TRABAJO DE FIN DE PROGRAMA

1. Descripción del caso de uso

1.1. Contexto y problemática del negocio o industria

En la industria financiera actual, y específicamente en los mercados de criptomonedas y ETFs (Exchange Traded Funds), los inversores se enfrentan a dos desafíos críticos: la **sobrecarga de información** y la **volatilidad extrema**.

- **Fragmentación de Datos:** Un analista o inversor debe consultar múltiples fuentes desconectadas: plataformas de precios (ej. Coin Gecko) para datos cuantitativos y portales de noticias para datos cualitativos. Integrar mentalmente el impacto de una noticia de última hora sobre un indicador técnico (como el RSI o medias móviles) es un proceso lento y propenso a sesgos cognitivos.
- **Barrera de Entrada Técnica:** El análisis técnico avanzado y el análisis de sentimiento automatizado han estado tradicionalmente reservados para fondos de cobertura (Hedge Funds) con infraestructura costosa. El inversor retail o el asesor financiero independiente carece de herramientas que combinen ambos mundos en tiempo real.
- **Dinámica del Mercado:** Los mercados financieros son entornos no estacionarios; las correlaciones que funcionan hoy pueden no funcionar mañana, lo que hace que los sistemas estáticos de reglas queden obsoletos rápidamente.

Problemática: La ausencia de una herramienta unificada que pueda "razonar" sobre datos estructurados (precios) y no estructurados (noticias) simultáneamente para ofrecer recomendaciones de inversión fundamentadas y rápidas.

1.2. Justificación del uso de IA Generativa y MLOps

La naturaleza del problema exige soluciones que vayan más allá del software tradicional:

- Por qué IA Generativa (GenAI):
    - **Análisis Semántico y Razonamiento:** A diferencia de los modelos predictivos tradicionales (que solo ven números), la IA Generativa (basada en GPT-4) tiene la capacidad de "leer" noticias financieras, interpretar el tono (sentiment analysis) y correlacionarlo semánticamente con el comportamiento técnico del activo.
    - **Interacción Natural (RAG):** La arquitectura RAG (Retrieval-Augmented Generation) permite al sistema consultar una base de conocimiento financiera actualizada y explicar su razonamiento al usuario en lenguaje natural, algo imposible para una "caja negra" de Deep Learning convencional.
- Por qué MLOps:
    - **Concept Drift (Desviación de los datos):** Los datos financieros cambian de distribución constantemente. MLOps es vital para monitorear cuándo los modelos de clasificación (ej. Random Forest para estrategias TOP/BOTTOM) pierden precisión y requieren reentrenamiento.

- **Reproducibilidad y Trazabilidad:** En un entorno financiero regulado o sensible, es crucial saber exactamente qué versión del modelo y qué dataset (versionado con DVC) generó una recomendación específica.
- **Orquestación:** La integración de múltiples APIs (CoinGecko, Alpha Vantage, Azure OpenAI) requiere un pipeline robusto de CI/CD para asegurar la disponibilidad del servicio.

1.3. Objetivo de negocio y objetivo técnico del proyecto

Objetivo de Negocio: Desarrollar un "Asistente de Inversión Inteligente" que democratice el acceso a estrategias de análisis institucional. El objetivo es reducir el tiempo de investigación de los usuarios en un **40-50%**, proporcionando análisis pre-digeridos que combinan indicadores técnicos y sentimiento de mercado, permitiendo decisiones de inversión más rápidas y menos emocionales.
Objetivos Técnicos:

- **Implementar una arquitectura RAG** sobre Azure OpenAI que integre datos de series temporales (estructurados) con noticias financieras (no estructurados) para mejorar la capacidad de razonamiento del modelo fundacional.
- **Desplegar un pipeline de MLOps completo** (usando MLflow y DVC) que automatice la ingesta diaria de datos de CoinGecko/Alpha Vantage, el cálculo de indicadores (RSI, SMA) y el reentrenamiento/monitoreo de los modelos de clasificación.
- **Desarrollar una interfaz interactiva** que permita a los usuarios consultar sobre activos específicos (ej. SPY, Bitcoin) y obtener respuestas fundamentadas en datos de los últimos 30 días.

