RAG Portfolio

10 Proyectos de Retrieval-Augmented Generation

De básico a avanzado • PostgreSQL + pgvector + NVIDIA NIM

Manuel Argüelles

Data Engineer / Analytics Engineer

Lima, Perú

FastAPI | PostgreSQL 17 | pgvector | NVIDIA NIM | LangChain | 3. | S

Febrero 2026

github.com/manuelarguelles/rag-portfolio

Contenido

¿Qué es RAG?

RAG (Retrieval-Augmented Generation) es un patrón de arquitectura que 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.

Este portafolio implementa 10 variantes progresivas del patrón RAG, desde la versión más básica hasta sistemas multi-agente y multimodales.

```
Pregunta del usuario
  -> Embedding -> Vector Search -> Top-K Chunks -> LLM -> Respuesta
     (1024d)     (pgvector)     (contexto)    (Kimi K2.5)  (grounded)
```

Stack Tecnológico

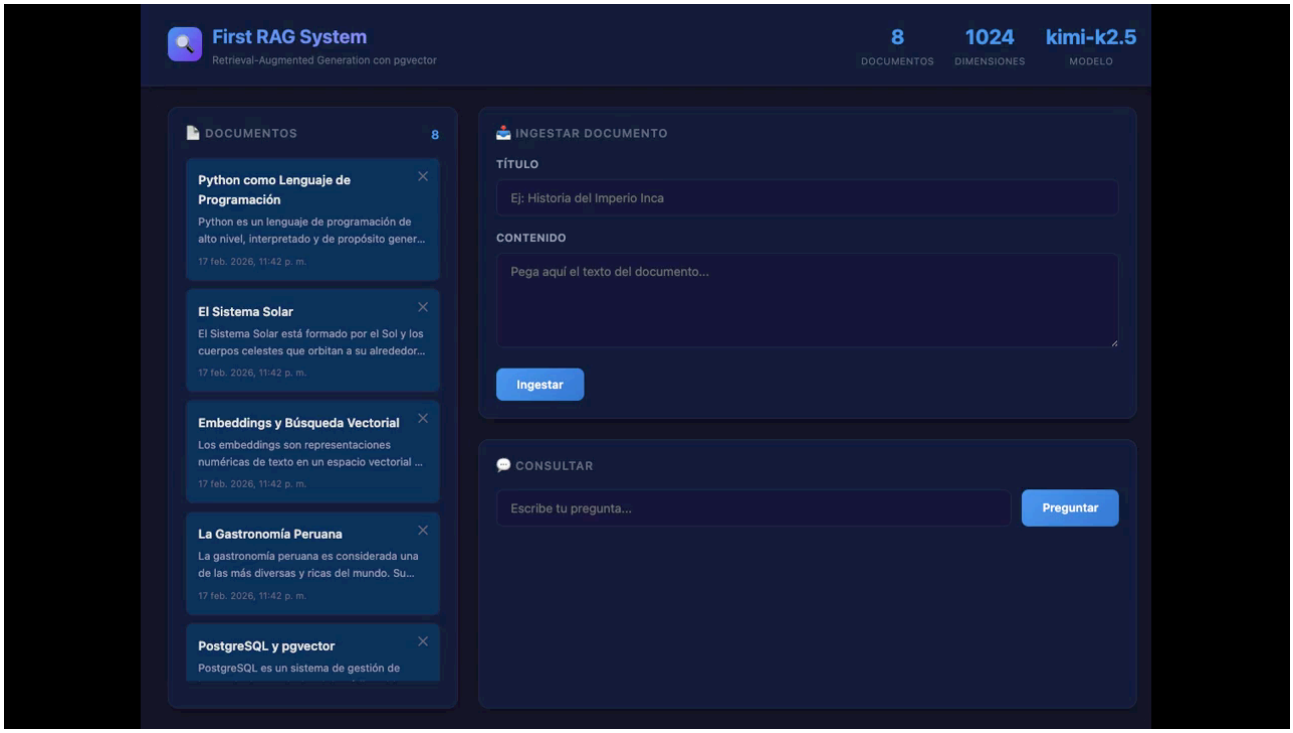
> **PostgreSQL 17 + pgvector 0.8.1**
Base de datos vectorial con índice HNSW para búsqueda eficiente por similitud coseno

> **NVIDIA NIM API (gratis)**
nv-embedqa-e5-v5 para embeddings (1024 dimensiones) + Kimi K2.5 como LLM de generación

> **Python 3.9 + FastAPI**
Backend ligero y rápido con async support. Proyecto 06 usa Flask + LangChain

> **HTML/CSS/JS vanilla**
Frontend responsivo con dark theme. Sin React ni frameworks pesados

01 — First RAG System



¿Qué es?

