UNIVERSIDAD DE CHILE

Inteligencia Artificial Generativa

Let's talk about hype stuff

Valentin Barriere // Clemente Henriquez

Universidad de Chile – DCC

Diplomado de Postítulo en Inteligencia Artificial, Primavera 2025

RAG

Outline : El Problema del Conocimiento Congelado

El Problema del Conocimiento Congelado

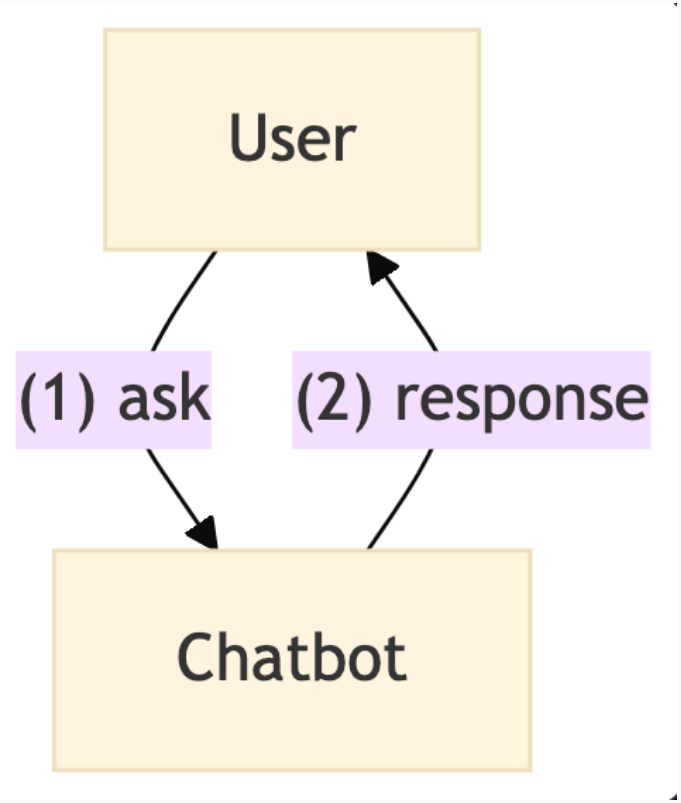

Figure 1: Ilustración Interacción Básica con un LLM. Fuente: [1]

Limitaciones de los LLMs Puros

Los modelos de lenguaje grandes enfrentan limitaciones fundamentales relacionadas con su conocimiento estático:

- **Knowledge cutoff**: El conocimiento del modelo está congelado en su fecha de entrenamiento
- **Datos privados inaccesibles**: No pueden acceder a información específica de organizaciones o dominios
- **Alucinaciones**: Tendencia a generar información plausible pero incorrecta cuando no conocen un dato
- **Costo prohibitivo**: Re-entrenar modelos para actualizar conocimiento cuesta millones de USD

El dilema fundamental

¿Cómo proporcionar a un LLM acceso a información actualizada y específica sin re-entrenamiento completo?