2. Selección de modelo y datos

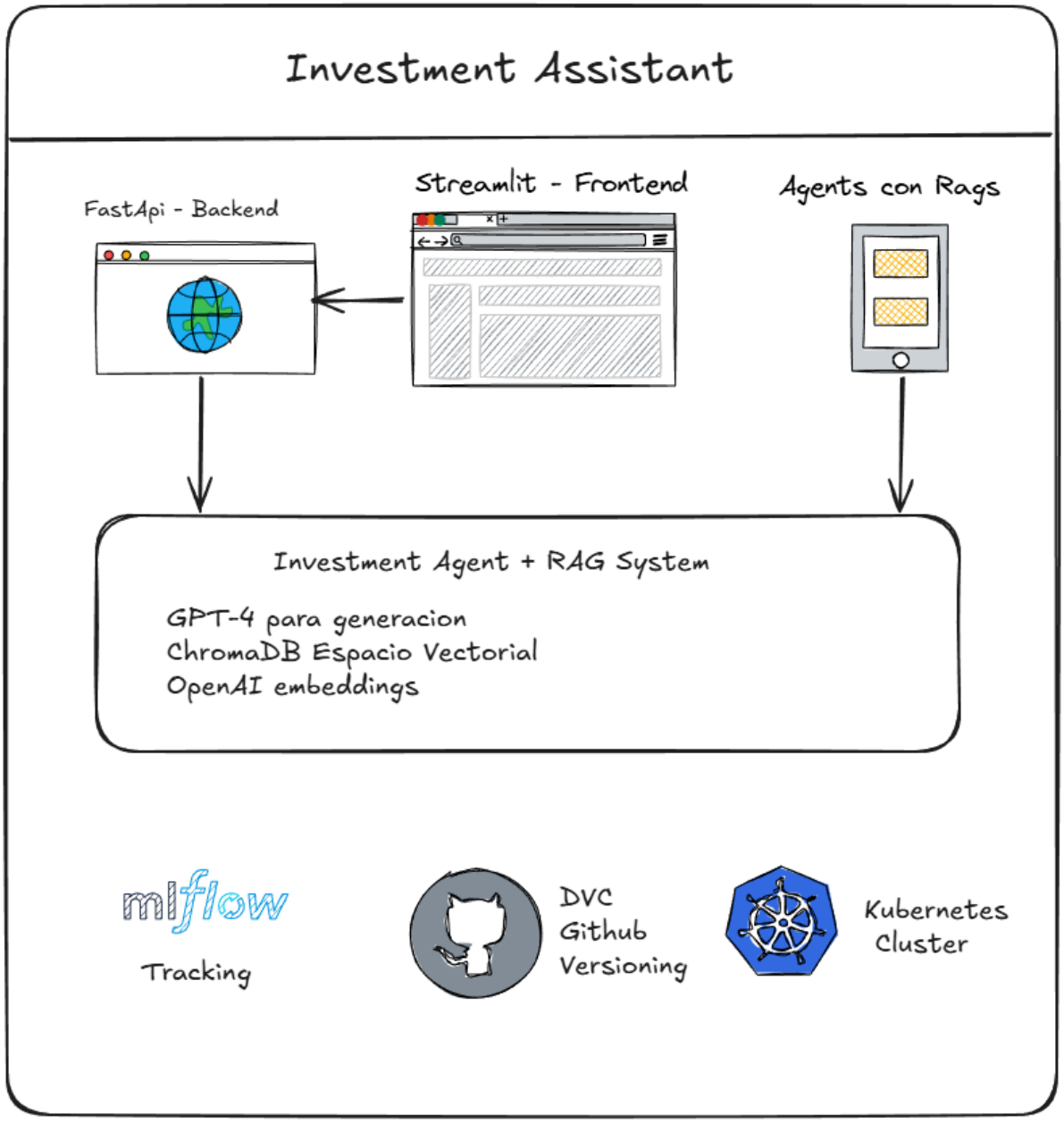
2.1. Elección del Modelo Generativo

La evaluación del modelo generativo fundacional basa su arquitectura en GPT-4 de Open AI,un modelo de lenguaje grande (LLM) generativo fundacional transformer, dado que concierte algunos criterios de seleccion como:

