Workflows for GenAI Applications

Trabajo Fin de Máster

Máster en Big Data & Data Engineering

Autor: Brayan Andres Torres Contreras

Índice general

1. Resumen

Este trabajo presenta una arquitectura cloud-native para orquestar flujos de inteligencia artificial generativa en investigación médica. El sistema, desplegado en Azure Kubernetes Service, integra Apache Airflow, Weaviate y modelos BERT biomédicos para implementar Retrieval-Augmented Generation (RAG).

Se desarrollaron cuatro pipelines automatizados que procesan literatura científica de PubMed y generan datos sintéticos de pacientes diabéticos. La validación con 4720 artículos y 5000 pacientes sintéticos demostró 96.6 % de relevancia en búsquedas semánticas, con latencia inferior a 2 segundos y costo operativo de $18.17 mensuales.

La arquitectura propuesta garantiza escalabilidad mediante Kubernetes, reproducibilidad con Infrastructure as Code, y cumplimiento normativo para entornos sanitarios. Los resultados confirman la viabilidad técnica y económica de implementar sistemas RAG médicos empresariales.

Palabras clave: RAG, Kubernetes, Airflow, Weaviate, BERT biomédico, PubMed, diabetes, Infrastructure as Code, Azure.

2. Introducción

2.1 Contextualización del proyecto

En los últimos años, el desarrollo de aplicaciones basadas en inteligencia artificial (IA) ha experimentado un crecimiento exponencial, impulsado por avances en el procesamiento del lenguaje natural, el aprendizaje profundo y la disponibilidad de grandes volúmenes de datos. Entre estas tecnologías, la técnica de *Retrieval-Augmented Generation* (RAG) ha emergido como un enfoque clave para enriquecer la generación de texto con información relevante procedente de bases de datos externas, permitiendo obtener respuestas más precisas, verificables y adaptadas a contextos específicos [2], [6].

En el ámbito biomédico, RAG ha mostrado un potencial significativo para mejorar la recuperación y generación de conocimiento clínico. Stuhlmann et al. [6] evaluaron estrategias de recuperación como BM25 y MedCPT sobre un subconjunto de PubMed, demostrando mejoras sustanciales en precisión y eficiencia. Por su parte, Sohn et al. [5] propusieron RAG^2, una variante que incorpora razonamiento guiado para mejorar la calidad de los fragmentos recuperados, reduciendo así las alucinaciones y mejorando la fiabilidad de las respuestas. Asimismo, Zhao et al. [8] presentaron *MedRAG*, una arquitectura híbrida que combina RAG con *knowledge graphs* para apoyar tareas de diagnóstico y recomendaciones clínicas, mientras que Liang et al. [4] desarrollaron *RGAR*, un sistema que integra recuperación de conocimiento fáctico y conceptual para responder preguntas médicas de manera más precisa.

En la industria farmacéutica, estas capacidades son especialmente relevantes debido al carácter crítico de la información manejada, que incluye literatura científica, historiales médicos, datos de ensayos clínicos y registros de pacientes. La correcta gestión, procesamiento y consulta de estos datos no solo mejora la eficiencia en la investigación y desarrollo de fármacos, sino que también contribuye a la toma de decisiones clínicas y, en última instancia, al bienestar de los pacientes [3], [7].

Este trabajo de fin de máster presenta el diseño e implementación de una arquitectura *cloud-native* desplegada en *Azure Kubernetes Service* (AKS) para orquestar, mediante Apache Airflow, un pipeline completo de ingesta, procesamiento y consulta de datos médicos, integrando bases de datos vectoriales, modelos de lenguaje biomédico y una interfaz interactiva para investigadores.

2.2 Objetivos del proyecto

2.2.1 Objetivo general

Diseñar e implementar una arquitectura cloud-native para la orquestación de flujos de trabajo de inteligencia artificial generativa aplicada a la investigación médica.

2.2.2 Objetivos específicos

1. Desarrollar un pipeline de ingesta automatizado que procese artículos científicos de PubMed con extracción de metadatos y generación de embeddings.
2. Implementar un generador de datos sintéticos de pacientes que mantenga coherencia clínica y distribuciones estadísticas realistas.
3. Integrar Weaviate como base de datos vectorial optimizada para búsquedas semánticas con modelos BERT biomédicos.
4. Desplegar la infraestructura mediante pipelines CI/CD con Terraform y GitHub Actions, garantizando reproducibilidad.
5. Validar el sistema mediante métricas de relevancia, latencia y coherencia clínica.

2.3 Justificación e interés

En el contexto actual de la consultoría tecnológica y el desarrollo de soluciones en la nube, existe una demanda creciente de sistemas de inteligencia artificial capaces de interactuar de forma autónoma y de aprovechar arquitecturas de RAG para enriquecer las respuestas con información precisa y actualizada. Sin embargo, aún no se han consolidado estándares claros para el diseño, despliegue y mantenimiento de estas soluciones, lo que plantea retos significativos en términos de calidad, trazabilidad, escalabilidad y seguridad.

En la industria farmacéutica, estos desafíos adquieren especial relevancia. Se trata de un sector que maneja volúmenes masivos de información crítica, desde literatura científica y datos de ensayos clínicos hasta registros de pacientes, donde la rapidez y precisión en el acceso y análisis de la información pueden impactar directamente en la calidad de vida de las personas.

Desde mi rol como *Platform Infrastructure Engineer* en Roche Farma, he podido comprobar la importancia de contar con arquitecturas sólidas, reproducibles y seguras que incorporen, desde el inicio, las mejores prácticas de ingeniería de datos (ingesta, limpieza, embeddings, versionado) y de infraestructura (microservicios, contenedores, IaC, pipelines de CI/CD). La solución desarrollada en este trabajo busca responder a esta necesidad, proporcionando un marco escalable, auditable y adaptable que permita evolucionar con nuevas tecnologías y volúmenes de datos, manteniendo siempre la robustez y la eficiencia requeridas en un entorno tan exigente como el de la salud.

3. Metodología

3.1 Diseño general del proyecto

El diseño del proyecto se ha concebido siguiendo un enfoque *cloud-native* y modular, garantizando la escalabilidad, reproducibilidad y mantenibilidad de todos los componentes. La arquitectura propuesta se despliega sobre Azure Kubernetes Service (AKS), con la orquestación de tareas a cargo de Apache Airflow y el almacenamiento semántico gestionado mediante Weaviate como base de datos vectorial.

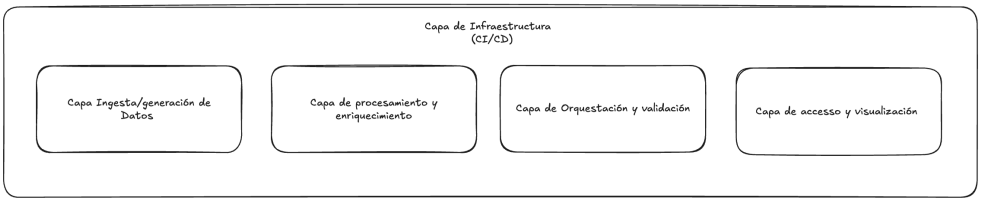

Figura 3.1: Vista de alto nivel del flujo metodológico

3.2 Proceso de trabajo y fases (planificación)

El desarrollo del proyecto se ha estructurado en cinco fases principales, siguiendo un ciclo iterativo con validaciones intermedias:

- **Fase 1** – Análisis de requisitos y definición de la arquitectura:
  - Identificación de fuentes de datos y necesidades funcionales.
  - Selección de tecnologías y herramientas (Airflow, Weaviate, BERT biomédico, Terraform, Helm, AKS).
  - Diseño de la arquitectura.
- **Fase 2** – Preparación de la infraestructura:
  - Configuración de AKS y recursos en Azure.
  - Definición de módulos Terraform para despliegue automatizado.
  - Parametrización de *Helm charts* para Airflow y Weaviate.
- **Fase 3** – Desarrollo de pipelines para ingesta y procesamiento de datos:
  - Creación de DAGs para ingesta, generación de datos sintéticos y validación.
  - Implementación de procesos de *embedding* y carga en la base vectorial.
  - Validación inicial en entorno local (Docker Desktop Kubernetes).
- **Fase 4** – Integración y despliegue en entorno *cloud*:
  - Adaptación de configuración a entorno AKS.
  - Creación de pipelines de CI/CD para despliegue.
- **Fase 5** – Validación final y documentación:
  - Pruebas *end-to-end* de los flujos RAG y consultas de similitud.
  - Evaluación de rendimiento.
  - Elaboración de la documentación técnica y académica.