Continual Pre-Training

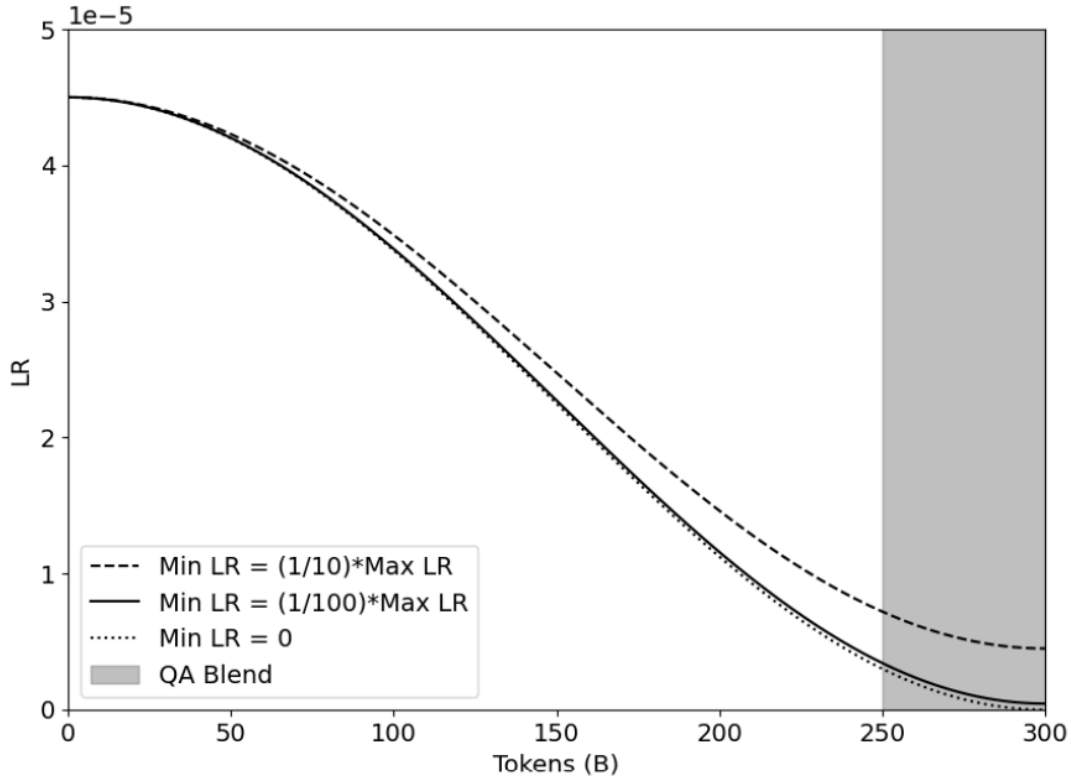

Figure 3: Cosine decay schedules with a Max LR of $4.5e$-5. Each schedule differently prioritizes LR magnitude and slope of decay.

Figure 2: Fuente: [2]

¿Por qué no simplemente fine-tuning?

El fine-tuning presenta limitaciones significativas para mantener conocimiento actualizado:

Limitaciones técnicas:

- **Costo computacional**: Millones de USD por iteración de entrenamiento
- **Tiempo de entrenamiento**: Semanas o meses para modelos grandes
- **Catastrophic forgetting**: El modelo olvida conocimiento previo al aprender nuevo
- **Dificultad de actualización**: Cambiar un dato específico requiere re-entrenar todo

Ejemplo: Actualizar un precio

Para actualizar el precio de Bitcoin en un LLM mediante fine-tuning, necesitaríamos re-entrenar el modelo completo. Con RAG, simplemente actualizamos el documento en la base de datos.

Outline : Fundamentos Históricos: Information Retrieval Clásico

Information Retrieval: La Era Pre-Neural (1960-2010)

¿Qué es Information Retrieval?

IR es la disciplina que busca identificar, dentro de grandes colecciones, la información no estructurada que responde a una necesidad del usuario.

Trabaja con contenido sin esquema fijo —principalmente texto— y se centra en cómo representar, organizar y acceder a esa información de forma eficiente.

Ejemplos clásicos:

- Buscadores web (Google, Bing)
- Búsqueda de documentos legales o científicos
- Gestión de contenido empresarial
- Motores de recomendación

Datos Estructurados vs No Estructurados

¿Cuál es la diferencia fundamental?

Datos Estructurados

- Esquema fijo y predefinido
- Bases de datos relacionales
- Queries exactas (SQL)
- Ejemplo: `SELECT * WHERE age > 25`

Datos No Estructurados

- Sin esquema fijo
- Texto libre, documentos
- Queries en lenguaje natural
- Ejemplo: "artículos sobre diabetes"

El desafío de IR

Encontrar información relevante cuando no hay estructura ni match exacto posible. ¿Cómo medir la relevancia de un documento D para una query Q?

Boolean Retrieval Model (1960s)

El primer modelo formal de IR: documentos recuperados mediante expresiones booleanas.

