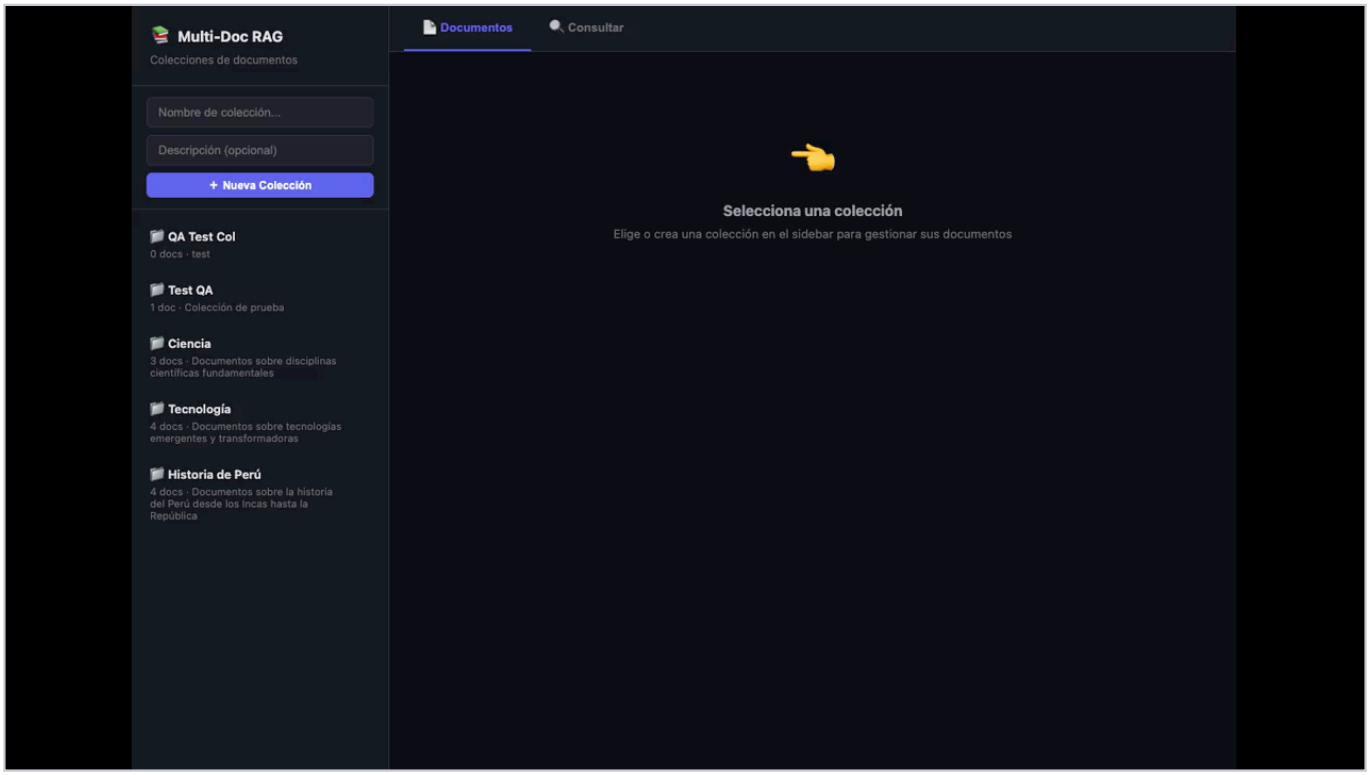
### ✓ Ventajas

+ Procesamiento real de docs
+ Trazabilidad doc→pág→chunk
+ Overlap preserva contexto

### ✗ Limitaciones

- Solo PDF
- Sin multi-documento
- Chunking fijo

# 03 — Multi-Document RAG




## ¿Qué es?

RAG con múltiples documentos en colecciones temáticas. Búsqueda filtrable por colección o corpus completo. Trazabilidad de 3 niveles: colección → documento → chunk.