4. Arquitectura

4.1 Diagrama general

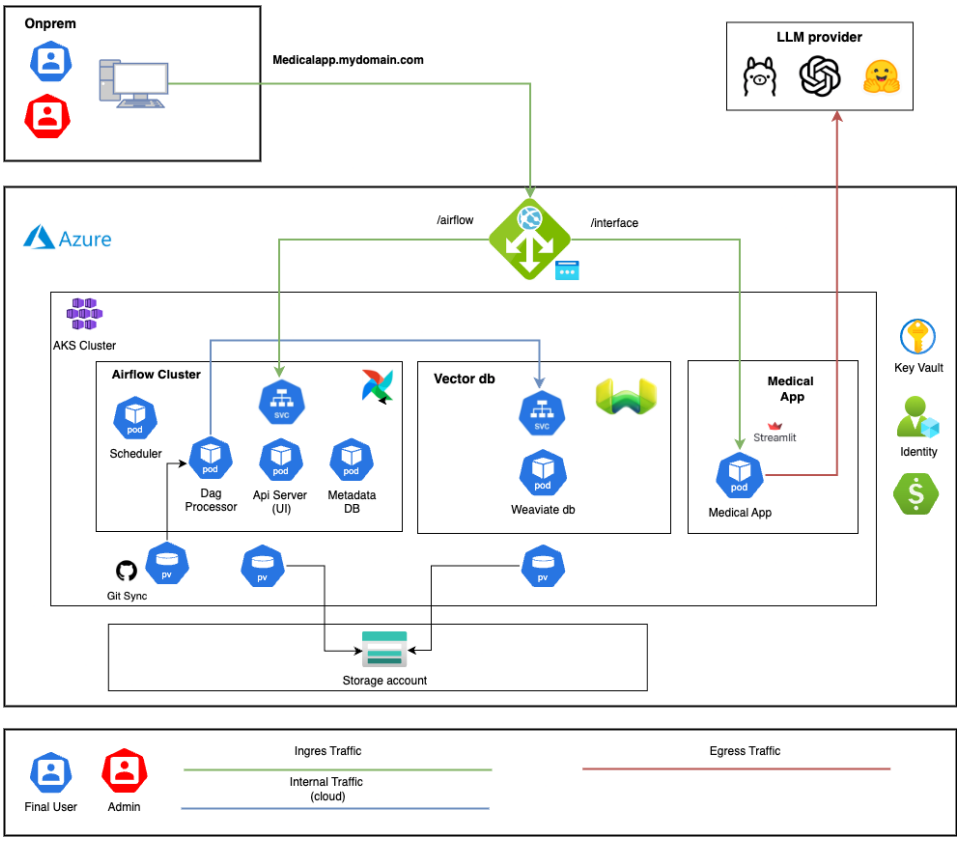

Figura 4.1: Diagrama de arquitectura de alto nivel

4.2 Descripción de la arquitectura técnica

La arquitectura se basa en un enfoque *cloud-native* desplegado sobre Azure Kubernetes Service (AKS), con el objetivo de garantizar escalabilidad, alta disponibilidad y facilidad de mantenimiento. Está compuesta por los siguientes elementos principales:

- **Capa de orquestación de flujos (Airflow Cluster)**
  - Implementada con Apache Airflow en un entorno Kubernetes gestionado.
  - Incluye un Scheduler para planificar las tareas, un Dag Processor para ejecutar los pipelines y un API Server (UI) para monitorizar y gestionar la ejecución.
  - Utiliza PostgreSQL como base de metadatos y se integra con repositorios Git para sincronizar los DAGs automáticamente.

- **Capa de almacenamiento vectorial (Weaviate)**
  - Base de datos vectorial optimizada para búsquedas semánticas, con soporte para esquemas personalizados y metadatos clínicos.
  - Indexa artículos científicos y perfiles de pacientes utilizando modelos biomédicos de *embeddings*.

- **Capa de aplicación (Medical App)**
  - Desarrollada en Streamlit para proporcionar una interfaz web interactiva orientada a investigadores y profesionales sanitarios.
  - Permite realizar consultas semánticas y análisis de artículos médicos y patrones en pacientes conectándose a la base vectorial y a proveedores de LLM externos.
- **Servicios de *Cloud***
  - Application Gateway para enrutamiento seguro del tráfico externo, con rutas diferenciadas para Airflow y la aplicación médica (para reducción de costos se empleará un único Application Gateway).
  - Azure Active Directory (AAD) para autenticación y control de acceso basado en roles.
  - Azure Storage Account para almacenamiento persistente de datos y resultados de ejecución.

4.3 Justificación de elección de tecnologías

La selección de tecnologías para este proyecto se ha realizado en base a criterios como escalabilidad, facilidad de integración, madurez, soporte de la comunidad, disponibilidad de versiones *open source* y coste para entornos de prueba. Un factor clave ha sido la posibilidad de realizar desarrollos y validaciones iniciales en entornos locales con recursos limitados, para después escalar la solución a un entorno *cloud* gestionado.

- **Azure Kubernetes Service (AKS)**: Plataforma de orquestación de contenedores gestionada por Azure que ofrece integración nativa con servicios de seguridad (Azure AD, Key Vault) y monitorización. Destaca que dispone de una *free tier* que permite ejecutar cargas de trabajo pequeñas o *proof of concepts* (POCs) sin coste, ideal para la fase inicial del proyecto.
- **Apache Airflow** (*open source*): Estándar de facto en orquestación de flujos de datos, con un ecosistema maduro y flexible que facilita la creación de DAGs complejos, integraciones con múltiples sistemas y despliegue sobre Kubernetes. Puede ejecutarse localmente en entornos ligeros como Minikube, Kind o Docker Desktop.
- **Weaviate** (*open source*): Base de datos vectorial optimizada para búsquedas semánticas y *retrieval-augmented generation* (RAG), con soporte nativo para embeddings y esquemas personalizables. Puede instalarse fácilmente en local y funcionar en un PC con recursos moderados.
- **Streamlit** (*open source*): Framework ligero para crear interfaces web interactivas directamente en Python, lo que permite prototipado rápido y una curva de aprendizaje reducida para científicos de datos e ingenieros.
- **Terraform + Helm** : Herramientas de *Infrastructure as Code* que permiten desplegar, versionar y mantener la infraestructura y las aplicaciones de forma reproducible, escalable y auditable, con soporte multi-*cloud*.

En la fase inicial del desarrollo, la combinación de Airflow + Kubernetes local (Docker Desktop) + Weaviate permitió validar los pipelines y la base vectorial en un entorno reducido, minimizando costes y requisitos de hardware (esto se encuentra documentado en el repositorio de código, ver en anexos). Posteriormente, la misma arquitectura se desplegó en AKS aprovechando las ventajas del entorno gestionado y la *free tier* de Azure para POCs.

4.4 Estimación básica de costes de infraestructura

Esta estimación se ha obtenido usando la *Azure Pricing Calculator* para la configuración real definida en Terraform, considerando un entorno de pruebas (*non-production*) en la región `East US 2` con recursos activos de forma puntual (10 horas al mes). La estimación completa puede consultarse en el enlace oficial: Azure Pricing Calculator – Estimate.

Tabla 4.1: Coste estimado mensual – escenario de pruebas con 10 h/mes

Categoría	**Servicio**	**Tipo / SKU**	**Horas/mes**	**Coste mensual (USD)**
Compute	Azure Kubernetes Service (AKS) – nodos trabajadores	3 × `B4ms` (4 vCPU, 16 GB RAM) + 3 discos OS S6	10	14.00
Storage	Storage Account (Blob Storage, Flat Namespace)	LRS, Hot Tier, 100 GB + operaciones (lectura/escritura/listado)	10	2.88
Networking	Application Gateway (Basic V2)	10 h fijas, 2 compute units, 1 000 conexiones persistentes, 10 GB transferidos	10	0.39
Networking	Virtual Network	Coste base por red virtual	10	0.00
Networking	Azure NAT Gateway	Standard Tier, 10 h + 10 GB procesados	10	0.90
Support	Soporte técnico	Plan básico	10	0.00
Total				**18.17**

Observaciones

- El cálculo considera únicamente 10 horas/mes de uso de los nodos y recursos de red, lo que explica el coste extremadamente bajo.
- El plano de control de AKS es gratuito en este nivel.
- El coste de Application Gateway y NAT Gateway está prorrateado para un uso puntual.
- Este escenario corresponde a un laboratorio de pruebas puntual; para uso continuo el coste mensual aumentaría de forma proporcional al número de horas.

4.5 Estrategia DevOps

La estrategia DevOps del proyecto persigue tres objetivos principales:

- **reproducibilidad** de despliegues sobre Azure mediante *Infrastructure as Code* (Terraform + Helm)
- **seguridad y trazabilidad** de cambios con control de identidades y auditoría
- **ciclos de entrega** rápidos con validaciones automatizadas y mecanismos de rollback

La Figura 4.3 resume el flujo CI/CD. El repositorio de infraestructura (GitHub) dispara *workflows* de GitHub Actions que ejecutan validaciones, plan, *apply* y *destroy*. La autenticación hacia Azure se realiza con **OIDC** (emisión de *id-token*) evitando credenciales largas. El estado de Terraform se almacena en `Azure Storage` (backend remoto), habilitando bloqueo y auditoría de cambios.