- Capacidad de razonamiento superior: GPT-4 demuestra capacidades avanzadas de razonamiento que son esenciales para análisis financiero complejo.
- Contexto extendido: Soporta contextos de hasta 128k tokens, permitiendo análisis de múltiples fuentes de datos simultáneamente.
- Disponibilidad comercial: Disponible a través de API de Azure OpenAI y OpenAI, facilitando integración en producción.
- Pre-entrenamiento financiero: Aunque no específicamente entrenado en finanzas, el modelo general muestra comprensión sólida de conceptos financieros.
- Documentación y soporte: Excelente documentación y comunidad de soporte.
- Modelos Complementarios:
    - Embeddings: OpenAI *text-embedding-3-small* para RAG (1536 dimensiones)

- Modelos de clasificación: Random Forest (scikit-learn) para estrategias TOP/BOTTOM

2.2. Arquitectura y características principales



Características Principales:

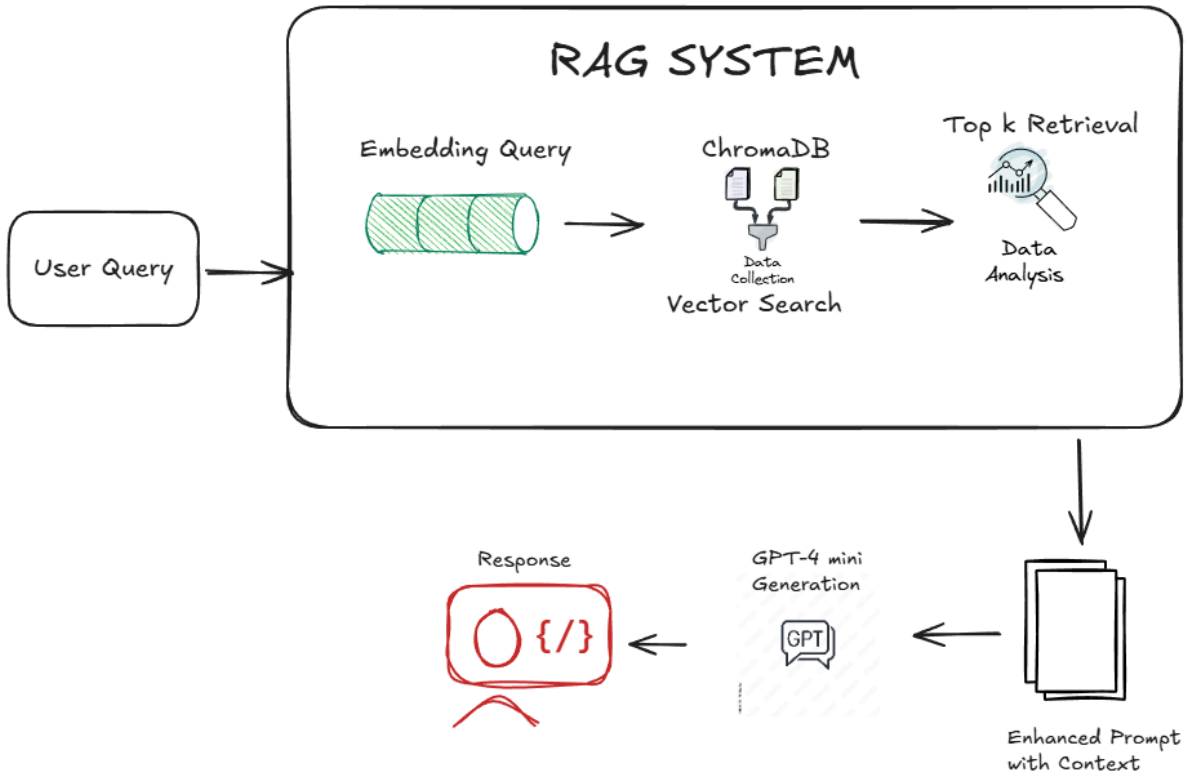
Arquitectura de Agentes con RAG:

Sistema de recuperación de contexto relevante desde knowledge base vectorial
Enriquecimiento de prompts con información financiera especializada
Historial de conversación para contexto continuo

Pipeline MLOps:

Versionado de datos y modelos con DVC
Tracking de experimentos con MLflow
Monitoreo de drift y métricas en tiempo real
Análisis Multi-dimensional:

Datos estructurados: precios, indicadores técnicos
Datos no estructurados: noticias, análisis de sentimiento
Recomendaciones combinando ambos tipos de datos