RAG básico construido desde cero, sin frameworks. Implementa el patrón fundamental: embedding → búsqueda vectorial → contexto al LLM → respuesta grounded con citas.

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

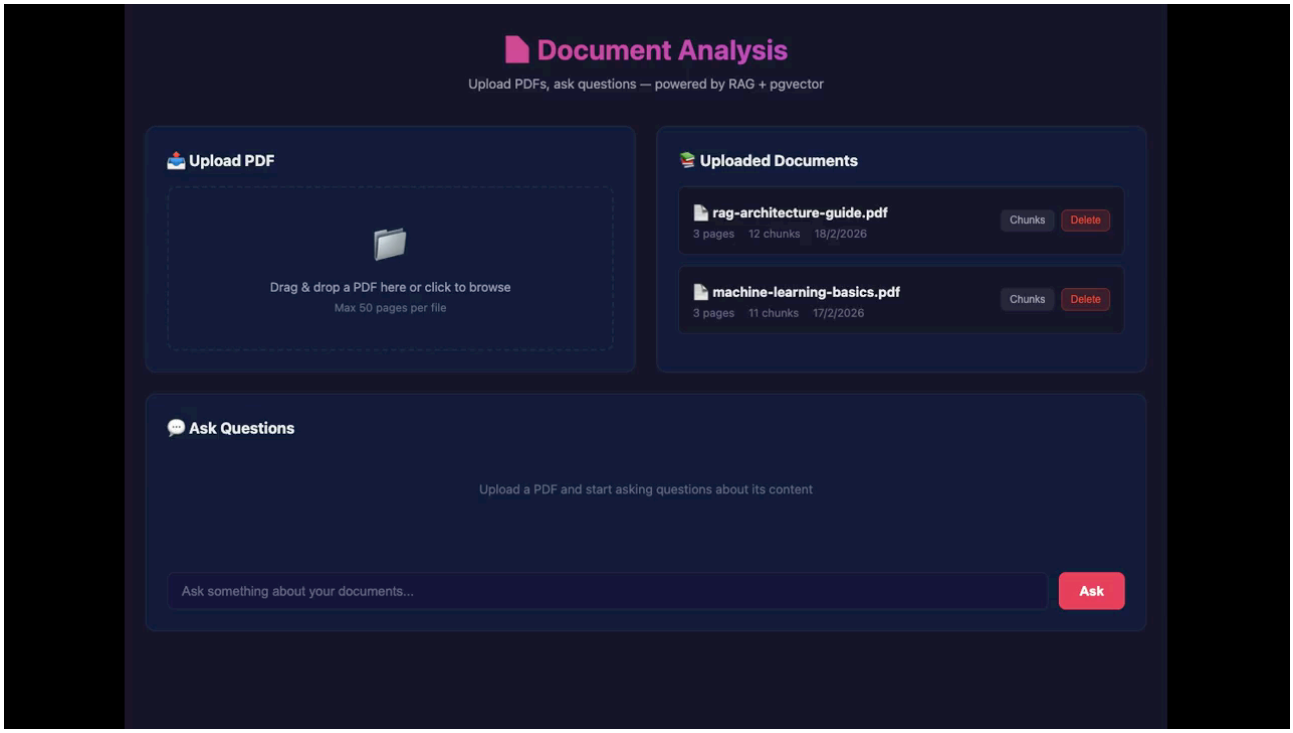
Ventajas

+ 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 500c/100 overlap → embedding → búsqueda con citas a página exacta.