## Implementación

- CRUD de colecciones y documentos
- Búsqueda filtrable por colección
- 4 colecciones precargadas
- Trazabilidad de 3 niveles

```
Stack: FastAPI + psycopg2 + pgvector  |  Puerto: 8003
```

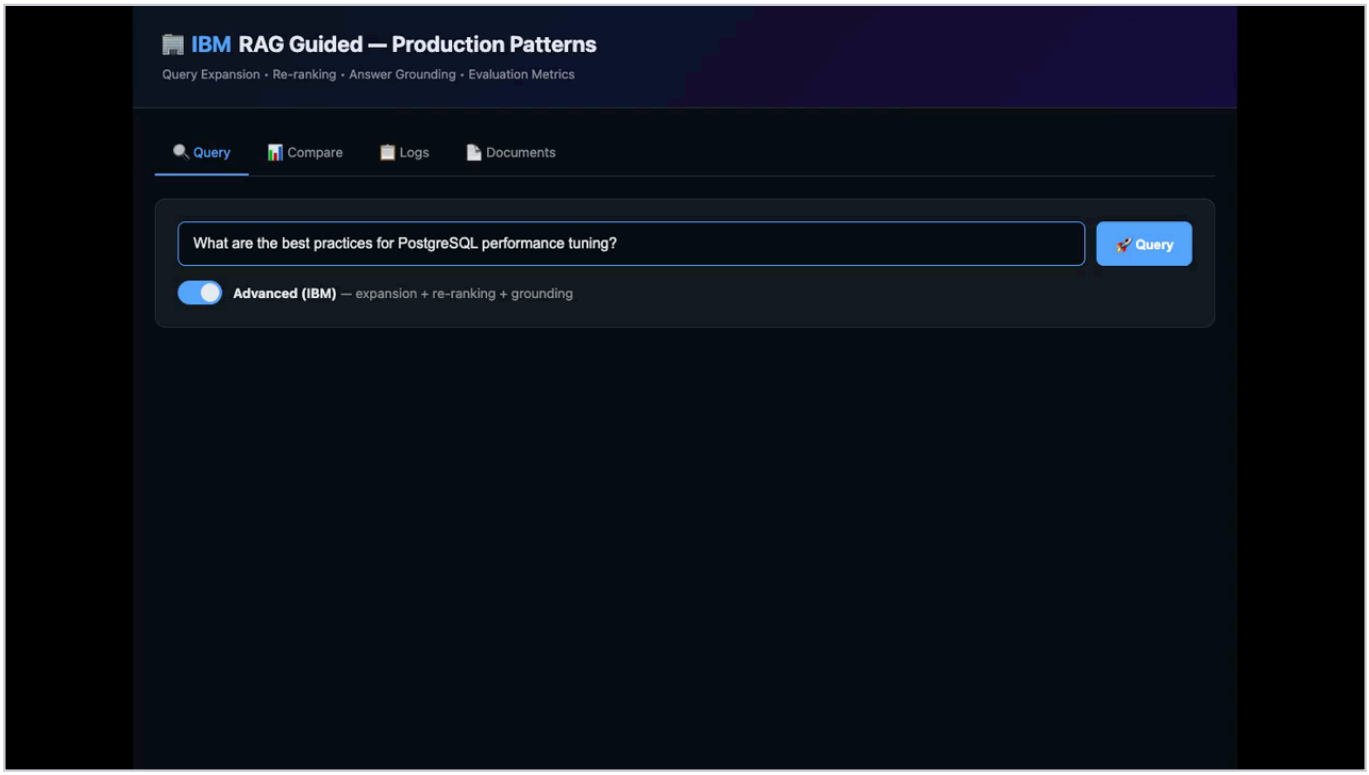
### ✓ Ventajas

+ Organización por colecciones
+ Búsqueda filtrada
+ Trazabilidad completa

### ✗ Limitaciones

- Sin reranking
- Sin cross-collection search

# 04 — IBM RAG Guided




## ¿Qué es?

Pipeline enterprise: Query Expansion (2-3 variantes), Re-ranking (LLM puntúa 1-10), Grounding Check (verifica fidelidad). Toggle entre modo Simple y Advanced.