2.3. Dataset Utilizado

Fuentes de Datos:

APIs de Mercado (Abiertas):

CoinGecko API (Criptomonedas): Datos gratuitos y sin autenticación
Precios históricos diarios (30 días)
Volumen, capitalización de mercado
Alpha Vantage API (ETF): Datos gratuitos con API key
Time series diarios de ETFs (SPY, QQQ, VTI, etc.)
Indicadores técnicos calculados
Datos Sintéticos/Calculados:

Indicadores técnicos: Calculados a partir de datos de precio
RSI (Relative Strength Index)
Medias móviles (SMA 10, SMA 20)
Volatilidad (desviación estándar de retornos)
Posición de precio en rango 30 días
Knowledge Base (Propia):

Base de conocimiento financiero estructurada
Estrategias TOP/BOTTOM documentadas
Conceptos de análisis técnico
Mejores prácticas de inversión
Datos de Noticias (Entrada Manual/API):

Noticias financieras procesadas con Azure Text Analytics
Análisis de sentimiento por símbolo
Agregación temporal de sentimiento
Características del Dataset:

Volumen: ~1000+ registros por símbolo (30 días)
Frecuencia: Datos diarios
Período: Últimos 30 días (configurable)
Símbolos: SPY, QQQ, Bitcoin, Ethereum (extensible)

2.4. Preprocesamiento de datos y embeddings

Preprocesamiento de Datos:

Limpieza de datos:

```
# Eliminación de valores nulos
df = df.dropna(subset=["rsi", "sma_10", "sma_20"])

# Manejo de valores faltantes
X = df[features].fillna(0)
```

Cálculo de indicadores técnicos:

```
# RSI
delta = df["price"].diff()
gain = delta.where(delta > 0, 0).rolling(window=14).mean()
loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()
rsi = 100 - (100 / (1 + gain/loss))

# Medias móviles
sma_10 = df["price"].rolling(window=10).mean()
sma_20 = df["price"].rolling(window=20).mean()

# Volatilidad
```

```
volatility = df["price"].pct_change().std() * 100

# Posición en rango
price_position = ((current - low_30d) / (high_30d - low_30d)) * 100
```

Normalización:

Escalado de features para modelos de ML
Estandarización de rangos (0-100 para RSI, 0-100 para price_position)
Embeddings:

OpenAI Embeddings (text-embedding-3-small):

Dimensión: 1536
Modelo: OpenAI text-embedding-3-small
Uso: Vectorización de documentos en knowledge base para RAG
Vector Store (ChromaDB):

Almacenamiento persistente de embeddings
Búsqueda de similitud semántica
Retrieval de top-k documentos relevantes
Preprocesamiento de Texto para Embeddings:

```
# Text splitting
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=500,
    chunk_overlap=50
)

# Metadata enrichment
documents = [Document(content=text, metadata=metadata)
             for text, metadata in knowledge_base]
```