Idea central:

- Cada documento es un conjunto de términos
- Query = expresión lógica con operadores AND, OR, NOT
- Un documento es recuperado si y solo si satisface la expresión

Ejemplo

Query: `(machine AND learning) OR (deep AND neural)`

Recupera documentos que contengan:

- "machine" Y "learning", O
- "deep" Y "neural"

Característica clave: Es un modelo de *exact match* — no hay ranking, solo match/no-match.

Boolean Retrieval: Ejemplo con El Señor de los Anillos

Colección de 5 documentos:

Doc	Contenido
D1	Frodo y Sam viajan a Mordor con el anillo
D2	Aragorn lidera la batalla en Minas Tirith
D3	Gandalf enfrenta al Balrog en Moria
D4	Frodo destruye el anillo en Monte del Destino
D5	Aragorn se convierte en rey de Gondor

Queries y resultados:

- `Frodo AND anillo` $\rightarrow \{D1, D4\}$
- `Aragorn OR Gandalf` $\rightarrow \{D2, D3, D5\}$
- `Frodo AND NOT Sam` $\rightarrow \{D4\}$
- `anillo AND Mordor` $\rightarrow \{D1\}$

Boolean Retrieval: Ventajas y Limitaciones

Ventajas:

- **Simple y eficiente**: Operaciones de conjuntos muy rápidas
- **Determinista**: Mismo query $\rightarrow$ mismos resultados
- **Control preciso**: Usuarios expertos pueden hacer queries complejas
- **Escalable**: Inverted index permite búsqueda en millones de docs

Limitaciones fundamentales:

- **Sin ranking**: Todos los resultados tienen igual importancia
- **Todo-o-nada**: Un doc es relevante o no, sin grados
- **Difícil para usuarios**: Expresiones booleanas requieren expertise
- **Vocabulary mismatch**: "automobile" vs "car" $\rightarrow$ no match

El problema clave

¿Cómo rankear resultados por relevancia cuando hay cientos de matches?

Evolución de IR: Paradigmas Fundamentales

Boolean retrieval fue solo el comienzo. La evolución de IR pre-neural:

Paradigmas fundamentales (1960-2010):

- **Boolean Retrieval (1960s)**: Queries como expresiones lógicas (AND, OR, NOT)
- **Vector Space Model (Salton, 1975)**: Representar documentos y queries como vectores en espacio de términos
- **Probabilistic Models (1970s-1990s)**: BM25, Language Models for IR
- **Latent Semantic Analysis (1990)**: Reducción de dimensionalidad con SVD

La pregunta que los motivó a todos

¿Cómo medir la relevancia de un documento D para una query Q de forma graduada, no binaria?

Vector Space Model (VSM)

El VSM (Salton, 1975) representa documentos y queries como vectores en un espacio de términos de alta dimensionalidad.

Formulación matemática:

Sea $\mathcal{V} = \{t_1, t_2, \ldots, t_n\}$ el vocabulario. Un documento D se representa como:

$$D = (w_{1,D}, w_{2,D}, \ldots, w_{n,D})$$

donde $w_{i,D}$ es el peso del término t_i en el documento D.

Esquema TF-IDF clásico:

$$w_{i,D} = \text{tf}(t_i, D) \times \text{idf}(t_i)$$

- $\text{tf}(t_i, D) = \frac{\text{count}(t_i, D)}{\text{total tokens en } D}$ (term frequency)
- $\text{idf}(t_i) = \log \frac{N}{\text{df}(t_i)}$ (inverse document frequency)
- N = número total de documentos, $\text{df}(t_i)$ = documentos que contienen t_i

Similitud Coseno en VSM

La relevancia se mide mediante similitud coseno entre vectores:

$$\text{sim}(Q, D) = \frac{Q \cdot D}{\|Q\|\|D\|} = \frac{\sum_{i=1}^{n} w_{i,Q} \cdot w_{i,D}}{\sqrt{\sum_{i=1}^{n} w_{i,Q}^2} \cdot \sqrt{\sum_{i=1}^{n} w_{i,D}^2}}$$