## Implementación

- Dos pipelines: Simple vs Advanced
- Logs visibles de cada etapa del pipeline
- Métricas de latencia por paso
- Grounding score de fidelidad

```
Stack: FastAPI + pgvector (4-5 LLM calls)  |  Puerto: 8004
```

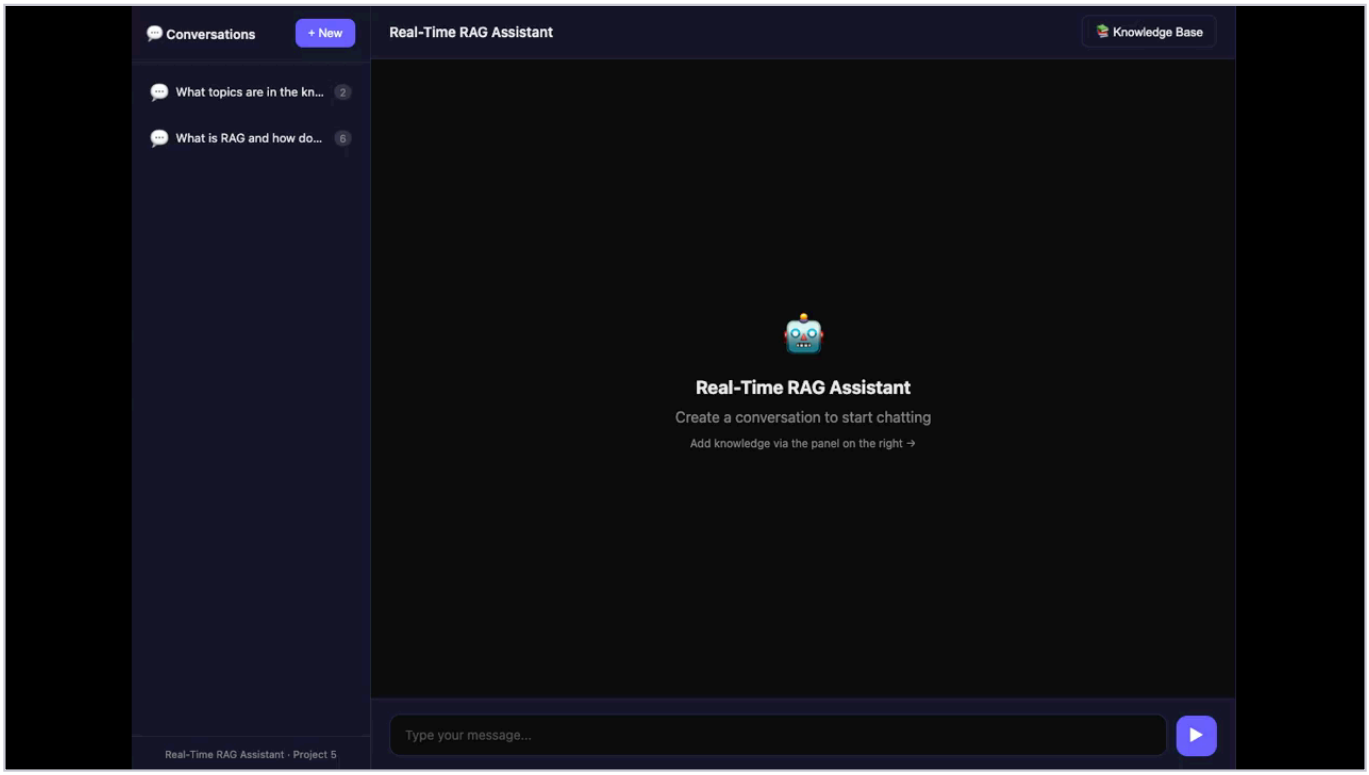
### ✓ Ventajas

+ Calidad superior
+ Multi-ángulo
+ Reduce alucinaciones

### ✗ Limitaciones

- Lento (~60-90s)
- Costoso (N LLM calls)
- Debug complejo

# 05 — Real-Time Assistant




## ¿Qué es?

Asistente con streaming SSE token-by-token como ChatGPT. Memoria de conversación (5 msgs). Knowledge base editable en caliente sin reiniciar.

## Implementación

- Chat UI estilo ChatGPT con sidebar
- Streaming vía Server-Sent Events
- Memoria conversacional (5 msgs)
- KB editable en caliente

```
Stack: FastAPI + SSE + pgvector  |  Puerto: 8005
```