3. Ingeniería de prompts y adaptación

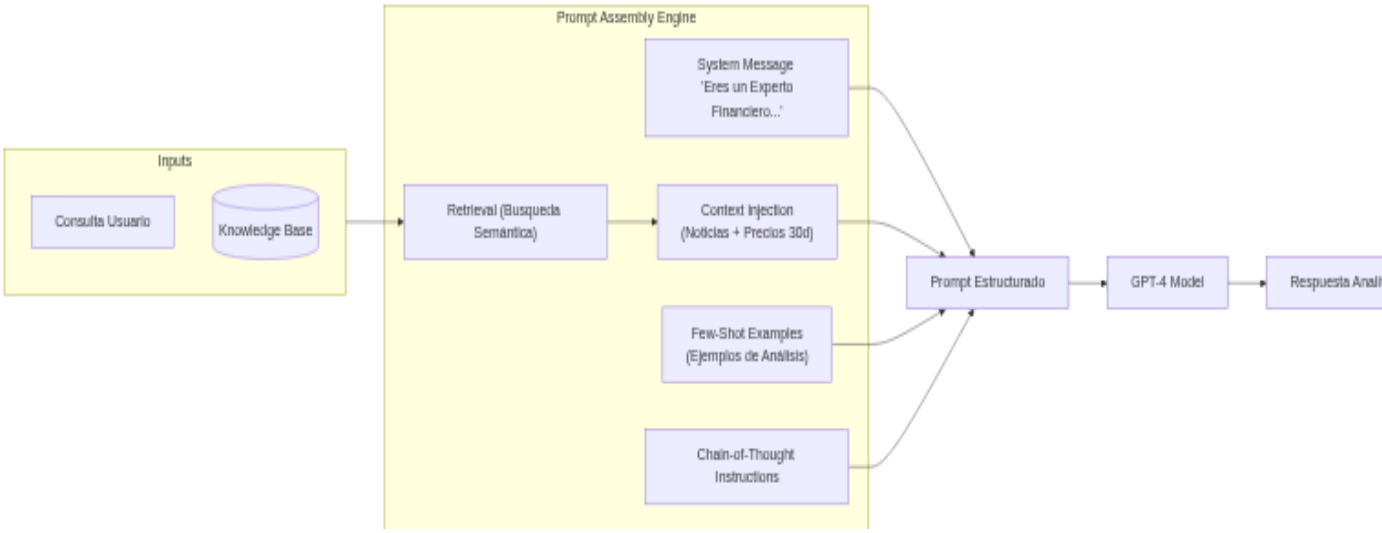
3.1. Estrategia de prompting

Para garantizar que **GPT-4** actúe como un analista financiero experto y evite alucinaciones, se ha diseñado una estrategia de prompting híbrida que combina varias técnicas avanzadas:

- **Persona Pattern (System Prompt):** Se define explícitamente el rol del modelo: *"Eres un Analista Financiero Senior experto en mercados de criptomonedas y ETFs. Tu objetivo es sintetizar datos técnicos y noticias para ofrecer una visión objetiva del mercado."* Esto acota el espacio de respuestas y ajusta el tono profesional.
- **RAG (Retrieval-Augmented Generation):** Dado que el modelo no tiene acceso a precios en tiempo real en su entrenamiento base, la estrategia principal es inyectar

en el prompt el contexto recuperado (precios de los últimos 30 días, noticias recientes y definiciones de la Knowledge Base) antes de solicitar la respuesta.

- **Chain-of-Thought (CoT):** Se instruye al modelo para "pensar paso a paso" antes de dar una conclusión.
    - *Paso 1:* Analizar la tendencia técnica (basada en RSI y SMAs proporcionados).
    - *Paso 2:* Analizar el sentimiento de las noticias recuperadas.
    - *Paso 3:* Identificar divergencias (ej. precio bajando pero noticias positivas).
    - *Paso 4:* Generar conclusión.
    - *Justificación:* Esta técnica ha demostrado aumentar significativamente la capacidad de razonamiento en tareas aritméticas y lógicas complejas (Wei et al., 2022).
- **Few-Shot Prompting:** Se incluyen en el prompt 1 o 2 ejemplos de análisis ideales (ej. estructura de salida deseada en JSON o formato Markdown específico) para guiar el estilo y la estructura de la respuesta sin necesidad de reentrenamiento.



3.2. Ajustes finos: Adaptación del modelo

Debido a la naturaleza altamente dinámica y cambiante de los mercados financieros, se ha optado por una estrategia de adaptación basada en contexto (In-Context Learning) en lugar de un *Fine-tuning* tradicional de los pesos del modelo LLM.

- Justificación de NO usar Fine-tuning/LoRA en el LLM: El "conocimiento" financiero (precios, noticias) caduca en cuestión de horas. Un modelo *fine-tuned* con datos de la semana pasada ya estaría obsoleto hoy. El costo computacional y temporal de reentrenar o ajustar adaptadores (LoRA) diariamente es ineficiente frente a la inyección dinámica de contexto mediante RAG.
- Modelo de Clasificación Auxiliar (Random Forest):
    - Aunque el LLM no se reentrena, el componente predictivo del sistema sí utiliza aprendizaje supervisado clásico.

    - Se entrena un modelo Random Forest (vía scikit-learn) utilizando las features generadas (RSI, Volatilidad, Posición en Rango) para clasificar oportunidades como "TOP" o "BOTTOM".
    - Este modelo es ligero, interpretable y se reentrena periódicamente dentro del pipeline de MLOps, actuando como un filtro previo de alta precisión antes de pasar los datos al LLM para su explicación narrativa.