Ejemplo simple

Vocabulario: {`machine, learning, deep, neural`}

`Query`: "machine learning" $\rightarrow (1, 1, 0, 0)$

`Doc1`: "machine learning basics" $\rightarrow (0.5, 0.5, 0, 0)$

`Doc2`: "deep neural networks" $\rightarrow (0, 0, 0.7, 0.7)$

$\text{sim}(Q, \text{Doc1}) = 1.0 \quad \text{sim}(Q, \text{Doc2}) = 0.0$

Ventaja sobre Boolean: Ranking graduado por similitud.

BM25: Best Match 25

BM25 (Robertson & Walker, 1994) es el algoritmo probabilístico que dominó IR por décadas y sigue siendo relevante hoy.

$$\text{score}(D, Q) = \sum_{t_i \in Q} \text{IDF}(t_i) \cdot \frac{f(t_i, D) \cdot (k_1 + 1)}{f(t_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Componentes:

- $f(t_i, D)$: frecuencia del término t_i en documento D
- $|D|$: longitud del documento D
- avgdl: longitud promedio de documentos en la colección
- $k_1 \in [1.2, 2.0]$: parámetro de saturación de frecuencia
- $b \in [0, 1]$: parámetro de normalización por longitud (típicamente 0.75)

BM25: Intuición y Componente IDF

Intuición general:

BM25 balancea la frecuencia del término con normalización por longitud de documento, evitando ventaja injusta para documentos largos.

El componente IDF (Inverse Document Frequency):

$$\text{IDF}(t_i) = \log \frac{N - n(t_i) + 0.5}{n(t_i) + 0.5}$$

donde:

- N: número total de documentos
- $n(t_i)$: número de documentos que contienen t_i

Intruicion BM25

Ejemplo numérico

Supongamos $N = 10,000$ documentos:

- Término común "the": $n(\text{the}) = 9,000 \Rightarrow \text{IDF} = 0.10$ (bajo peso)
- Término raro "photosynthesis": $n(\text{photo}) = 50 \Rightarrow \text{IDF} = 5.29$ (alto peso)

Los términos raros aportan más a la relevancia que los términos comunes.

Limitaciones del IR Clásico

Los métodos clásicos de IR (TF-IDF, BM25) enfrentaban problemas fundamentales:

1. **Vocabulary Mismatch Problem:**
   - Query: "automobile" vs. Documento: "car" $\rightarrow$ similitud $= 0$
   - No captura sinónimos ni relaciones semánticas
2. **Lexical Gap:**
   - Dependencia estricta de coincidencia léxica exacta
   - Query: "heart attack" vs. "myocardial infarction" $\rightarrow$ no match
3. **Bag-of-Words Assumption:**
   - Ignora orden de palabras y estructura sintáctica
   - "dog bites man" vs. "man bites dog" $\rightarrow$ mismo vector
4. **Polisemia no resuelta:**
   - "bank" (institución financiera vs. orilla de río) tratado idénticamente

Latent Semantic Analysis (LSA)

LSA (Deerwester et al., 1990) intentó resolver el vocabulary mismatch usando álgebra lineal:

Idea central: Reducir dimensionalidad del espacio de términos usando SVD (Singular Value Decomposition).

Dada una matriz término-documento $X \in \mathbb{R}^{n \times m}$:

$$X = U \Sigma V^T$$

donde:

- $U \in \mathbb{R}^{n \times k}$: términos en espacio latente
- $\Sigma \in \mathbb{R}^{k \times k}$: valores singulares (importancia de cada dimensión)
- $V \in \mathbb{R}^{m \times k}$: documentos en espacio latente
- $k \ll n$: número reducido de dimensiones (típicamente 100-300)

Ventaja de LSA

En el espacio latente, "car" y "automobile" tienen representaciones similares incluso sin co-ocurrir en el mismo documento.