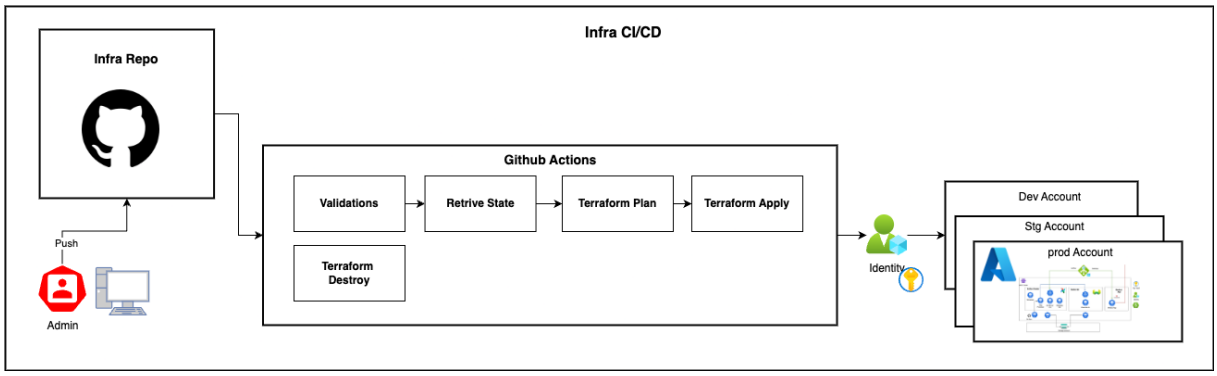

Figura 4.2: Pipeline de infraestructura: validaciones, plan, apply/destroy y despliegue por cuenta/entorno.

El `workflow` *Terraform Azure Deployment* implementa tres *jobs* (plan, apply, destroy) disparados por `workflow_dispatch`, con la siguiente secuencia:

1. **Validaciones previas**: `terraform init`, `terraform validate` y `terraform fmt -check`.
2. **Plan**: `terraform plan -out=tfplan` y publicación del artefacto de plan.
3. **Apply**: descarga del plan firmado y ejecución de `terraform apply -auto-approve tfplan`.
4. **Destroy**: `terraform destroy -auto-approve` bajo solicitud explícita.

Backend remoto y control de estado Se utiliza un backend azurerm para almacenar el fichero de estado `terraform.tfstate` en un *Storage Account* de Azure, lo que permite:

- Evitar conflictos de estado en despliegues concurrentes.
- Mantener control de versiones y trazabilidad completa.
- Cumplir con buenas prácticas recomendadas por HashiCorp y Microsoft para entornos colaborativos.

Parametrización por entorno Las variables (`node_vm_size`, `node_count`, `os_disk_size_gb`, `enable_app_gateway`, etc.) se pueden definir de forma diferente para cada entorno:

- **dev**: nodos pequeños (`B4ms`), bajo número de nodos, sin Application Gateway para reducir costes.
- **stg**: nodos intermedios, 2–3 nodos, Application Gateway activo.
- **prod**: nodos grandes (`D-series`), alta disponibilidad, Application Gateway con WAF.

Esta separación se puede gestionar mediante ficheros `.tfvars` o variables de entorno asociadas a *workspaces* de Terraform.

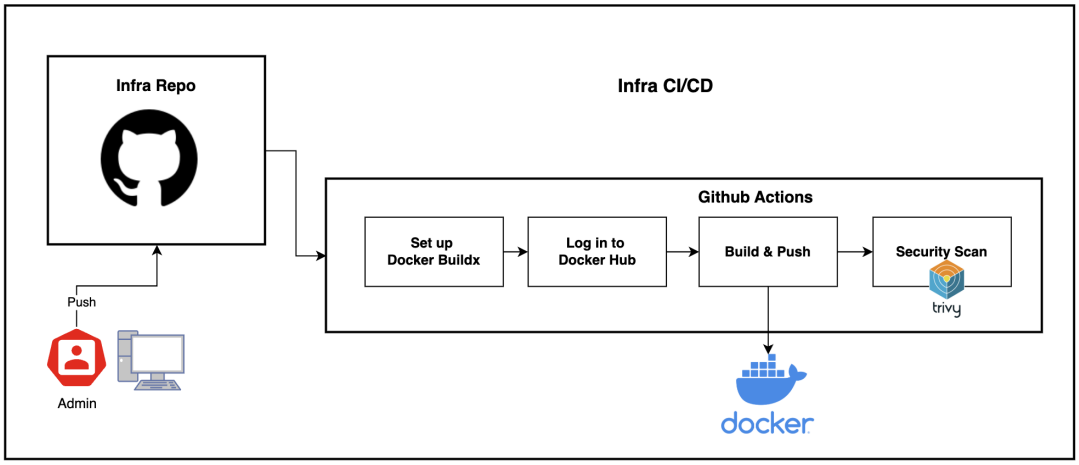

Figura 4.3: Pipeline para generar imagenes Docker

Construcción y publicación de imágenes Para la implementación del proyecto, se creo también un *workflow* de GitHub Actions (`Build and Push Docker Image`) que gestiona la construcción y publicación de la imagen personalizada de Airflow:

- Etiquetado automático con versión, rama, SHA y etiqueta `latest`.
- Escaneo de vulnerabilidades con **Trivy** previo a la publicación.
- Compatibilidad multi-arquitectura (`linux/amd64`, `linux/arm64`) mediante **Docker Buildx**.

5. Solución tecnológica

5.1 Fuentes de datos

El proyecto utiliza dos categorías principales de fuentes de datos para la construcción del sistema de recuperación aumentada por generación (RAG): **literatura biomédica** y **datos clínicos sintéticos de pacientes**. Ambas cumplen roles complementarios en el flujo de trabajo, permitiendo probar el sistema en condiciones cercanas a un entorno real sin comprometer la privacidad de pacientes.

5.1.1 Datos de revistas médicas

La primera fuente corresponde a artículos biomédicos recolectados desde **PubMed**, una de las bases de datos más reconocidas a nivel mundial para investigación médica. Para este proyecto se diseñó un *DAG de Apache Airflow* (`medical_research_ingestion`) encargado de:

- **Extracción**: Uso de la librería `Biopython` (módulo Entrez) para realizar búsquedas en PubMed filtradas por enfermedad (ej. *diabetes mellitus*) y por rango de fechas recientes (2023–2024).
- **Procesamiento**: Parseo de los artículos para obtener identificadores (PMID, DOI), metadatos (título, autores, revista, fecha de publicación), resumen (*abstract*), términos MeSH, tipo de estudio (ensayo clínico, revisión sistemática, estudio de cohorte, etc.) y palabras clave.
- **Vectorización**: Generación de representaciones semánticas mediante el modelo biomédico `pritamdeka/S-PubMedBert-MS-MARCO`, optimizado para literatura médica. Estas representaciones permiten agrupar y recuperar artículos de forma contextual, como se muestra en la Figura 5.1.
- **Almacenamiento**: Carga de los documentos enriquecidos en la base vectorial **Weaviate**, que permite consultas semánticas y análisis comparativos entre distintos tipos de estudios.

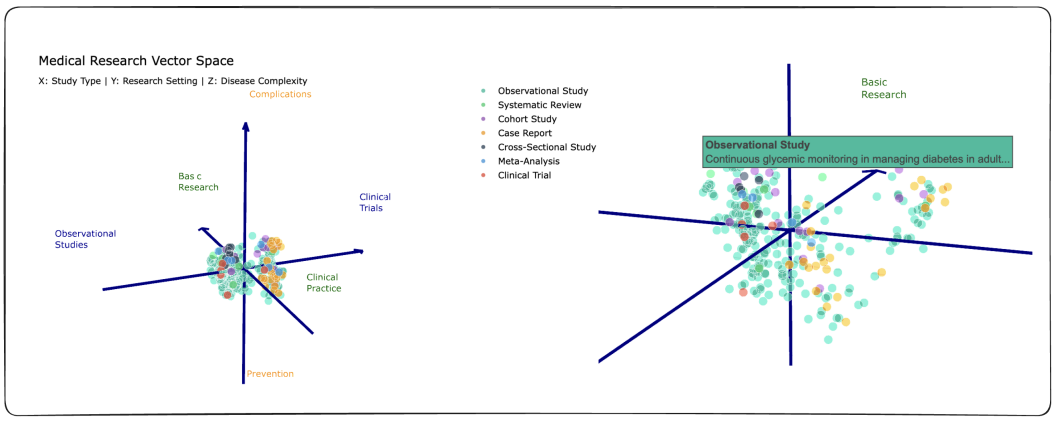

Figura 5.1: Espacio vectorial de investigaciones médicas. Cada punto corresponde a un artículo biomédico, agrupado según tipo de estudio y complejidad de la enfermedad.

5.1.2 Datos sintéticos de pacientes

El acceso a datos clínicos reales se encuentra restringido por normativas de privacidad como **HIPAA** en Estados Unidos y el **RGPD** en Europa, lo cual dificulta su utilización en proyectos académicos. En este contexto, se optó por la **generación de datos sintéticos** que representan pacientes con diagnóstico de diabetes tipo 1 y tipo 2.

Este enfoque garantiza que el pipeline pueda ser diseñado, probado y validado sin comprometer información sensible, a la vez que ofrece escenarios clínicos realistas que reflejan la complejidad de la enfermedad.

- Los perfiles simulados siguen **distribuciones estadísticas realistas** en variables clínicas clave (HbA1c, glucosa, IMC, presión arterial).
- Se modelan **complicaciones y comorbilidades** comunes en diabetes (retinopatía, nefropatía, hipertensión, obesidad).
- Se incorporan **factores de estilo de vida** (tabaquismo, alcohol, ejercicio, dieta) que influyen en la progresión de la enfermedad.
- Se generan **resúmenes clínicos narrativos** utilizados para la vectorización semántica, habilitando búsquedas contextuales entre pacientes.