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

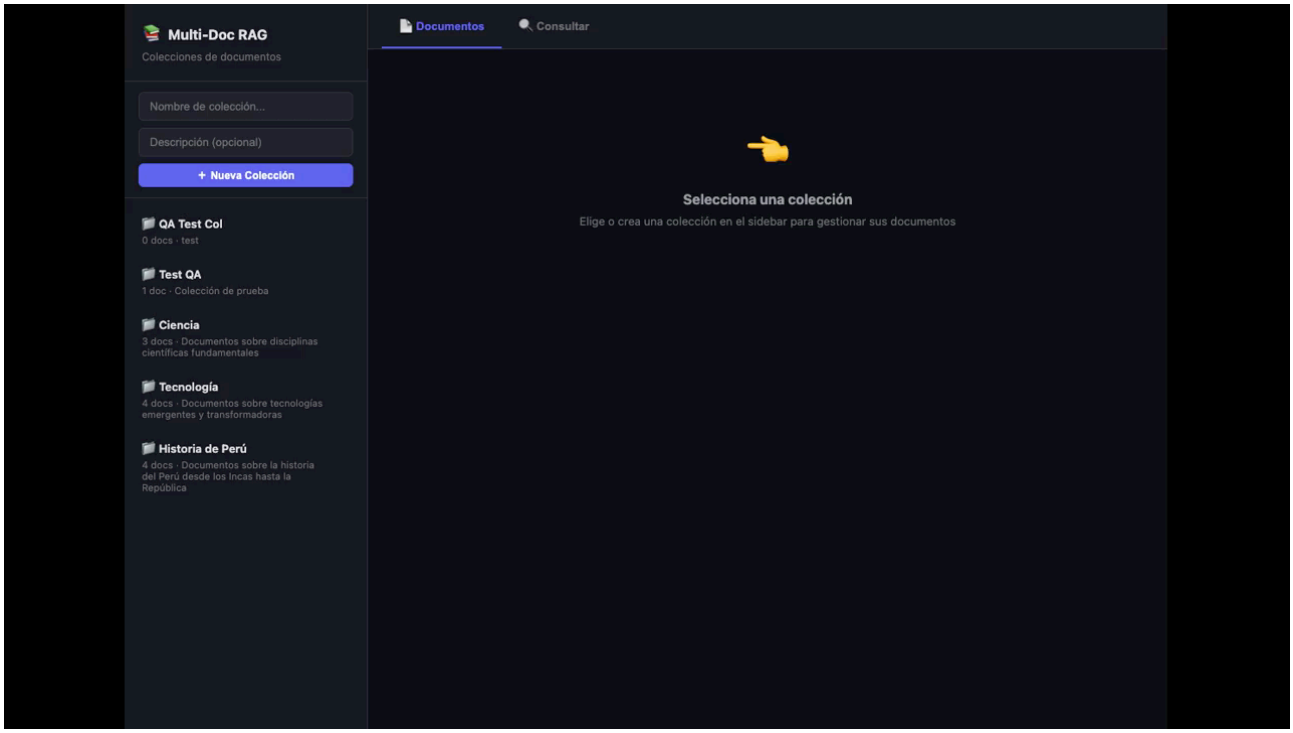
Ventajas

+ Procesamiento real de docs
+ Trazabilidad doc→página→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: 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

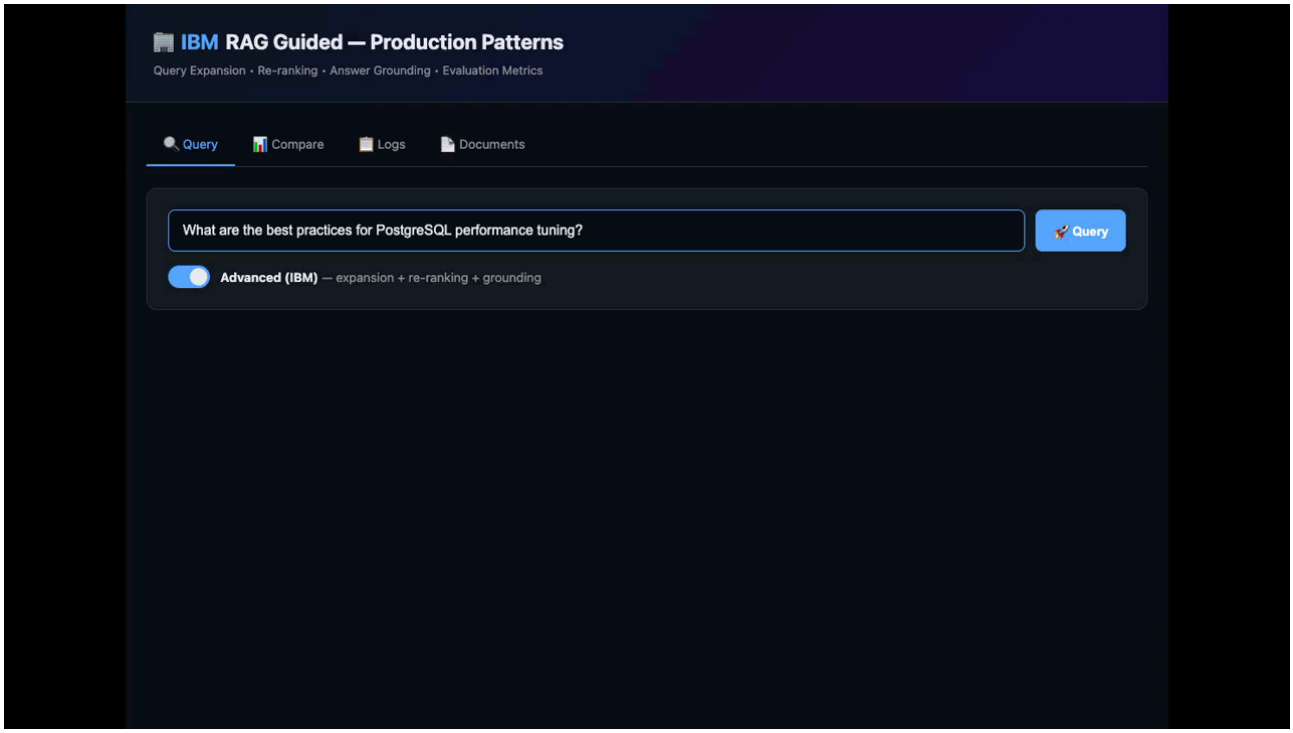
Ventajas

+ 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). Dos modos: Simple y Advanced.