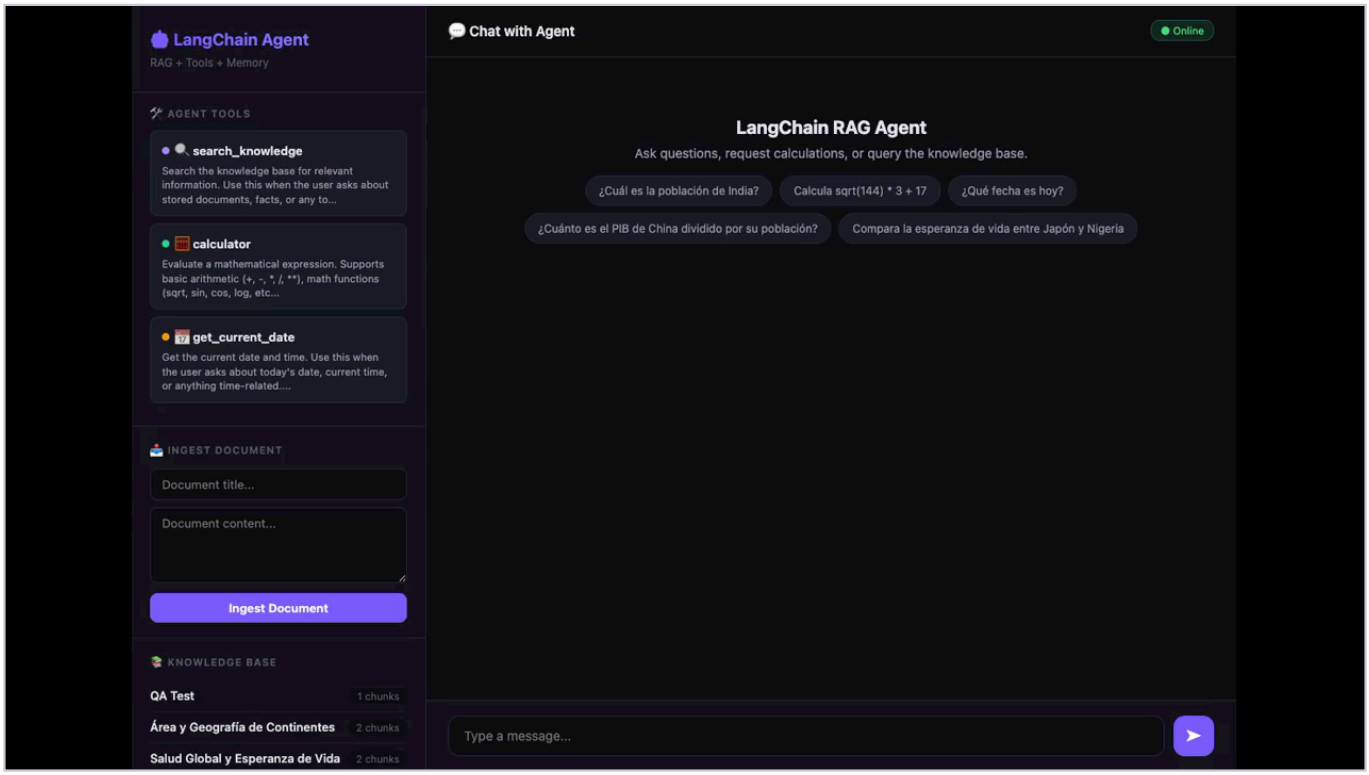
### ✓ Ventajas

+ UX superior (streaming)
+ Memoria conversacional
+ KB en caliente

### ✗ Limitaciones

- Solo texto directo
- Memoria limitada

# 06 — LangChain Agent




## ¿Qué es?

Agente ReAct con LangChain + LangGraph. Decide autónomamente entre 3 tools: búsqueda vectorial, calculadora, y fecha actual. Custom NvidiaEmbeddings.

## Implementación

- 3 tools: search, calculator, date
- NvidiaEmbeddings custom (asimétrico)
- Pipeline de ingestión LangChain
- 6 documentos precargados

```
Stack: Flask + LangChain + LangGraph  |  Puerto: 5006
```