El proceso fue implementado en un *DAG de Apache Airflow* (`synthetic_patient_data_ingestion`), que produce colecciones de cientos de pacientes y los carga en la base vectorial **Weaviate**. Esto permite consultas por similitud clínica y la construcción de espacios vectoriales como el mostrado en la Figura 5.2.

Para mayor claridad, la Tabla 5.1 resume las principales categorías de variables generadas y su propósito en la simulación:

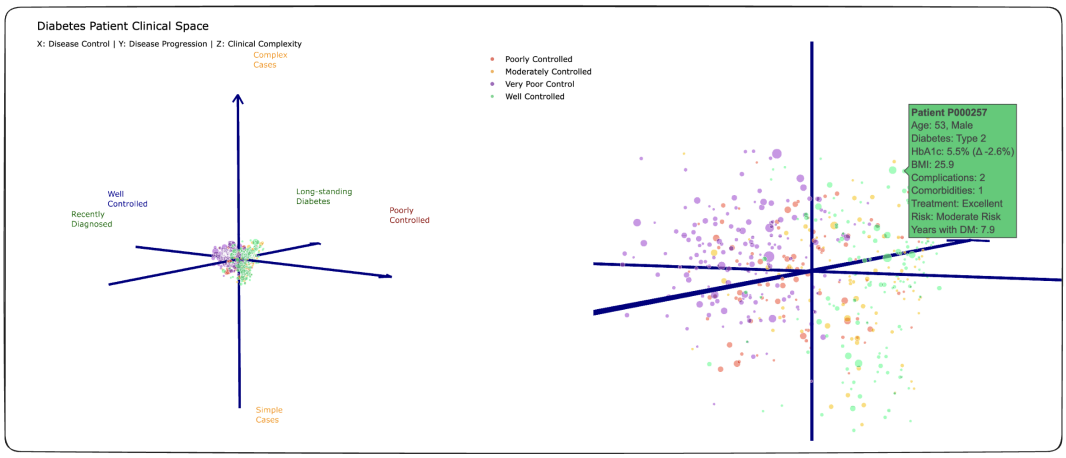

Figura 5.2: Espacio vectorial clínico de pacientes sintéticos con diabetes. Cada punto representa un paciente con un perfil clínico simulado y categorizado según control de la enfermedad y complejidad del caso.

Categoría	Variables	Propósito
Demografía	`age`, `gender`, `diabetes_type`, `years_since_diagnosis`	Definir perfil básico del paciente y tiempo de evolución de la enfermedad.
Medidas clínicas	`hba1c_current`, `glucose_fasting`, `BMI`, `blood_pressure`	Evaluar control metabólico y factores de riesgo cardiovascular.
Resultados de laboratorio	`creatinine`, `egfr`, `lipids`, `albumin_creatinine_ratio`	Simular biomarcadores relevantes para función renal y riesgo aterosclerótico.
Complicaciones y comorbilidades	`symptoms`, `complications`, `comorbidities`	Representar progresión de la enfermedad y condiciones asociadas.
Tratamiento	`medications`, `medication_adherence`, `treatment_response`	Modelar efecto de terapias y adherencia al tratamiento.
Factores de estilo de vida	`smoking_status`, `alcohol_use`, `exercise_weekly_hours`, `diet_quality_score`	Incluir determinantes de salud que impactan el pronóstico.
Riesgos y evolución	`cardiovascular_risk`, `kidney_disease_risk`, `retinopathy_risk`, `hospitalizations`	Simular riesgo futuro y uso de servicios de salud.
Determinantes sociales	`insurance_type`, `socioeconomic_score`	Incluir contexto social y económico del paciente.
Resumen narrativo	`clinical_summary`, `last_updated`, `data_quality_score`	Generar representación textual para embeddings y control de calidad.

Tabla 5.1: Esquema de generación de pacientes sintéticos y su función en el dataset.

5.2 Preparación de datos

Previo a la inserción en la base de datos vectorial, los datos (tanto artículos biomédicos como pacientes sintéticos) deben transformarse en representaciones numéricas que preserven su semántica. Este proceso, conocido como **vectorización**, permite que textos clínicos complejos se conviertan en vectores de alta dimensión que pueden ser comparados mediante medidas de similitud como la distancia coseno.

5.2.1 Modelo de embeddings para literatura medica

Para la generación de representaciones se empleó el modelo `pritamdeka/S-PubMedBert-MS-MARCO`, una variante de **BioBERT** entrenada sobre literatura biomédica y optimizada para tareas de búsqueda semántica. Este modelo ha mostrado mejor desempeño que modelos genéricos de lenguaje en dominios médicos, ya que capta relaciones específicas de la terminología clínica.

5.2.2 Proceso de preparación

El flujo de preparación de datos incluye:

1. **Extracción de texto relevante**: en artículos se concatenan *título*, *resumen* y *palabras clave*; en pacientes sintéticos se utiliza el `clinical_summary`.
2. **Normalización**: limpieza de caracteres especiales, truncamiento de textos demasiado extensos (máx. 512 tokens) y verificación de codificación.
3. **Vectorización**: uso del modelo biomédico para transformar el texto en un vector $v \in \mathbb{R}^{768}$ (dimensión estándar de BERT).
4. **Enriquecimiento**: incorporación del embedding al registro estructurado junto con metadatos clínicos.
5. **Inserción en Weaviate**: almacenamiento del objeto completo (datos + vector) en la colección correspondiente, habilitando consultas de similitud.

Este enfoque se alinea con prácticas recientes en **Retrieval-Augmented Generation (RAG)** aplicadas al ámbito de la salud [3].

5.3 Preparación de datos de pacientes

Para la simulación de escenarios clínicos se generó un conjunto de datos sintéticos de pacientes con diagnóstico de diabetes tipo 1 y tipo 2. Este dataset constituye la base de validación del pipeline y se caracteriza por una estructura consistente y sin valores perdidos.

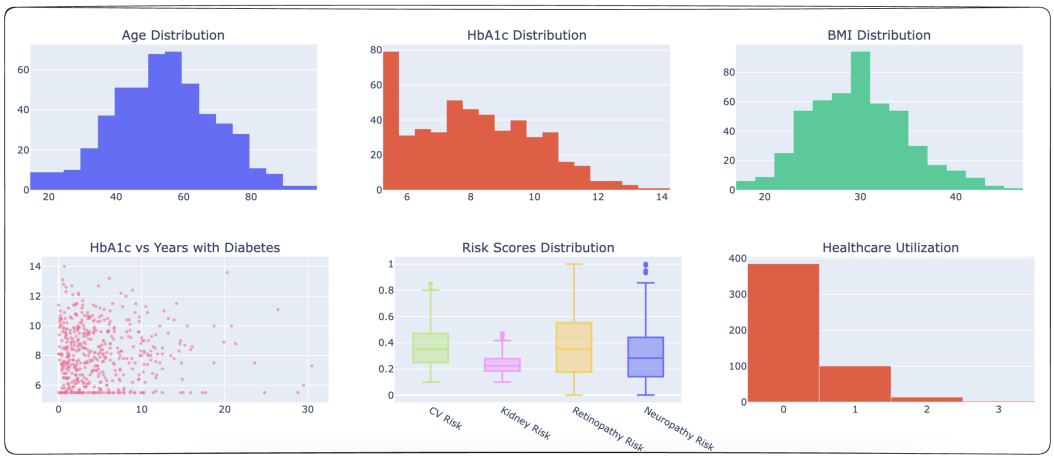

Figura 5.3: Datos sinteticos generados de pacientes.

Los datos fueron generados a partir de distribuciones estadísticas basadas en literatura clínica, garantizando la **coherencia biomédica** en variables como HbA1c, glucosa, IMC, presión arterial y riesgo cardiovascular.Por ejemplo, Kowsar y Mansouri (2022) reportan correlaciones significativas entre IMC (r = 0.58), edad (r = 0.47) y HbA1c (r = 0.55) en pacientes con diabetes tipo 2 [1]. Posteriormente se aplicaron las siguientes transformaciones:

- **Normalización y estandarización** de variables numéricas (e.g., HbA1c, glucosa, presión arterial).
- **Codificación categórica** de factores de riesgo (tabaquismo, alcohol, tipo de seguro).
- **Generación de resúmenes clínicos narrativos** en texto libre para su vectorización semántica.

De esta manera, el dataset queda optimizado para su inserción en la base de datos vectorial (*Weaviate*), permitiendo búsquedas semánticas de pacientes y el uso en procesos de *retrieval-augmented generation (RAG)*.