Implementación

> Pipeline Simple y Advanced con toggle
> Logs visibles de cada etapa
> Métricas de latencia por paso
> Grounding score de fidelidad

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

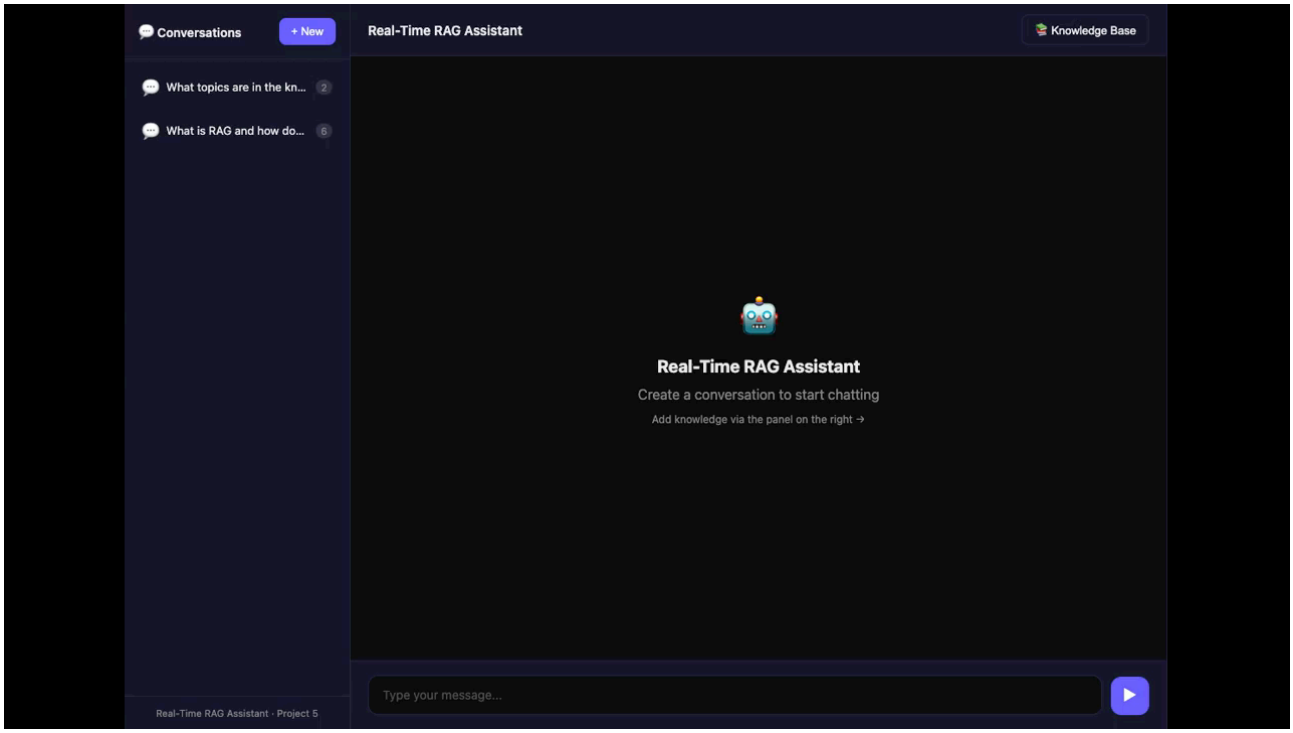
Ventajas

+ Calidad superior
+ Query expansion multi-ángulo
+ Reduce alucinaciones

Limitaciones

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

05 — Real-Time Assistant



¿Qué es?