3.3. Integración de datos externos

La capacidad del sistema para responder sobre eventos actuales depende de su integración con fuentes externas a través de una Base de Datos Vectorial.

- Pipeline de Ingesta y Vectorización:
    - Extracción: Las noticias y documentos de la Knowledge Base se extraen y limpian.
    - Chunking (Fragmentación): Se utiliza RecursiveCharacterTextSplitter con un tamaño de fragmento (*chunk size*) de 500 caracteres y un solapamiento (*overlap*) de 50 caracteres. Este tamaño es óptimo para capturar el contexto de una noticia financiera sin diluir la información específica.
    - **Embedding:** Los fragmentos de texto se convierten en vectores densos utilizando el modelo **text-embedding-3-small** de OpenAI (1536 dimensiones), elegido por su eficiencia y alto rendimiento en tareas de recuperación semántica multilingüe.
    - **Almacenamiento (Vector Store):** Los vectores se almacenan en **ChromaDB**.
- Recuperación (Retrieval):
    - Cuando el usuario realiza una consulta, esta se vectoriza y se buscan los k documentos más similares (top-k retrieval) en ChromaDB.
    - Se utiliza enriquecimiento de metadatos (símbolo del activo, fecha de publicación) para filtrar los resultados y asegurar que el modelo solo "lea" noticias relevantes al activo en cuestión y al periodo de tiempo analizado.

4. Implementación de la aplicación

4.1. Descripción del flujo de la solución

La lógica central de la aplicación se ha implementado utilizando LangGraph para crear un flujo de control cíclico y robusto, superando las limitaciones de las cadenas lineales simples (LangChain Chains). La arquitectura se define como un grafo de estado (StateGraph) donde un Agente Supervisor toma decisiones dinámicas sobre qué herramientas utilizar.

Componentes del Grafo:

- Estado del Agente (AgentState): Mantiene el historial de mensajes, la consulta actual del usuario y los datos intermedios recuperados (precios, noticias).
- Nodo Supervisor (LLM): Es el cerebro del sistema. Recibe el estado actual y decide si necesita:
    - Llamar a la herramienta de datos de mercado.
    - Llamar a la herramienta de noticias (RAG).
    - Llamar al modelo predictivo (Random Forest).
    - Finalizar y generar la respuesta.
- Nodos de Herramientas (Tools Nodes): Ejecutan funciones Python específicas y devuelven la salida al Supervisor.

Flujo de Ejecución: Inicio -> Supervisor -> *¿Necesita datos?* -> (SÍ) -> Ejecutar Herramientas -> Supervisor -> (NO) -> Generar Respuesta Final -> Fin.

4.2. Integración de componentes

El sistema integra múltiples subsistemas a través de APIs y contenedores, asegurando una comunicación fluida entre la interfaz de usuario, el motor de IA y las fuentes de datos.

- Interfaz de Usuario (Frontend):
    - Desarrollada en Streamlit (Python) por su capacidad de prototipado rápido de aplicaciones de datos.
    - Permite la selección de activos (Dropdown), configuración de parámetros (Horizonte temporal) y visualización de gráficos de precios (Plotly) junto con el análisis textual del LLM.
- Motor de Inferencia (Backend):
    - LLM: Conexión a Azure OpenAI Service (GPT-4) mediante la librería langchain-openai. Se utiliza Azure para garantizar cumplimiento normativo y seguridad empresarial.
    - Datos de Mercado: Integración con CoinGecko API (para criptos) y Alpha Vantage (para acciones/ETFs) mediante peticiones HTTP (requests). Los datos se procesan en DataFrames de pandas en memoria.
    - Modelo Predictivo: El modelo Random Forest entrenado se carga mediante joblib o mlflow.sklearn para realizar inferencias rápidas sobre los datos tabulares recién descargados.
- Base de Datos Vectorial:
    - ChromaDB actúa como memoria a largo plazo para documentos y noticias. Se ejecuta como un proceso persistente local o en contenedor separado, permitiendo consultas semánticas rápidas durante la ejecución del agente.

4.3. Diseño modular y escalable del sistema

El proyecto sigue principios de ingeniería de software como **Separation of Concerns (SoC)** para facilitar el mantenimiento y la escalabilidad futura.