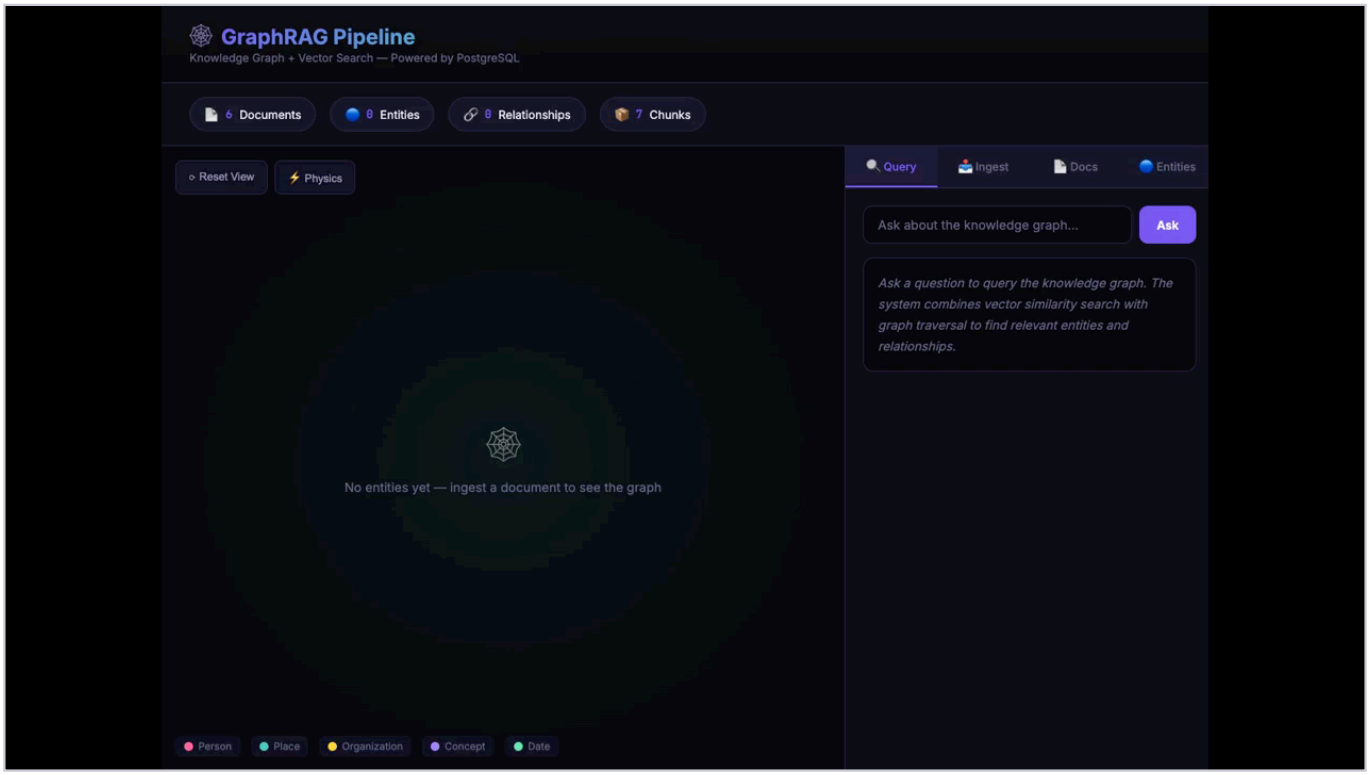
### ✓ Ventajas

+ Framework maduro
+ Tool selection
+ Patrón ReAct

### ✗ Limitaciones

- Vendor lock-in
- Overhead
- Debug difícil

# 07 — GraphRAG




## ¿Qué es?

Híbrido: vector search + knowledge graph. Extrae entidades y relaciones vía LLM. Multi-hop reasoning implementado en PostgreSQL puro (sin Neo4j).

## Implementación

- Extracción de entidades y relaciones
- Grafo en PostgreSQL (sin Neo4j/Docker)
- Búsqueda híbrida: cosine + graph JOINs
- 6 documentos precargados

```
Stack: FastAPI + pgvector + SQL JOINs  |  Puerto: 5007
```