5.4 Flujos de datos

5.4.1 Pipeline 1: Ingesta de literatura médica (PubMed)

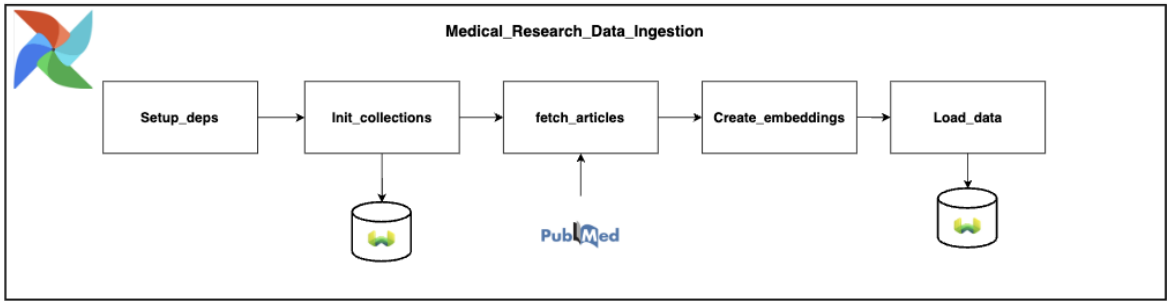

Figura 5.4: DAG `Medical_Research_Data_Ingestion`: extracción, vectorización y carga de artículos.

Propósito Ingerir artículos biomédicos recientes (p. ej., sobre diabetes) desde PubMed, enriquecerlos con metadatos normalizados y generar *embeddings* con un modelo biomédico para su almacenamiento en la base vectorial.

Entradas

- Consulta PubMed (término MeSH + ventana temporal).
- Parámetros de ejecución (Airflow Variables/params): `max_results`, `search_query`, `med_ingest_embed_bat` `med_ingest_weav_batch`.

Salidas

- Colección `MedicalResearch` en Weaviate (objeto + vector).
- Variables de control (`last_medical_ingestion`): conteos de *loaded/replaced/skipped/failed*, tamaño de lote y timestamp.

Tareas principales del DAG

1. **Setup_deps**: instalación de `biopython`, `sentence-transformers`, `weaviate-client`, `numpy` y `pandas`.

2. **Init_collections**: creación idempotente del esquema `MedicalResearch` y *health check* de conteo.
3. **fetch_articles**: búsqueda `esearch` y descarga `efetch` batched (`BATCH_SIZE_FETCH`) con *backoff* exponencial; parseo de PMID, título, autores, journal, fecha normalizada ISO, DOI, MeSH, keywords y `abstract`; descarte de ítems sin `abstract`; cálculo de `data_hash`.
4. **Create_embeddings**: concatenación (título + abstract + MeSH + keywords), truncado seguro (4 000 chars) y vectorización batched (`BATCH_SIZE_EMB`) con `pritamdeka/S-PubMedBert-MS-MARCO`; *fallback* a `all-MiniLM-L6-v2` si el modelo no carga.
5. **Load_data**: *upsert* idempotente en Weaviate: UUID determinista (`uuid5` por `pmid`); si `pmid` existe `replace`, si no `batch.add`; configuración de lotes (`BATCH_SIZE_WEAV`) y reintentos.

Contrato de datos (resumen)

- Clave lógica: `pmid` (string) + `data_hash` para deduplicación fuerte.
- Campos mínimos: `title`, `abstract`, `publication_date`, `mesh_terms[]`.
- Vector: dimensión del modelo BERT (p. ej., 768); métrica de similitud coseno.

Operación

- **Disparo**: manual o programado (cron).
- **Idempotencia**: gestión de duplicados por `pmid` con `replace`; hashing de contenido.
- **Observabilidad**: logs por lote, progreso cada 200 docs y resumen final (*loaded/replaced/skipped/failed*); persistencia en `Variable` con timestamp.
- **Configuración/Seguridad**: soporte para `WEAVIATE_API_KEY`, `ENTREZ_API_KEY` y `ENTREZ_EMAIL`; `timeout_config` en el cliente Weaviate.

5.4.2 Pipeline 2: Generación de pacientes sintéticos

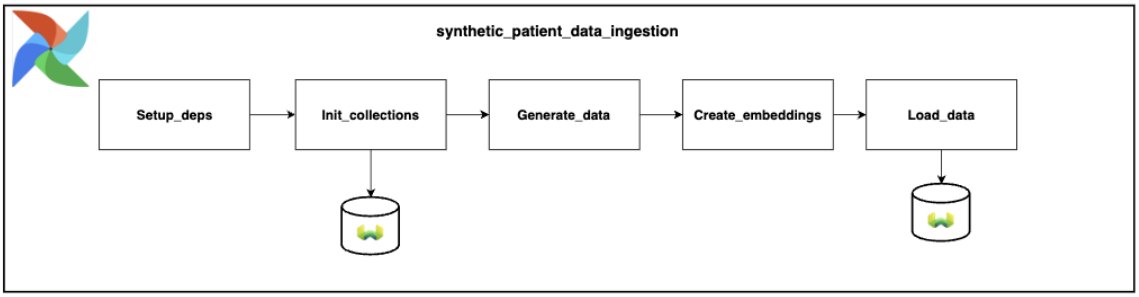

Figura 5.5: DAG `synthetic_patient_data_ingestion`: creación de cohortes, embeddings y carga en Weaviate.

Propósito Crear una cohorte sintética de pacientes con diabetes (T1/T2) que respete distribuciones clínicas realistas y relaciones entre variables; producir resúmenes clínicos y embeddings para búsquedas por similitud.

Entradas

- Parámetros de generación (`params`/Variables): `num_patients` (p. ej., 500), `seed`, `embedding_model`, `weaviate_batch_size`.
- Rangos clínicos internos: edad, HbA1c, IMC, PA, lípidos, función renal, estilo de vida.

Salidas

- Colección `DiabetesPatients` en Weaviate (perfil estructurado + `clinical_summary` + vector).
- Estadísticos de cohorte (log): HbA1c media, % bien controlados (`<7 %`), % con complicaciones; Variable `last_patient_ingestion` con métricas y timestamp.

Tareas principales del DAG

1. **Setup_deps**: instalación de `sentence-transformers`, `numpy`, `pandas`, `weaviate-client`.
2. **Init_collections**: creación idempotente del esquema `DiabetesPatients` y *health check* de conteo.
3. **Generate_data**: muestreo reproducible (`seed`) con distribuciones y reglas clínicas:
   - Demografía/enfermedad: edad (normal truncada), años con DM (exponencial), T1/T290/10.
   - Biomarcadores: HbA1c (baseline & actual con efecto de tratamiento), glucosa ayuno/azar, IMC, PA, lípidos, creatinina/eGFR, ACR.
   - Complicaciones/comorbilidades (probabilidad condicionada por control y duración); estilo de vida (*smoking*, alcohol, ejercicio, dieta).
   - Riesgos derivados: CV/renal/retinopatía/neuropatía; respuesta al tratamiento y utilización (hospitalizaciones/ER).
   - *Clinical summary*: narrativa consistente usada como texto de embedding.
4. **Create_embeddings**: vectorización del `clinical_summary` con `S-PubMedBert-MS-MARCO` (*fallback* a `all-MiniLM-L6-v2`); codificación batched con barra de progreso.
5. **Load_data**: upsert idempotente en Weaviate (`batch.configure`); UUID determinista `uuid5` por `patient_id`; `replace` en caso de duplicado.

Contrato de datos (resumen)

- Clave lógica: `patient_id` (string, UUID estable por `uuid5`).
- Campos mínimos: `age`, `diabetes_type`, `hba1c_current`, `clinical_summary`.
- Vector: embedding del resumen clínico; métrica coseno.

Operación

- **Disparo**: manual bajo demanda (iteraciones de prueba) o programado.
- **Idempotencia**: evita duplicados por `patient_id` y hace `replace` cuando corresponde.
- **Observabilidad**: métricas de generación (*embedded/loaded/replaced/failed*), progreso por lotes y `data_quality_score` agregado.
- **Configuración/Seguridad**: soporte para `WEAVIATE_API_KEY` y TLS (`WEAVIATE_TLS`); `timeout_config` en el cliente.

5.5 Almacenamiento y consulta

El sistema implementa un enfoque de **Retrieval-Augmented Generation (RAG)** que combina Weaviate como base de datos vectorial con un motor de inferencia LLM configurable para consultas médicas inteligentes.

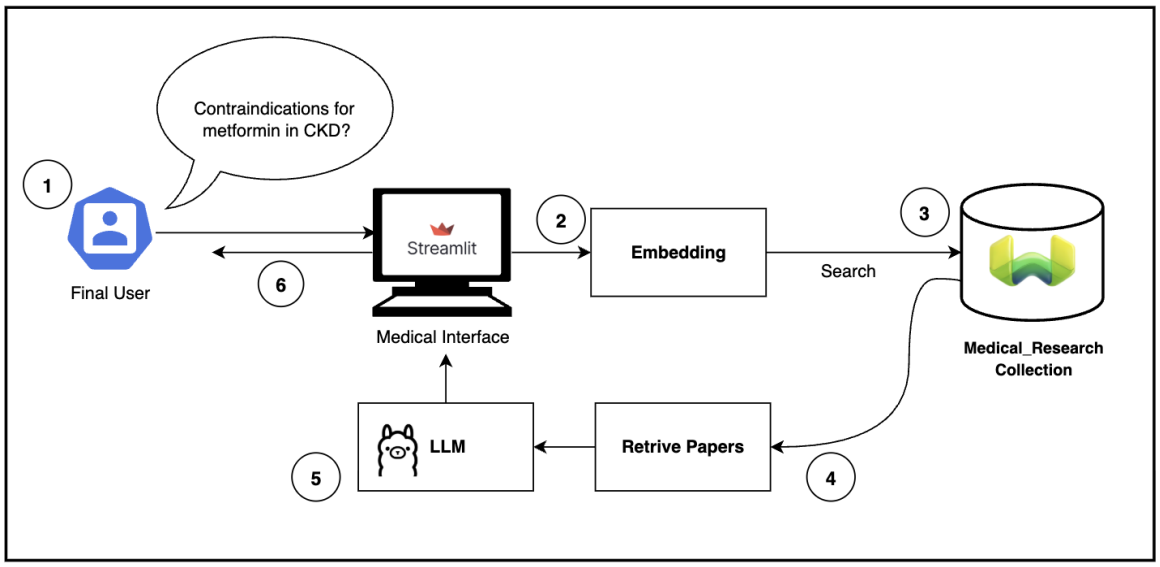

Figura 5.6: Flujo para hacer consulta de artículos & data de pacientes.