- Estructura Modular del Código:

        - src/data_ingestion: Módulos encargados exclusivamente de conectar con APIs externas, limpiar JSONs y normalizar datos. Si cambia una API (ej. CoinGecko v3), solo se modifica este módulo.
        - src/models: Contiene la lógica de entrenamiento (Random Forest) y las definiciones de los Agentes (LangGraph).
        - src/rag: Gestiona la creación de embeddings y la conexión con ChromaDB.
        - src/app: Lógica de presentación (Streamlit).
- Estrategia de Escalabilidad:
    - **Contenedorización:** La aplicación completa se empaqueta en **Docker**. Esto elimina el problema de "funciona en mi máquina" y facilita el despliegue en cualquier nube (Azure Container Apps, AWS ECS, Hugging Face Spaces).
    - **Desacoplamiento del LLM:** Al usar la API de Azure OpenAI, la carga computacional pesada de la generación de texto se delega a la nube de Microsoft, permitiendo que el servidor de la aplicación sea ligero (solo orquestación y UI).
    - **Base Vectorial Escalable:** ChromaDB puede migrarse de una instancia local a una arquitectura cliente-servidor distribuida si el volumen de noticias crece exponencialmente.

5. Orquestación y despliegue

5.1. Creación de API para servir el modelo

Para desacoplar completamente la lógica del agente financiero de la interfaz de usuario, se ha expuesto el motor de inteligencia artificial a través de una API RESTful de alto rendimiento utilizando FastAPI.

- Elección de FastAPI: Se priorizó sobre Flask debido a su soporte nativo para operaciones asíncronas (async/await), lo cual es crucial cuando se orquestan llamadas concurrentes a modelos LLM y APIs de terceros (evitando cuellos de botella I/O). Además, genera automáticamente documentación interactiva (Swagger UI).

- Diseño de Endpoints:
    - POST /v1/chat/analyze: Endpoint principal que recibe el símbolo del activo (ej. "BTC") y devuelve el análisis completo generado por el agente LangGraph.
    - GET /v1/market/price/{symbol}: Endpoint ligero para obtener solo datos técnicos normalizados.
    - GET /health: Para verificaciones de estado (health checks) en el clúster de Kubernetes.

5.2. Contenerización con Docker

La portabilidad de la aplicación se garantiza mediante el uso de Docker. Se utiliza una estrategia de construcción en múltiples etapas (*multi-stage build*) para minimizar el tamaño de las imágenes y mejorar la seguridad.

- Dockerfile Optimizado:
    - *Base:* python:3.10-slim para reducir la superficie de ataque.
    - *Dependencias:* Instalación de librerías compiladas y limpieza de caché de pip en la misma capa para reducir el peso.
    - *Usuario:* Ejecución del contenedor como usuario no root (appuser) para mitigar riesgos de seguridad.
- Docker Compose: Se define un archivo docker-compose.yml para orquestar localmente los servicios: el contenedor de la API, el contenedor de ChromaDB y el contenedor de Streamlit durante el desarrollo.