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

Implementación

> Chat UI estilo ChatGPT con sidebar
> Streaming vía Server-Sent Events
> Memoria conversacional
> KB editable en caliente

Stack: FastAPI + SSE + pgvector | Puerto: 8005

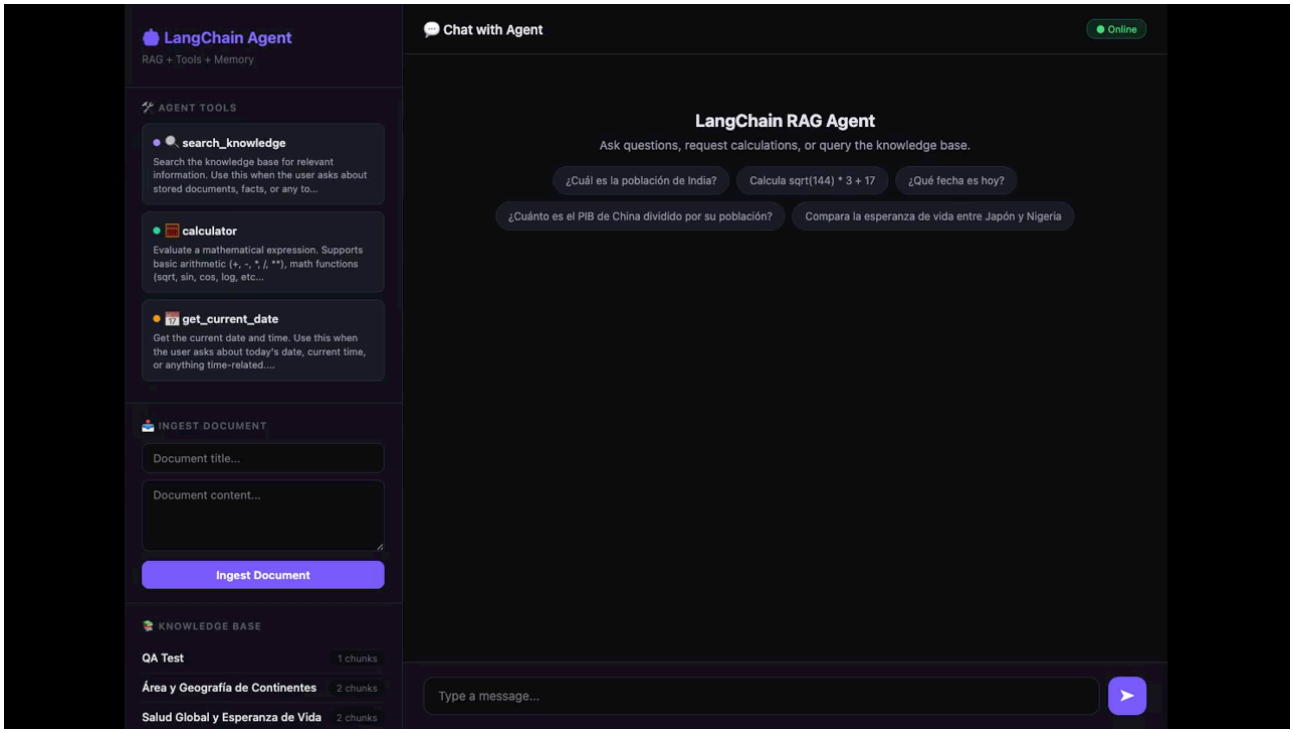
Ventajas

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

Limitaciones

- Solo texto directo
- Memoria limitada (5 msgs)

06 — LangChain Agent



¿Qué es?

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

Implementación

> 3 tools: search, calculator, date
> Custom NvidiaEmbeddings (modelo asimétrico)
> Pipeline de ingestión LangChain
> 6 documentos precargados