De LSA a Neural Embeddings

Limitaciones de LSA:

- Computacionalmente costoso para grandes colecciones (SVD es $O(mn^2)$)
- Interpretabilidad limitada de dimensiones latentes
- Dificultad para incorporar nuevos documentos (requiere re-computar SVD)

La transición neural (2013-2015):

- **Word2Vec (Mikolov et al., 2013)**: Embeddings densos mediante predicción de contexto
- **GloVe (Pennington et al., 2014)**: Factorización de matriz de co-ocurrencias
- **Doc2Vec (Le & Mikolov, 2014)**: Extensión a nivel documento

El salto conceptual

De representaciones *sparse* basadas en conteos (TF-IDF, BM25) a representaciones *dense* aprendidas (neural embeddings). Esto permitió capturar similitud semántica sin coincidencia léxica.

Outline : RAG: Uniendo LLMs e Information Retrieval

Recordatorio LLM - El Problema del Conocimiento Congelado

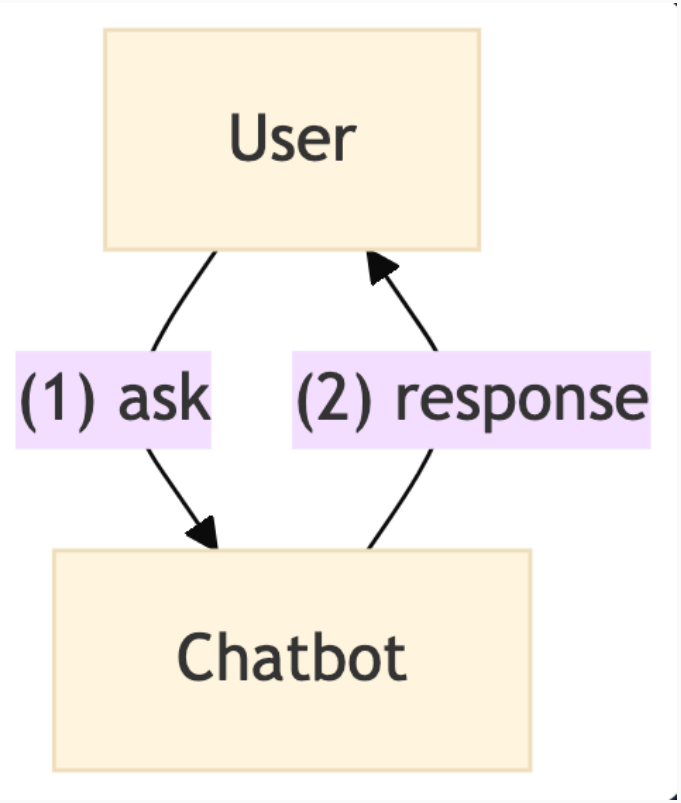

Figure 3: Ilustración Interacción Básica con un LLM. Fuente: [1]

