Universidad Rey Juan Carlos

TEMA 6 – HOJA DE EJERCICIOS II

Se plantean diferentes ejercicios basados en técnicas simples de RAG.

Objetivo: comprender RAG a nivel de arquitectura y construir un RAG funcional en Python con LangChain.

Requisitos y entorno

Los ejercicios prácticos están pensados para ejecutarse en un entorno Python 3.12+.

Instalación mínima (elige UNA opción de LLM):

```
# Opción A: OpenAI (requiere OPENAI_API_KEY)
pip install -U langchain langchain-core langchain-community langchain-openai
langchain-chroma chromadb tiktoken

# Opción B: Ollama local (sin API key, requiere Ollama instalado)
pip install -U langchain langchain-core langchain-community langchain-ollama
langchain-chroma chromadb tiktoken

# Embeddings locales recomendados (sin token HF)
pip install -U langchain-huggingface sentence-transformers
```

Se permite usar cualquier corpus pequeño (p.ej., 2–5 documentos TXT/PDF), pero se recomienda preparar un directorio ./data con 3–6 ficheros (se proporcionan algunos de base pero se pueden ampliar con los que se deseen).

Bloque A — Teoría (sin código)

Ejercicio A1 — Pipeline RAG y puntos de fallo

Explica el pipeline RAG completo (ingestión → chunking → embeddings → vector store → retrieval → prompting → generación) e identifica al menos 4 puntos donde el sistema puede fallar (p.ej., chunking mal calibrado, retrieval irrelevante, etc.). Para cada punto de fallo, propone una mitigación concreta.

Ejercicio A2 — RAG vs Fine-Tuning vs Prompting

Para un asistente sobre normativa universitaria interna (documentos privados que cambian cada semestre): (1) ¿Cuándo elegirías RAG? (2) ¿Cuándo elegirías fine-tuning? (3) ¿Qué papel juega el prompt engineering en ambos? Da una respuesta razonada (máx. 12 líneas).

Ejercicio A3 — Evaluación de RAG

Propón un plan de evaluación con al menos 3 métricas o criterios: (i) métricas de retrieval (p.ej., Recall@k), (ii) métricas de respuesta (p.ej., exactitud/fidelidad), (iii) métricas de robustez (p.ej., sensibilidad a perturbaciones). Indica cómo obtendrías datos de evaluación (conjunto de preguntas, gold answers, etc.).

Bloque B — Práctica (Python + LangChain)

Ejercicio B1 — RAG mínimo funcional (TXT → Chroma → QA)
Construye un RAG mínimo que:

1) Cargue varios .txt desde ./data

2) Divida en chunks con RecursiveCharacterTextSplitter

3) Genere embeddings (HuggingFaceEmbeddings recomendado)

4) Indexe en Chroma (persistente)

5) Responda una pregunta con create_retrieval_chain

Requisitos:

- El LLM debe responder SOLO con base en el contexto recuperado.
- La respuesta debe incluir una sección 'Fuentes' con los metadatos (p.ej., filename) de los chunks usados.

Ejercicio B2 — Mejora del retrieval (MMR + filtros + k)
Partiendo de B1, implementa mejoras y compara resultados:

- Cambia a búsqueda MMR (max marginal relevance) para diversificar chunks.
- Añade metadatos (source, section) en Document y filtra por source.
- Compara k=2 vs k=5 y discute trade-off entre cobertura y ruido.

Entrega: fragmento de código + breve análisis (10–12 líneas).

Ejercicio B3 — RAG conversacional (historial → mejor consulta)
Implementa una variante conversacional donde el sistema genere una 'search query' mejor usando el historial. Puedes resolverlo con un paso previo que reescriba la pregunta (query rewriting) usando el LLM y luego llame al retriever.

Requisitos:

- Mantener un historial simple (lista de turnos).
- Mostrar por pantalla la query final usada para retrieval.
- Demostrar con un ejemplo que mejora sobre usar la pregunta original.