Stack: Flask + LangChain + LangGraph | Puerto: 5006

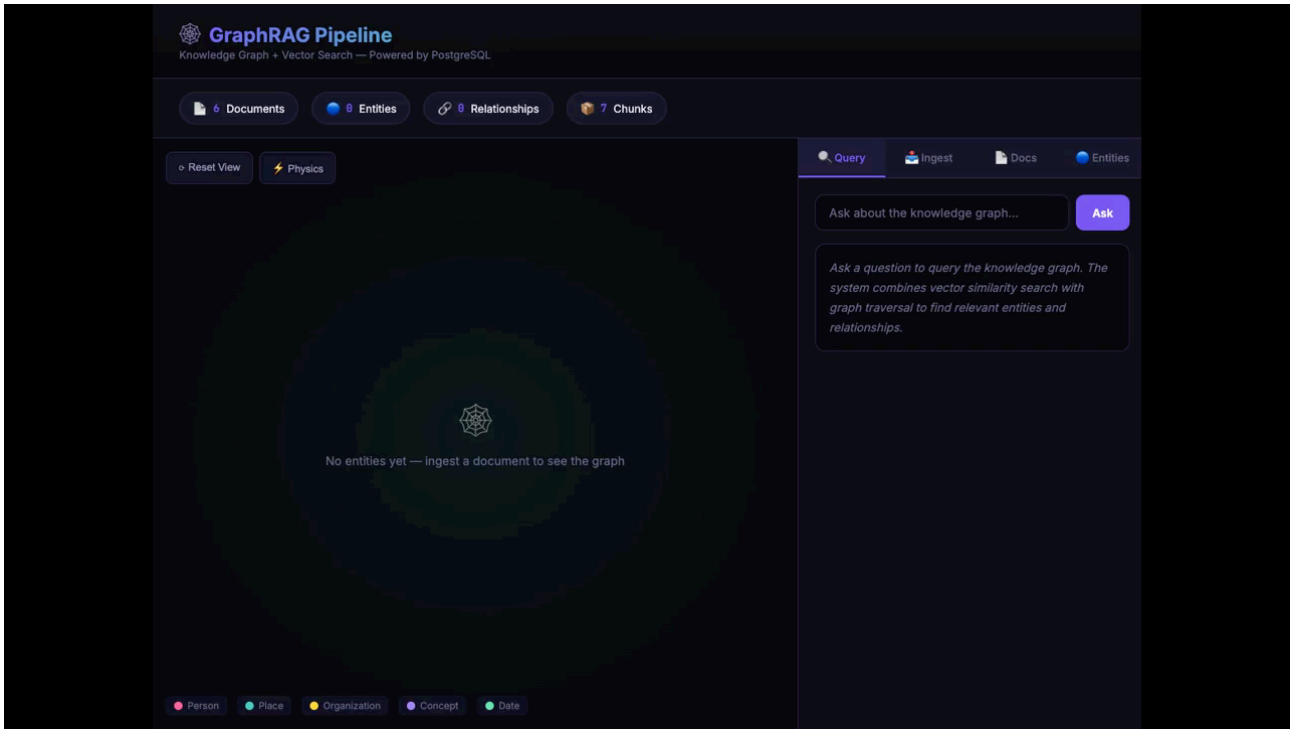
Ventajas

+ Framework maduro
+ Tool selection autónoma
+ Patrón ReAct

Limitaciones

- Vendor lock-in
- Overhead de abstracción
- Debug difícil

07 — GraphRAG



¿Qué es?

Híbrido: vector search + knowledge graph. Extrae entidades y relaciones vía LLM. Multi-hop reasoning 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

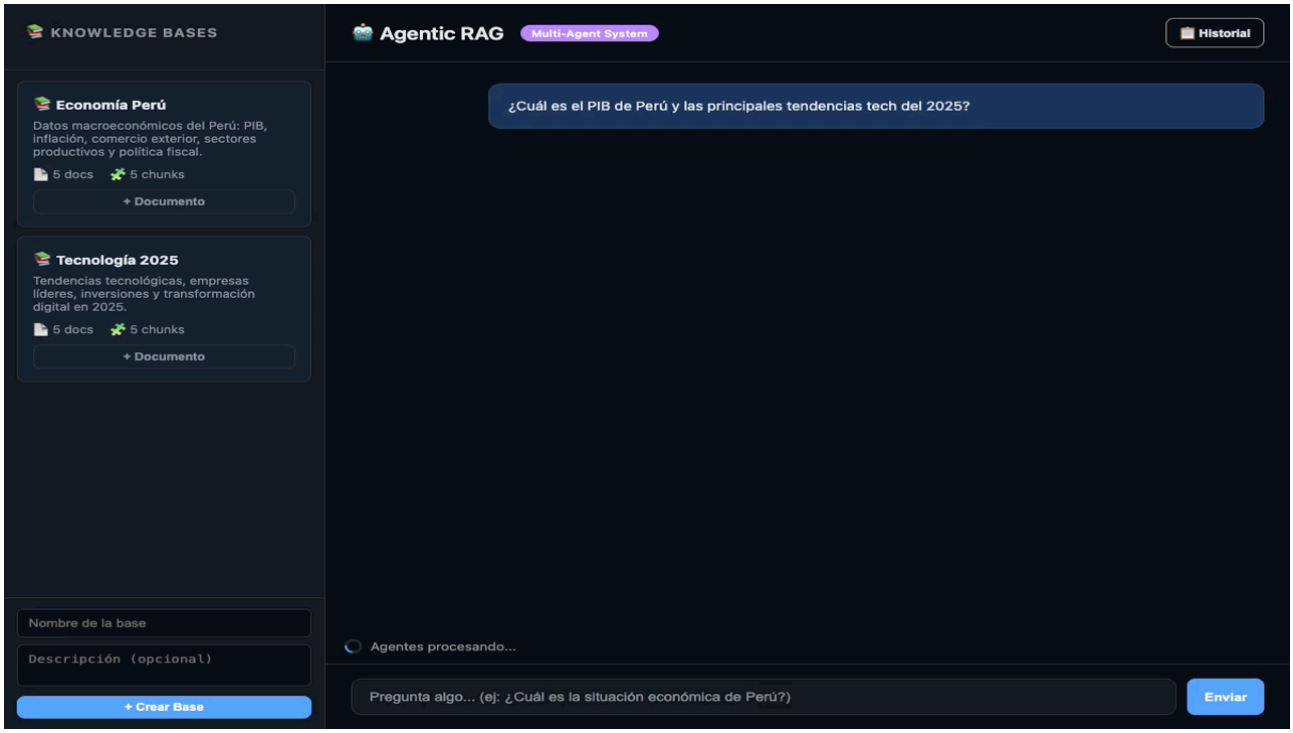
Ventajas

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

Limitaciones

- Extracción lenta (LLM)
- Calidad depende del LLM
- Más complejo

08 — Agentic RAG



¿Qué es?