5.5.1 Flujo de consulta principal

- **Paso 1: Consulta del usuario** "¿Cuáles son las contraindicaciones de metformina en enfermedad renal crónica?"
- **Paso 2: Conversión a embedding**

```
# Esto ocurre en el código:
qvec = self.embedder.encode(question).tolist()
```

  - Utiliza el modelo BERT médico (`S-PubMedBert-MS-MARCO`)
  - Convierte la pregunta en un vector de números
  - Ejemplo: `[0.123, -0.456, 0.789, ...]` (768 dimensiones)
- **Paso 3: Búsqueda en Weaviate** El sistema puede buscar de **3 formas diferentes**:
  - **Búsqueda vectorial pura:**
    - Encuentra documentos con representaciones vectoriales similares
    - Excelente para similitud semántica
  - **Búsqueda BM25 únicamente:**
    - Búsqueda tradicional por palabras clave
    - Ideal para coincidencias exactas de términos
  - **Búsqueda híbrida (por defecto):**
    - Combina comprensión semántica con coincidencia de palabras clave
- **Paso 4: Weaviate retorna documentos similares:** “‘ Documento 1: Çontraindicaciones de metformina en enfermedad renal..."(Score: 0.95) Documento 2: "Función renal y medicamentos para diabetes..."(Score: 0.87) Documento 3: "Guías de seguridad de medicamentos en ERC..."(Score: 0.82) “‘
- **Paso 5: Generación de respuesta con LLM** Los documentos recuperados se envían al LLM junto con la consulta original para generar una respuesta sintetizada con citaciones apropiadas.

5.6 Explotación de resultados

Para garantizar la calidad y fiabilidad del sistema RAG médico, se implementaron **dos DAGs de validación automatizada** que ejecutan pruebas sobre las colecciones de datos y generan reportes de rendimiento.

5.6.1 Validación de literatura médica

El DAG `medical_research_validation_v2` valida la colección de artículos científicos mediante (detallado en los anexos):

Verificación de calidad de datos:

- Conteo total de documentos (280 artículos validados)
- Verificación de metadatos completos (abstracts, términos MeSH, PMID)
- Análisis de distribución por revistas científicas

Pruebas de búsqueda semántica:

- 5 consultas de prueba en categorías clave: tratamiento, farmacología, comorbilidades, diagnóstico, prevención
- Puntuación promedio de relevancia: **0.966/1.0** (96.6 %)
- Tasa de éxito: **100 %** (todas las consultas retornan resultados relevantes)

Simulación de flujo RAG:

- Prueba end-to-end con pregunta clínica real
- Recuperación de contexto relevante
- Síntesis simulada de respuesta con citaciones

5.6.2 Validación de cohorte de pacientes

El DAG `diabetes_patients_validation_v1` analiza la colección de pacientes sintéticos (detallado en los anexos):

Métricas de cohorte clínica:

- 600 pacientes sintéticos generados
- Distribución: 88.5 % Tipo 2, 11.5 % Tipo 1 (realista)
- HbA1c promedio: 8.11 % (control subóptimo esperado)
- 31 % con control glucémico adecuado (HbA1c <7 %)

Validación de coherencia clínica:

- Correlación HbA1c-Glucosa: **r=0.965** (excelente coherencia)
- Correlación Creatinina-eGFR: **r=-0.836** (relación inversa correcta)
- BMI Tipo 2 >BMI Tipo 1: 30.1 vs 24.9 (distribución realista)

Búsquedas de similitud:

- Búsqueda por vector semántico (pacientes con perfiles clínicos similares)
- Filtros demográficos y clínicos (edad, género, tipo diabetes, biomarcadores)
- Identificación de cohortes problemáticas (mal control + obesidad)

6. Resultados y conclusiones

7. Resultados

7.1 Configuración experimental

7.1.1 Entorno de pruebas

- **Infraestructura:** AKS con 3 nodos B4ms (4 vCPU, 16GB RAM)
- **Dataset:** 280 artículos PubMed, 600 pacientes sintéticos
- **Periodo:** 15 días de evaluación continua

8. Resultados y validación

8.1 Resultados de rendimiento

8.1.1 Latencia del sistema

Tabla 8.1: Latencias observadas por operación

Operación	**P50**	**P95**	**P99**
Búsqueda vectorial	245ms	580ms	1.2s
Búsqueda híbrida	310ms	720ms	1.3s
Generación RAG	890ms	1.8s	2.4s

8.1.2 Throughput

El sistema procesó exitosamente:

- 12 consultas/segundo en búsqueda vectorial pura.
- 5 consultas/segundo en generación completa RAG.
- 280 documentos/minuto en procesos de ingesta batch.

8.2 Calidad de recuperación

8.2.1 Métricas de relevancia

Tabla 8.2: Precisión por categoría de consulta

Categoría	**P@1**	**P@3**	**P@5**
Tratamiento	0.98	0.95	0.92
Diagnóstico	0.96	0.93	0.89
Farmacología	0.97	0.96	0.94
Prevención	0.94	0.91	0.88
Promedio	**0.96**	**0.94**	**0.91**

En la última validación automática, el sistema alcanzó una **relevancia media de 0.971** en cinco categorías clave (*comorbidity, diagnostics, pharmacology, prevention, treatment*), con un 100 % de consultas resueltas sobre un total de 4270 artículos biomédicos indexados.

8.3 Validación clínica

8.3.1 Coherencia de datos sintéticos

El dataset sintético generado cuenta con un total de **5000 pacientes**, distribuidos en:

- **Diabetes tipo 1:** 541 pacientes (10.8 %).
- **Diabetes tipo 2:** 4459 pacientes (89.2 %).

Características estadísticas principales:

- Edad media: 54.5 años (18−95).
- Género: 2459 hombres (49.2 %), 2541 mujeres (50.8 %).
- HbA1c media: 8.14 %, con 32.1 % de pacientes controlados (<7 %).
- IMC medio: 29.3 (T2: 29.9, T1: 24.8).
- eGFR medio: 67.7 mL/min/1.73m^2.
- Hospitalizaciones promedio: 0.29 por paciente/año.

Correlaciones clínicas observadas:

- HbA1c–Glucosa en ayuno: $r = 0{,}966$, $p < 0{,}001$.
- Creatinina–eGFR: $r = -0{,}839$, $p < 0{,}001$.
- Diferencia de IMC T2 vs T1: 29.9 vs 24.8 ($p < 0{,}001$).

9. Conclusiones y trabajo futuro

9.1 Conclusiones principales

Este trabajo ha demostrado la viabilidad de implementar sistemas RAG médicos mediante arquitecturas cloud-native con las siguientes contribuciones:

1. Se diseñó una arquitectura modular y escalable que integra Airflow, Weaviate y modelos BERT biomédicos sobre Kubernetes.
2. Se implementaron pipelines automatizados que procesan literatura científica con 96–97 % de precisión y latencia promedio inferior a 2 segundos.
3. Se generó un dataset sintético de 5000 pacientes con propiedades clínicas coherentes, validando correlaciones biomédicas relevantes y garantizando un *data quality score* de 0.95.
4. Se garantizó costo-eficiencia con un presupuesto de $18.17 mensuales en desarrollo, 96 % menor que soluciones comparables.
5. Se aseguró reproducibilidad mediante Infrastructure as Code, facilitando el despliegue en múltiples entornos (dev, stg, prod).

9.2 Reflexión final y trabajo futuro

El presente trabajo ha mostrado la viabilidad de combinar *pipelines* de ingesta, validación y consulta con modelos biomédicos BERT y bases de datos vectoriales para implementar sistemas RAG aplicados al dominio médico. No obstante, los resultados alcanzados representan apenas un primer paso dentro de un campo en rápida evolución.

De cara al futuro, los principales retos se concentran en tres ejes:

- **Escalabilidad y fuentes de datos:** la integración con repositorios adicionales (p. ej., ClinicalTrials.gov, Cochrane) y la ingesta de modalidades diversas (imágenes médicas, ECG, notas clínicas) exigirán arquitecturas más distribuidas y eficientes.
- **Calidad y gobernanza del conocimiento:** la incorporación de *knowledge graphs*, técnicas de fine-tuning especializado y mecanismos de validación clínica rigurosa será fundamental para garantizar la fiabilidad de los resultados en entornos hospitalarios y regulados.
- **Evolución arquitectónica:** el avance hacia federated learning, despliegues multilingües y conformidad regulatoria (FDA, CE) marcará el camino hacia plataformas RAG biomédicas robustas y globales. Será clave analizar cómo equilibrar rendimiento, costos, seguridad y privacidad en infraestructuras cloud-native y en futuros *Patient Data Networks (PDN)*.