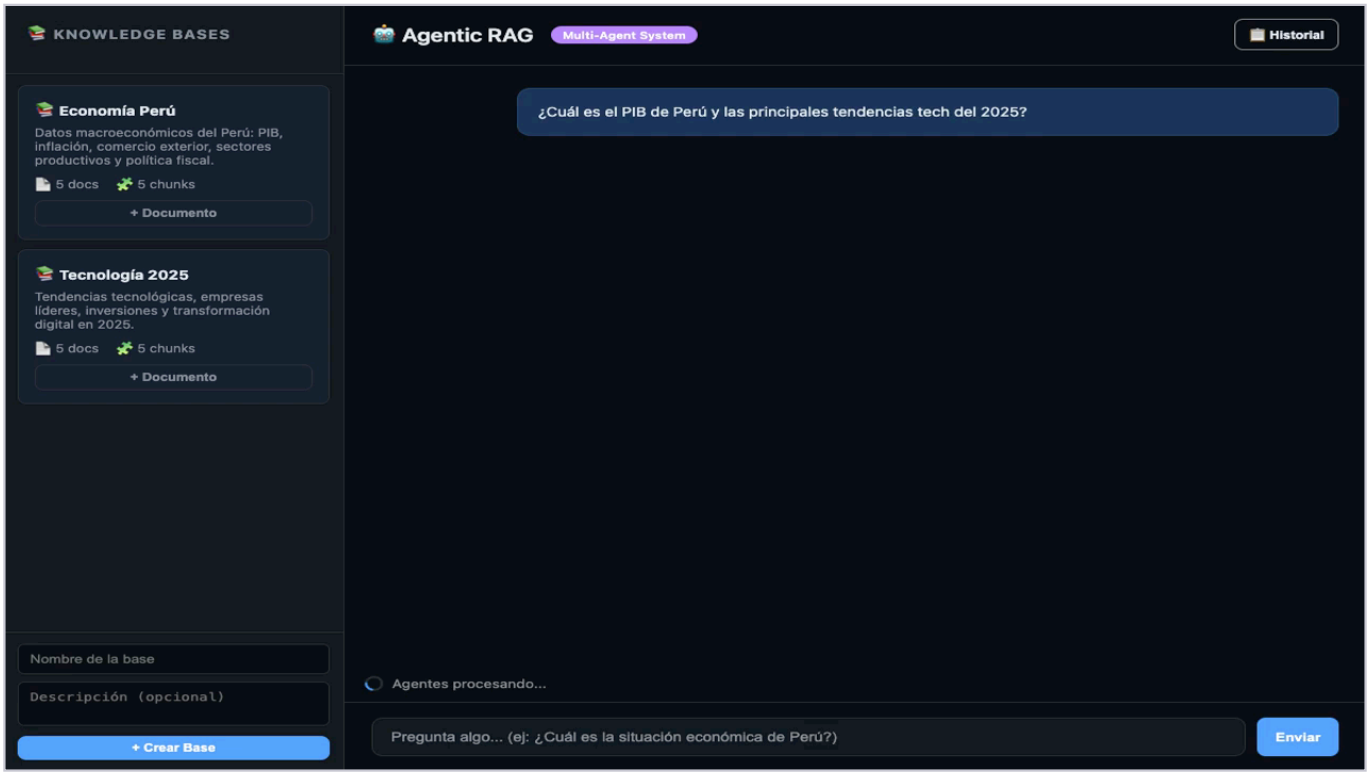
### ✓ Ventajas

+ Multi-hop reasoning
+ Todo en PostgreSQL
+ Enriquece contexto

### ✗ Limitaciones

- Extracción lenta
- Depende del LLM
- Más complejo

# 08 — Agentic RAG




## ¿Qué es?

4 agentes: Router (planifica), Research (busca), Analyst (analiza), Writer (compone). Python puro — sin LangChain, sin CrewAI. Trace visible de cada decisión.

## Implementación

- 4 agentes especializados
- 2 knowledge bases temáticas
- Panel de trace en tiempo real
- Historial de queries

```
Stack: FastAPI + pgvector (Python puro)  |  Puerto: 8008
```