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

Implementación

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

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

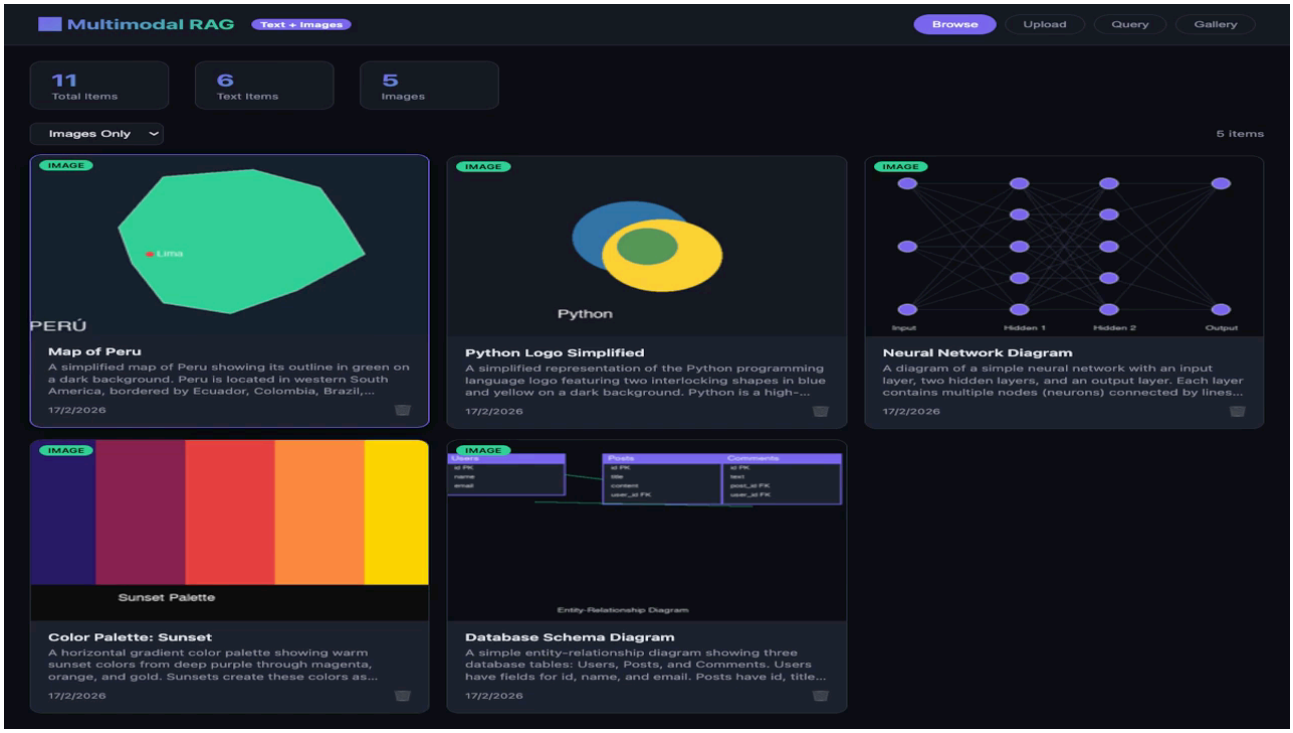
Ventajas

+ Divide y vencerás
+ Trace visible
+ Sin frameworks

Limitaciones

- 4+ LLM calls (~90-120s)
- 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.