Rag + LLM

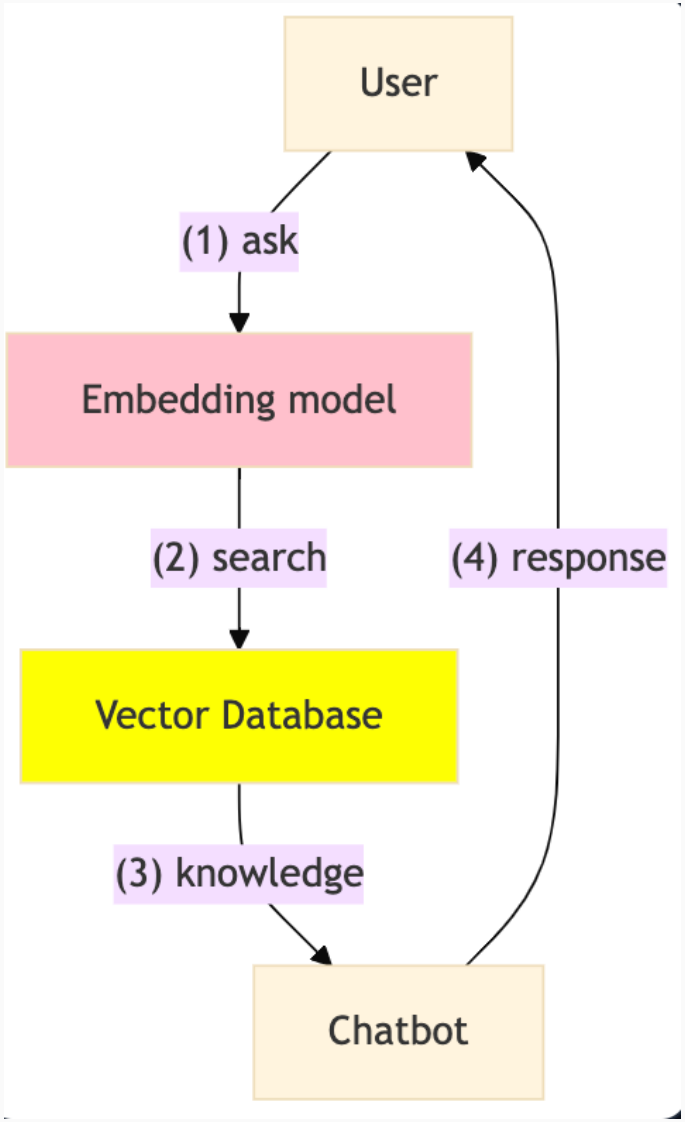

Figure 4: Ilustración Rag Simple. Fuente: [1]

Rag as Tool

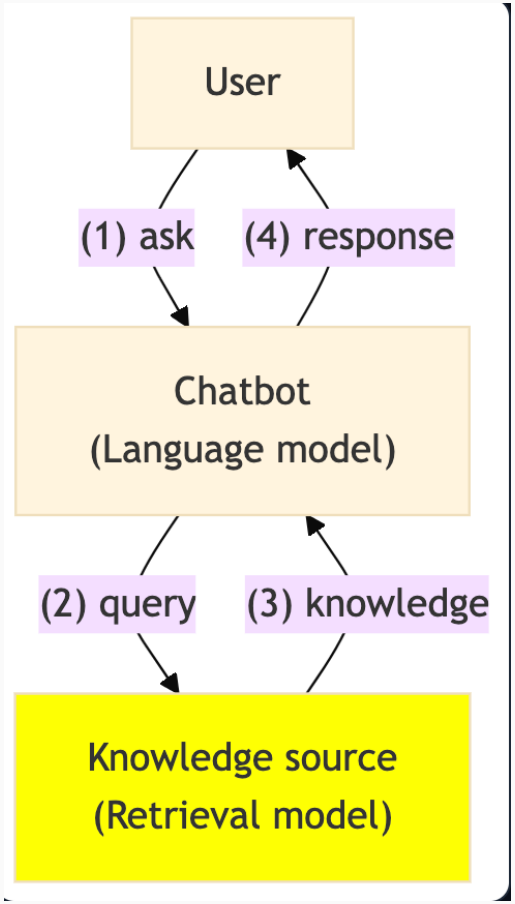

Figure 5: Ilustración Rag as Tool. Fuente: [1]

El Paper Original: Lewis et al. 2020

RAG (Retrieval-Augmented Generation) combina modelos de lenguaje con recuperación de información:

Formulación matemática:

$$p(y \mid x) = \sum_{z \in \text{top-}k(x)} p(z \mid x) \cdot p(y \mid x, z)$$

donde:

- x: input query
- y: output generado
- z: documento recuperado de la base de conocimiento
- $p(z \mid x)$: probabilidad de relevancia del documento (retriever)
- $p(y \mid x, z)$: probabilidad de generar y dado x y z (generator)

Los tres componentes fundamentales

Retrieve $\rightarrow$ **Augment** $\rightarrow$ **Generate**

RAG Clásico: Arquitectura Original

Componentes del paper original (Lewis et al., 2020):