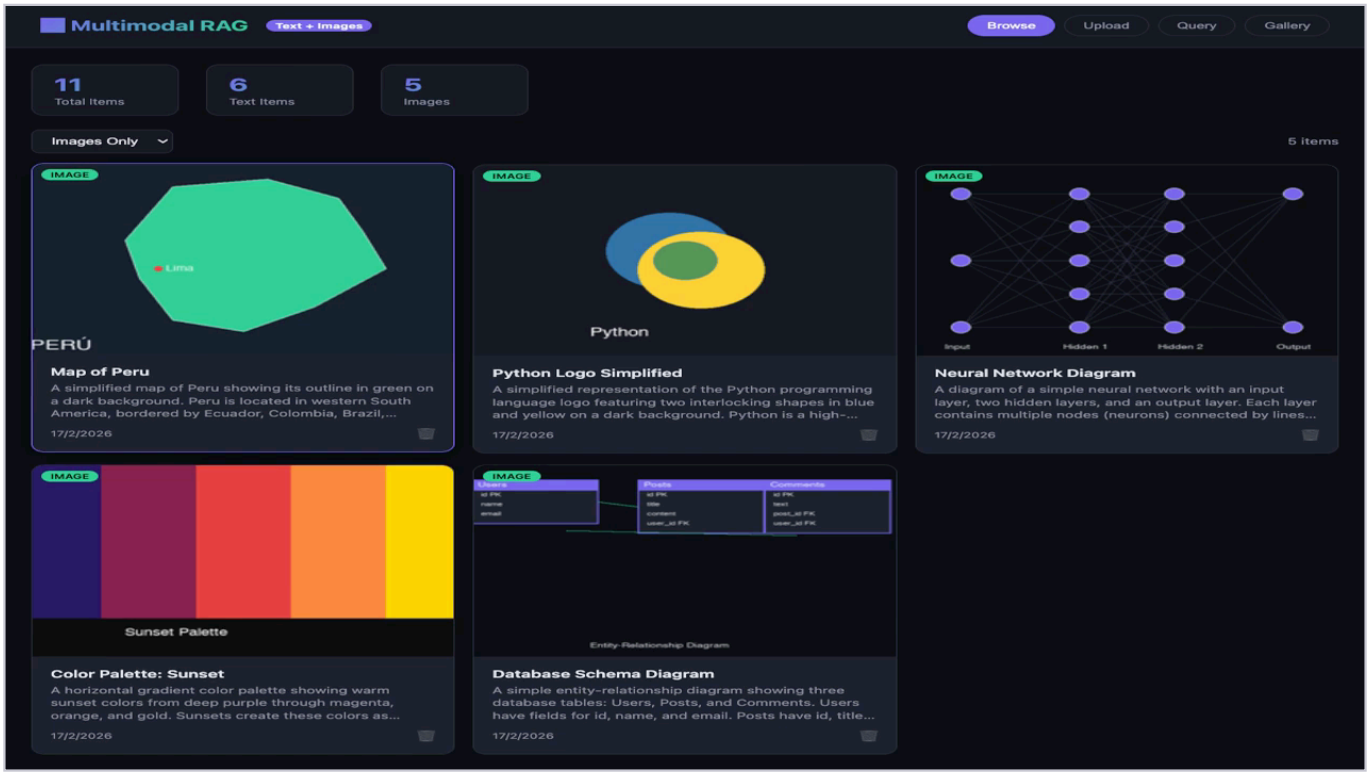
### ✓ Ventajas

+ Divide y vencerás
+ Trace visible
+ Sin frameworks

### ✗ Limitaciones

- 4+ LLM calls (~90s)
- Costoso en tokens
- Orquestación compleja

# 09 — Multimodal RAG




## ¿Qué es?

Unifica texto e imágenes en un espacio vectorial. Imágenes indexadas via descripción textual (text-bridge). Galería con filtros y búsqueda semántica unificada.

## Implementación

- 11 items: 6 textos + 5 imágenes
- 4 vistas: Browse, Upload, Query, Gallery
- Filtro por tipo (All/Text/Images)
- Imágenes en BYTEA (PostgreSQL)

```
Stack: FastAPI + pgvector (text-bridge)  |  Puerto: 5009
```