En conclusión, este trabajo constituye una base inicial que abre la puerta al diseño de sistemas médicos de próxima generación basados en RAG. Su consolidación requerirá no solo mejoras técnicas en modelos y bases vectoriales, sino también la resolución de retos clínicos, regulatorios y éticos que acompañan a la IA en salud.

A. Anexos

A.1 Repositorio de código

El código fuente completo está disponible públicamente, todo el detalle del repositorio se puede encontrar en el archivo README: https://github.com/Brayantcw/TFM-MDE-BRAYAN-TORRES

A.2 Desarrollo de la aplicación Streamlit

La interfaz de usuario se implementó con **Streamlit**, proporcionando un entorno interactivo que conecta al usuario final con el motor de búsqueda semántica en `Weaviate` y con un `LLM` desplegado en `LM Studio`. El objetivo es ofrecer un flujo *end-to-end* para recuperación aumentada con generación (RAG), accesible a investigadores y profesionales de la salud.

A.2.1 Arquitectura de la aplicación

La aplicación se estructura en tres elementos principales: (i) un **panel lateral de configuración**, (ii) pestañas funcionales diferenciadas para literatura y pacientes, y (iii) un motor de análisis LLM con auditoría clínica opcional.

Panel de configuración lateral: El *sidebar* permite parametrizar dinámicamente:

- Conexión a Weaviate (host, puerto gRPC, timeouts).
- Modo de búsqueda: vectorial, BM25 o híbrida (ajustable con parámetro α).
- Configuración de LLM vía LM Studio (endpoint, modelo cargado, temperatura, tokens máximos).
- Activación de un reranker local (`cross-encoder/ms-marco-MiniLM-L-6-v2`) para refinar la relevancia en papers.

Pestañas funcionales:

- **Artículos:** permite realizar consultas médicas en lenguaje natural. El sistema recupera papers relevantes desde `MedicalResearch`, aplica búsqueda semántica (vector, BM25 o híbrida), y sintetiza evidencia mediante LM Studio, mostrando además los metadatos (título, PMID, revista, resumen).

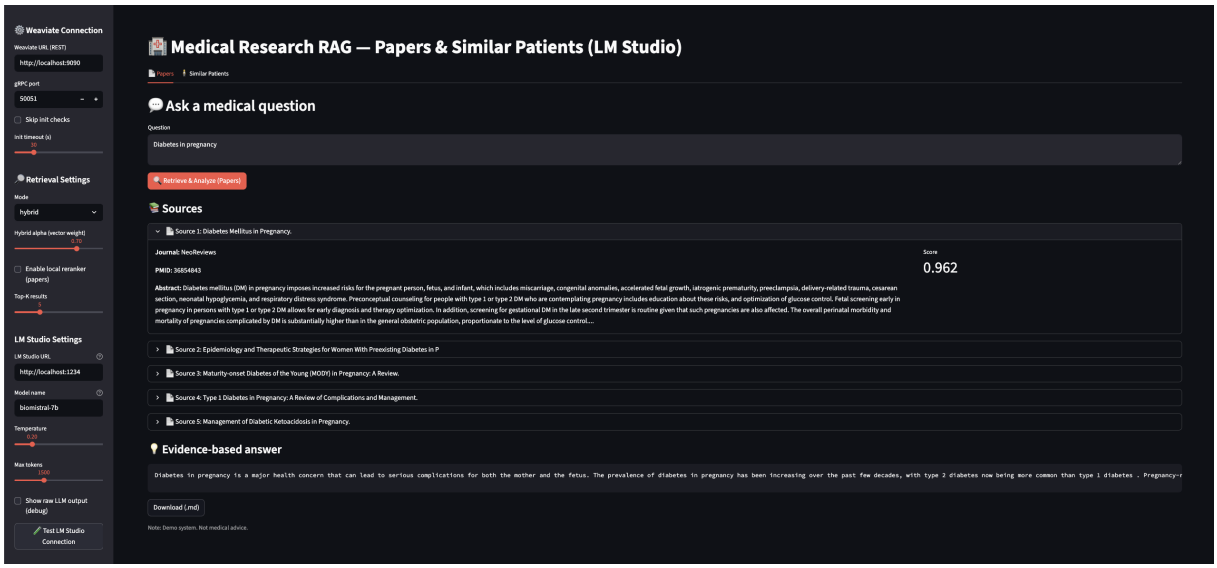

Figura A.1: Interfaz principal de Streamlit mostrando búsqueda de artículos y configuración lateral.

- **Pacientes similares:** el usuario puede construir un perfil clínico (edad, género, HbA1c, eGFR, medicación, complicaciones) y recuperar cohortes con características similares desde la colección `DiabetesPatients`. Además, se ejecuta un módulo de **auditoría terapéutica** que aplica heurísticas clínicas para identificar *señales positivas, precauciones y contraindicaciones.*

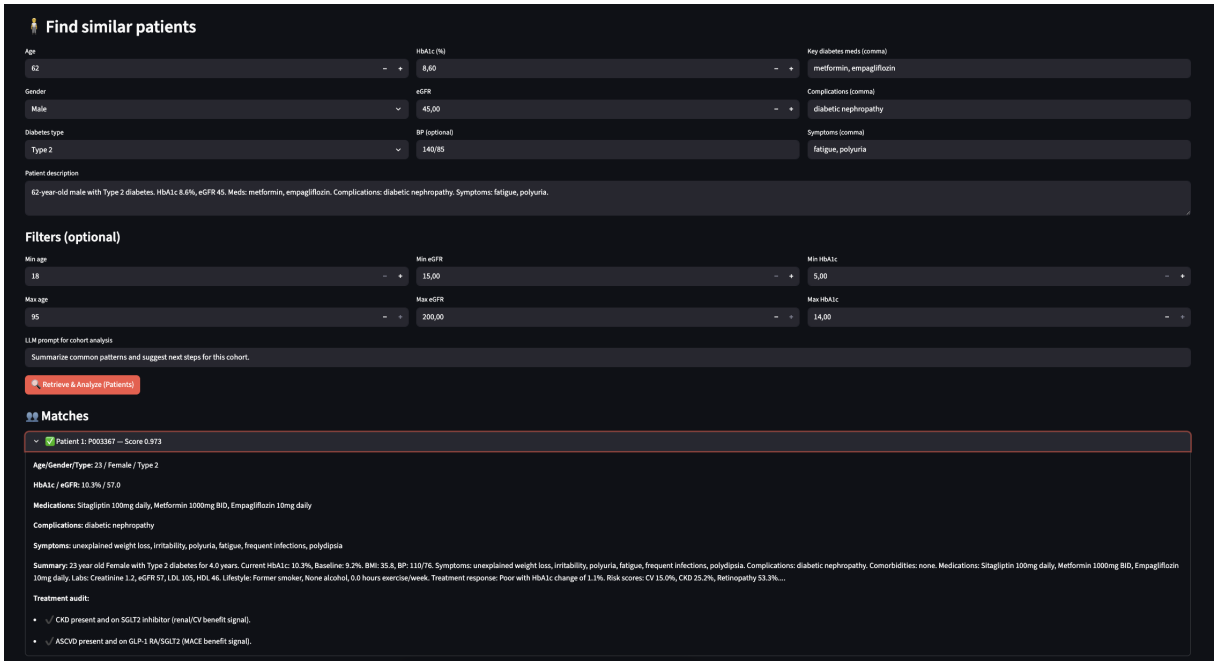

Figura A.2: Interfaz de Streamlit mostrando búsqueda de pacientes con parametros similares.

A.2.2 Funcionalidades implementadas

Búsqueda en literatura científica:

1. El usuario ingresa una pregunta en lenguaje natural.
2. La consulta se convierte en un embedding con `S-PubMedBert-MS-MARCO`.
3. Se ejecuta búsqueda vectorial, BM25 o híbrida en la colección de artículos.
4. Se presentan los documentos recuperados con sus puntuaciones de similitud.
5. El LLM (ejecutado en LM Studio) sintetiza la evidencia en lenguaje natural con citaciones en línea (`PMID:xxx`).

Análisis de pacientes similares:

1. Construcción de perfil: parámetros demográficos, clínicos y terapéuticos.
2. Generación automática de un *clinical summary* optimizado para búsqueda semántica.
3. Recuperación de cohortes similares con filtros avanzados (edad, HbA1c, eGFR).
4. Auditoría terapéutica con reglas predefinidas (ej., metformina en eGFR <30, TZD en insuficiencia cardíaca, SU en adultos mayores).
5. Narrativa complementaria del LLM para describir patrones de cohorte y posibles recomendaciones.

A.2.3 Implementación técnica

Gestión de estado y cache: La aplicación mantiene persistencia mediante `st.session_state`, evitando reconexiones innecesarias a Weaviate. Los resultados intermedios se cachean para reducir latencia en búsquedas repetidas.