- **Retriever**: DPR (Dense Passage Retriever)
  - Bi-encoder basado en BERT
  - Codifica queries y documentos independientemente
  - Similitud mediante producto punto en espacio denso
- **Generator**: BART (seq2seq transformer)
  - Recibe query + documentos recuperados concatenados
  - Genera respuesta mediante decoder autoregresivo
- **Knowledge Source**: Wikipedia completa
  - 21 millones de passages de 100 palabras
  - Pre-computados y almacenados en índice FAISS

Entrenamiento end-to-end

Los parámetros del retriever y generator se entrenan conjuntamente, propagando gradientes a través de documentos recuperados (usando marginalización).

Rag + LLM

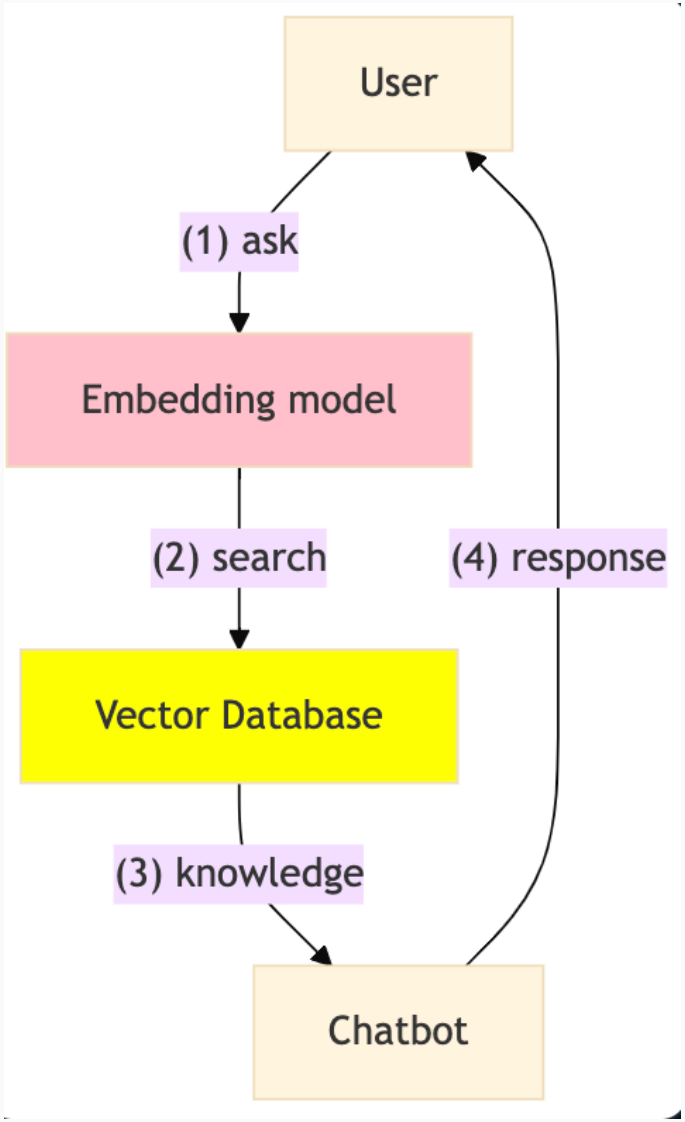

Figure 6: Ilustración Rag Simple. Fuente: [1]

Ejemplo de Flujo RAG Clásico: El Señor de los Anillos

Escenario: Sistema RAG con base de conocimiento sobre LOTR

1. **RETRIEVE** - DPR encuentra passages relevantes:

Query del usuario

"¿Quién destruyó el Anillo Único y dónde?"

Top-3 passages recuperados por DPR:

- **P1** (score: 0.89): "Frodo Bolsón llegó al Monte del Destino en Mordor, donde finalmente el Anillo fue destruido..."
- **P2** (score: 0.76): "El Anillo Único fue forjado por Sauron y solo podía ser destruido en los fuegos del Monte del Destino..."
- **P3** (score: 0.68): "Sam acompañó a Frodo hasta las Grietas del Destino, donde culminó la misión..."