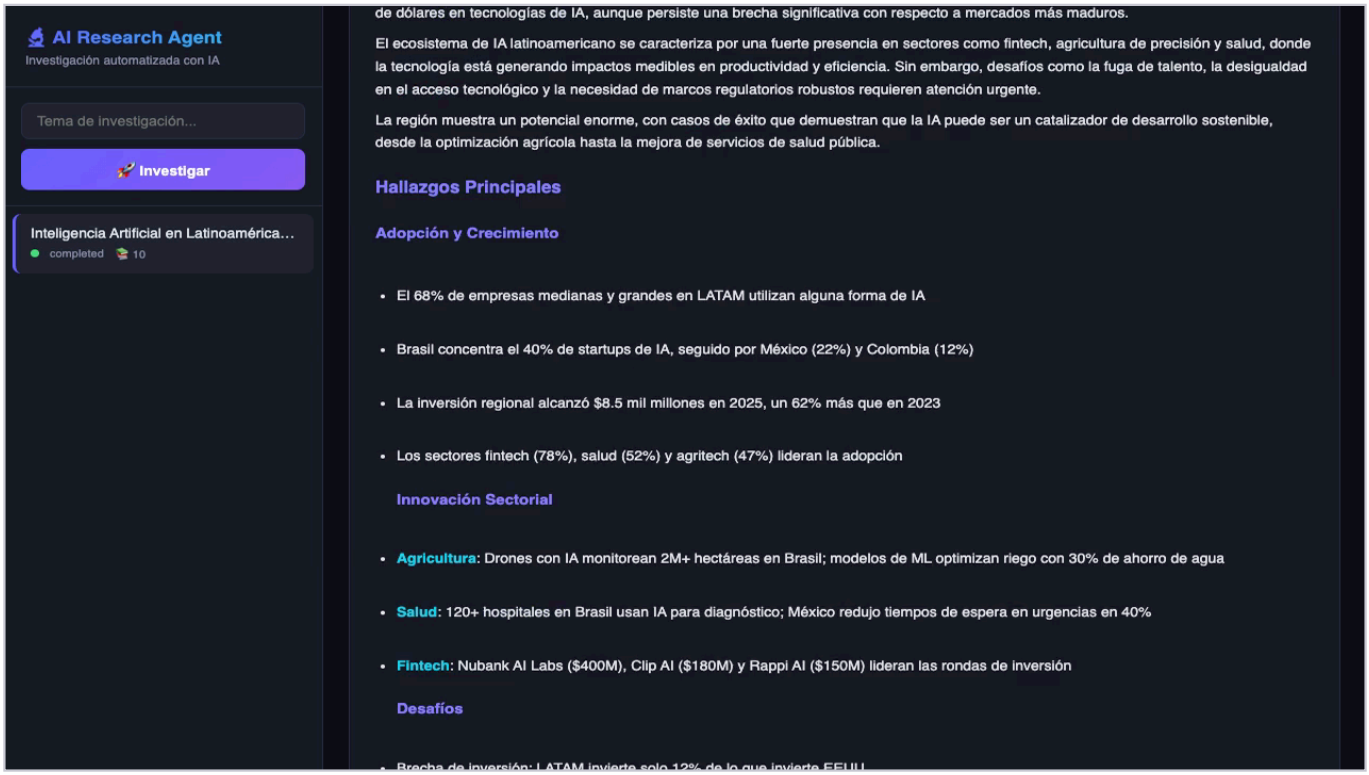
### ✓ Ventajas

+ Búsqueda unificada
+ UI tipo galería
+ Sin modelo de visión

### ✗ Limitaciones

- No embedding visual
- Depende de descripción
- Sin image-to-image

# 10 — AI Research Agent




## ¿Qué es?

Investigación automatizada: genera queries → scrapea web → chunkea → embede → analiza → genera reporte con resumen ejecutivo y hallazgos categorizados.

## Implementación

- Pipeline completo de investigación
- Demo: 'IA en Latinoamérica 2025'
- 5 hallazgos categorizados
- Reporte Markdown estructurado

```
Stack: FastAPI + httpx + BS4 + pgvector  |  Puerto: 8010
```

### ✓ Ventajas

+ End-to-end automático
+ Hallazgos categorizados
+ Reporte exportable

### ✗ Limitaciones

- Depende de scraping
- Fuentes variables
- Tarda varios minutos

# Resumen Comparativo

Cada proyecto agrega complejidad sobre el anterior. El RAG ideal depende del caso de uso — no siempre más complejo es mejor.

## Progresión de Complejidad


## ¿Cuál usar según el caso?

| | |
|---|---|
| Aprender RAG desde cero | → **01 First RAG** |
| Documentos internos (PDFs) | → **02 Document Analysis** |
| Múltiples fuentes organizadas | → **03 Multi-Document** |
| Producción enterprise | → **04 IBM RAG Guided** |
| Chatbot conversacional | → **05 Real-Time Assistant** |
| Múltiples herramientas | → **06 LangChain Agent** |
| Preguntas sobre relaciones | → **07 GraphRAG** |
| Consultas complejas | → **08 Agentic RAG** |
| Contenido mixto (text+images) | → **09 Multimodal RAG** |
| Investigación autónoma | → **10 Research Agent** |


## Manuel Argüelles

Data Engineer / Analytics Engineer • Lima, Perú

**GitHub: manuelarguelles • LinkedIn: manuelarguelles**