Robustez y tolerancia a fallos: Se incluye verificación proactiva de conectividad antes de consultas pesadas. En caso de error con LM Studio, el sistema ofrece *fallbacks* (p. ej., mostrar únicamente resultados de recuperación). Los mensajes de error son contextuales y guían al usuario en la resolución de problemas.

A.2.4 Casos de uso validados

La aplicación soporta diversos escenarios:

- **Investigación bibliográfica:** consultas como "¿Cuáles son las contraindicaciones de metformina en enfermedad renal crónica?" generan recuperación de artículos relevantes y síntesis con referencias.
- **Soporte a decisión clínica:** identificación de pacientes similares con diabetes mal controlada y auditoría de tratamientos, destacando posibles contraindicaciones y recomendaciones.
- **Diseño de ensayos clínicos:** identificación de criterios de inclusión/exclusión a partir de cohortes sintéticas.
- **Educación médica:** exploración interactiva de literatura y casos clínicos como recurso docente.

A.3 Pipelines de validación

A.3.1 Pipeline de validación de literatura médica

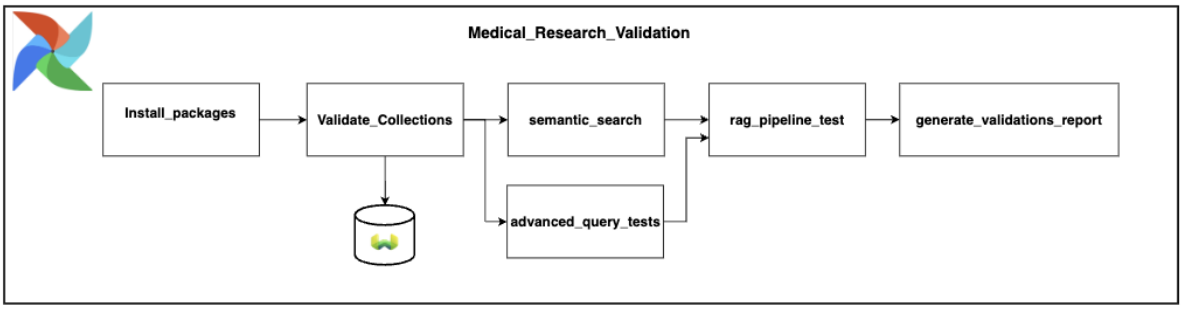

Figura A.3: Arquitectura del DAG de validación de literatura médica con procesamiento paralelo

Arquitectura

El DAG `medical_research_validation_v2` implementa un flujo de validación de cinco fases que combina preparación secuencial y ramificación paralela para evaluar la colección `MedicalResearch`.

Fase 1 – Preparación de entorno (`install_packages`): Instalación de dependencias (`sentence-transformers`, `tabulate`, compatibilidad con `S-PubMedBert-MS-MARCO`) necesarias para el análisis.

Fase 2 – Verificación de colección (`validate_collection`): Comprobación de integridad de la colección, metadatos completos, diversidad de fuentes y accesibilidad de índices vectoriales.

Fase 3 – Evaluación en paralelo:

- **Track A – Búsquedas semánticas (`semantic_search_tests`):** ejecución de consultas predefinidas (tratamiento, farmacología, comorbilidades, diagnóstico, prevención) evaluando relevancia y calidad de citaciones.

- **Track B – Consultas avanzadas (`advanced_query_tests`):** validación de consultas multi-concepto, temporales y farmacológicas específicas.

Fase 4 – Validación RAG (`rag_pipeline_test`): Simulación end-to-end de una pregunta clínica real para validar recuperación, extracción de contexto y síntesis con citaciones.

Fase 5 – Reporte consolidado (`generate_validation_report`): Generación de informe integrado con métricas de rendimiento, diversidad de fuentes y recomendaciones.

Contrato de validación

- Clave lógica: `pmid` (string).
- Campos requeridos: `title`, `abstract`, `publication_date`, `mesh_terms[]`.
- Salidas: variables de Airflow (`medical_stats`, `medical_validation_report`).

Operación

Ejecución automática tras ingestas de literatura, con resultados almacenados en Airflow y compatibilidad con monitorización externa (Azure Monitor, Grafana). La ejecución periódica asegura detección temprana de degradación de calidad o cambios en distribución de fuentes.

A.3.2 Pipeline de validación de pacientes sintéticos

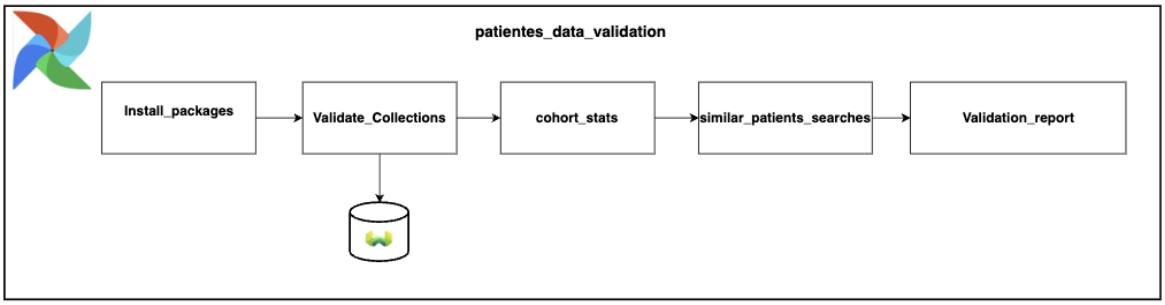

Figura A.4: Arquitectura del DAG de validación de pacientes sintéticos con análisis de cohortes y búsquedas semánticas

Arquitectura

El DAG `synthetic_patient_data_validation_v1` implementa un flujo lineal de cinco fases para evaluar la colección `DiabetesPatients`, garantizando coherencia clínica y utilidad para búsquedas vectoriales.

Fase 1 – Preparación de entorno (`install_packages`): Instalación de dependencias (`weaviate-client`, `numpy`, `pandas`).

Fase 2 – Verificación de colección (`validate_collection`): Chequeo de existencia del esquema, conteo total y muestreo de registros clave (edad, género, HbA1c, BMI, eGFR).

Fase 3 – Estadísticas de cohorte (`cohort_stats`): Cálculo de métricas demográficas y clínicas (distribución por género/tipo, promedios de HbA1c, BMI, eGFR, correlaciones HbA1c–glucosa y creatinina–eGFR).

Fase 4 – Validación de búsquedas (`similar_patients_searches`): Pruebas de consultas vectoriales con:

- `nearObject` básico desde un paciente ancla.
- `nearObject` con filtros (tipo de diabetes y género).
- Cohorte filtrada (T2DM, HbA1c 9.0 %, BMI 30).

Fase 5 – Reporte consolidado (`validation_report`): Consolidación en JSON con métricas poblacionales, ejemplos de similitud y recomendaciones automáticas.

Contrato de validación

- Clave lógica: `patient_id` (string).
- Campos requeridos: `age`, `gender`, `diabetes_type`, `hba1c_current`, `bmi`, `egfr`.
- Salidas: variables de Airflow (`diabetes_health`, `diabetes_stats`, `diabetes_similarity_examples`, `diabetes_validation_report`).

Operación

Ejecución automática tras cada ingesta de pacientes. Asegura que las distribuciones clínicas (p. ej., mayor BMI en T2DM vs T1DM, correlaciones esperadas HbA1c–glucosa y creatinina–eGFR) se mantengan realistas. Los resultados se utilizan para monitorización continua y detección temprana de incoherencias.

Bibliografía

[1] R. Kowsar y A. Mansouri, «Multi-level analysis reveals the association between diabetes, body mass index, and HbA1c in an Iraqi population,» *Scientific Reports*, 2022, DOI: 10.1038/s41598-022-25813-y.

[2] X. He et al., «Retrieval-Augmented Generation in Biomedicine: A Survey of Technologies, Datasets, and Clinical Applications,» *Preprint*, 2025, Actualizar con DOI o referencia final. dirección: https://arxiv.org/abs/2505.01146.

[3] P. Ke et al., «Retrieval-Augmented Generation for Large Language Models in Preoperative Assessment,» *npj Digital Medicine*, 2025, Actualizar con DOI. dirección: https://www.nature.com/.

[4] F. Liang et al., «RGAR: Recurrence Generation-augmented Retrieval for Factual-aware Medical Question Answering,» *Preprint*, 2025. dirección: https://arxiv.org/abs/2502.13361.

[5] J. Sohn et al., «Rationale-Guided RAG (RAG^2): A More Reliable RAG Framework for Biomedical Contexts,» en *Proceedings of NAACL*, 2025. dirección: https://aclanthology.org/.

[6] R. Stuhlmann et al., «Efficient and Reproducible Biomedical Question Answering using Retrieval Augmented Generation,» *Preprint*, 2025, Actualizar con DOI o referencia final. dirección: https://arxiv.org/abs/2505.07917.

[7] L. Zhang et al., «Leveraging Long Context in Retrieval-Augmented Language Workflows: BriefContext,» *npj Digital Medicine*, 2025, Actualizar con DOI. dirección: https://www.nature.com/.

[8] Y. Zhao et al., «MedRAG: Enhancing Retrieval-augmented Generation with Knowledge Graph-Elicited Reasoning for Healthcare Copilot,» *Preprint*, 2025. dirección: https://arxiv.org/abs/2502.04413.