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

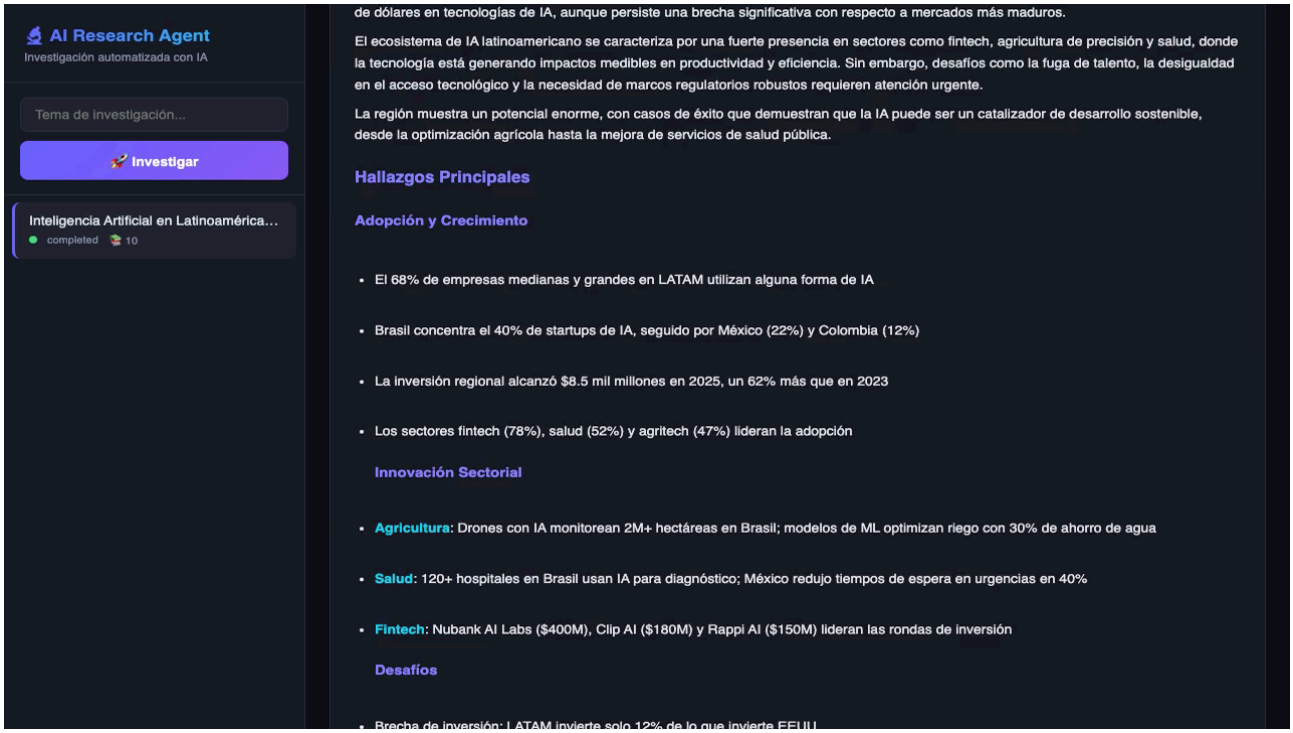
Ventajas

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

Limitaciones

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

10 — AI Research Agent



¿Qué es?

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

Implementación

> Pipeline completo de investigación
> Proyecto demo: 'IA en Latinoamérica 2025'
> 5 hallazgos categorizados
> Reporte Markdown estructurado

Stack: FastAPI + httpx + BeautifulSoup + pgvector | Puerto: 8010

Ventajas

+ End-to-end sin intervención
+ Hallazgos categorizados
+ Reporte exportable

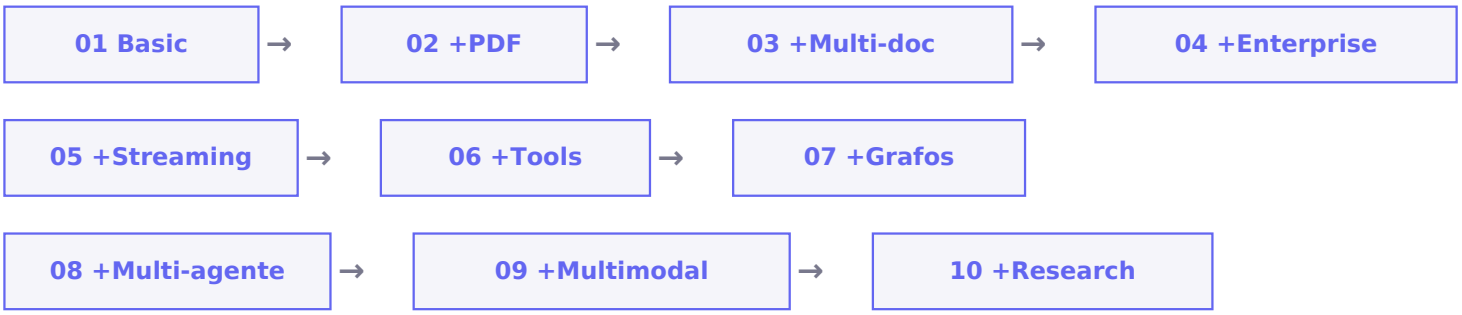
Limitaciones

- Depende de scraping
- Calidad de fuentes variable
- Tarda varios minutos

Resumen Comparativo

Cada proyecto agrega una capa de 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

github.com/manuelarguelles/rag-portfolio