5.3. Automatización con CI/CD (GitHub Actions)

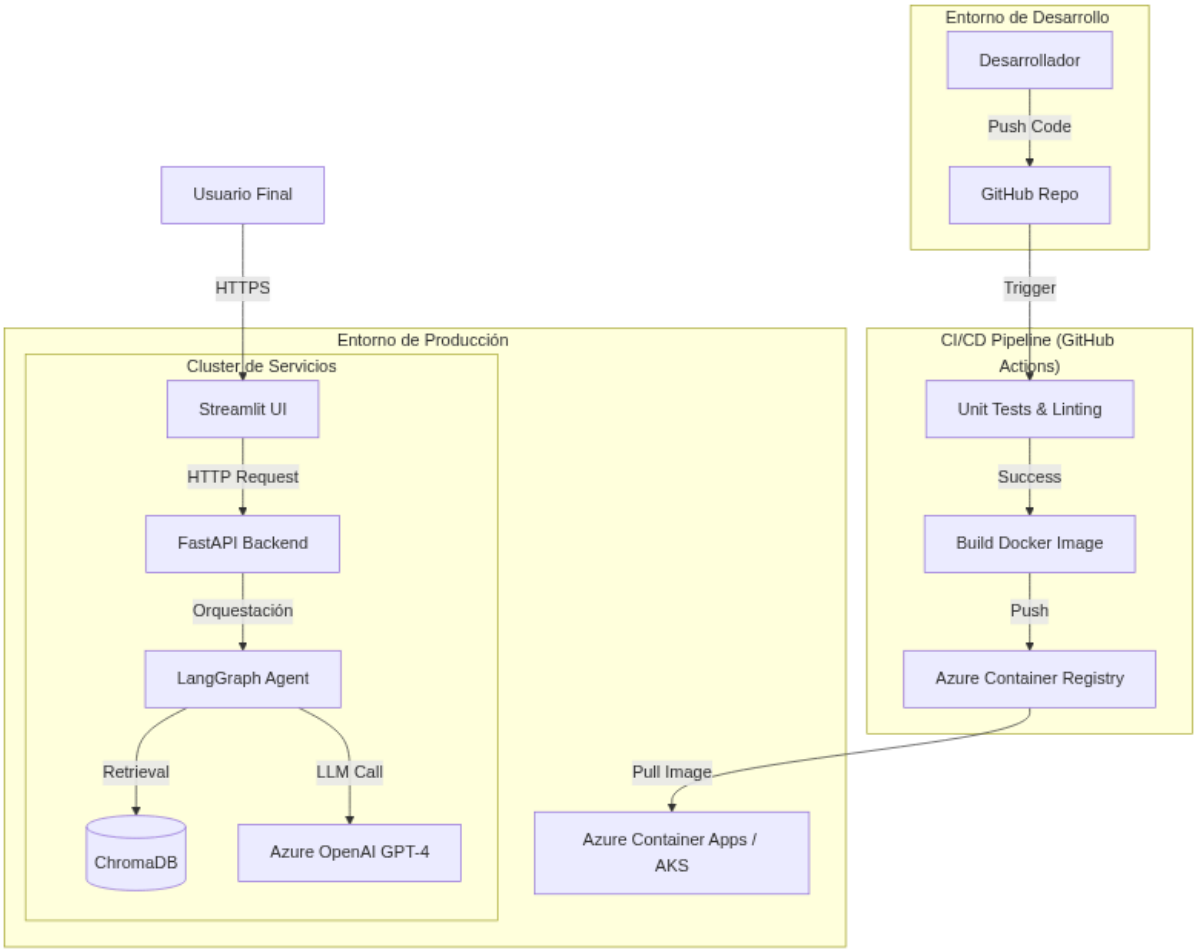
El ciclo de vida del desarrollo se automatiza mediante un pipeline de CI/CD en GitHub Actions, asegurando que solo código validado llegue a producción.

- Workflow de Integración Continua (CI):
    - Linting & Formatting: Verificación automática de estilo de código con Ruff y Black.
    - Unit Testing: Ejecución de pruebas unitarias con pytest para validar la lógica de cálculo de indicadores financieros.
- Workflow de Despliegue Continuo (CD):
    - Build & Push: Al hacer merge a la rama main, se construye la imagen Docker y se sube a Azure Container Registry (ACR).
    - Deploy: Se dispara un evento que actualiza la revisión del servicio en el entorno de nube.

5.4. Despliegue en Kubernetes - Cloud

Para un entorno de producción escalable y resiliente, se ha diseñado una arquitectura basada en Azure Kubernetes Service (AKS), aunque para la fase de prototipo (MVP) se utiliza Azure Container Apps por su simplicidad serverless.

- Estrategia de Despliegue:
    - El servicio de inferencia (FastAPI) escala horizontalmente basado en el uso de CPU/Memoria.
    - Se utilizan Secretos de Kubernetes (o Azure Key Vault) para gestionar de forma segura las API Keys de OpenAI y Alpha Vantage, evitando que estas queden expuestas en el código fuente.


Entorno de Desarrollo
Desarrollador
Push Code
GitHub Repo
Usuario Final
HTTPS
Trigger
Entorno de Producción
Cluster de Servicios
CI/CD Pipeline (GitHub Actions)
Streamlit UI
Unit Tests & Linting
HTTP Request
Success
FastAPI Backend
Build Docker Image
Orquestación
Push
LangGraph Agent
Azure Container Registry
Retrieval
LLM Call
Pull Image
ChromaDB
Azure OpenAI GPT-4
Azure Container Apps / AKS