Ejemplo de Flujo RAG Clásico (continuación)

2. **AUGMENT** - Construcción del prompt aumentado:

Prompt enviado a BART Generator

```
Contexto:

[P1] Frodo Bolsón llegó al Monte del Destino en Mordor...

[P2] El Anillo Único fue forjado por Sauron y solo podía ser destruido...

[P3] Sam acompañó a Frodo hasta las Grietas del Destino...

Pregunta: ¿Quién destruyó el Anillo Único y dónde?

Respuesta:
```

3. **GENERATE** - BART genera respuesta basada en contexto:

Respuesta generada

"Frodo Bolsón destruyó el Anillo Único en el Monte del Destino, en Mordor, específicamente en las Grietas del Destino, con el apoyo de Sam."

DPR: Dense Passage Retriever

DPR (Karpukhin et al., 2020) revolucionó la recuperación de información usando embeddings densos en lugar de métodos léxicos como BM25.

Idea central: Bi-encoder

Dos encoders BERT separados convierten queries y documentos en vectores densos. La similitud se calcula mediante producto punto en el espacio de embeddings.

Ventajas sobre BM25:

- Captura similitud semántica ("automobile" $\approx$ "car")
- No requiere coincidencia léxica exacta
- Aprende representaciones end-to-end con datos de relevancia

Trade-off: Mayor precisión semántica, pero requiere pre-computar embeddings de todos los documentos.

FAISS: Búsqueda Eficiente en Alta Dimensión

FAISS (Facebook AI Similarity Search) permite búsqueda rápida en millones de vectores de alta dimensión.

El problema

Con millones de documentos embeddeados, buscar exhaustivamente el más similar a una query es muy costoso ($O(Nd)$). FAISS optimiza esta búsqueda.

Estrategias de indexación:

- **Flat Index**: Búsqueda exacta pero lenta (baseline)
- **IVF**: Clustering para buscar solo en regiones prometedoras
- **HNSW**: Grafo navegable multi-nivel (muy usado en producción)
- **Product Quantization**: Compresión de vectores para ahorrar memoria

Trade-off

Precisión vs. velocidad vs. memoria. HNSW ofrece el mejor balance para producción.

Limitaciones del RAG Clásico

A pesar de su éxito, el RAG original presenta limitaciones importantes:

1. Single-stage retrieval insuficiente

Queries complejas requieren múltiples pasos de razonamiento. Un único retrieval puede no capturar toda la información necesaria.

2. Embeddings genéricos subóptimos

Modelos entrenados en corpus general fallan en dominios específicos. Vocabulario especializado (médico, legal, técnico) pobremente representado.

3. Ausencia de re-ranking

Bi-encoders optimizan para velocidad, no precisión máxima. No hay filtrado post-retrieval de resultados irrelevantes.

4. Context window limitado

LLMs clásicos (BART, T5) tenían límites de 1024-2048 tokens, restringiendo la cantidad de documentos recuperables.

Outline : RAG Moderno: Arquitecturas Multi-Stage

El Pipeline RAG Moderno

La evolución de RAG ha resultado en arquitecturas multi-stage sofisticadas:

Query $\rightarrow$ **Embedding** $\rightarrow$ **Retrieval (top-100)** $\rightarrow$
Reranking (top-5) $\rightarrow$ **Augmented Prompt** $\rightarrow$ **LLM**

Ventajas del enfoque multi-stage:

- **Recall alto en primera etapa**: Recuperar ampliamente (100-200 documentos)
- **Precision alta en segunda etapa**: Re-ranking preciso de candidatos
- **Eficiencia computacional**: Bi-encoder rápido + cross-encoder preciso
- **Flexibilidad**: Combinar múltiples estrategias de retrieval

Principio de diseño

Separar concerns: velocidad en retrieval, precisión en re-ranking.

Rag Moderno

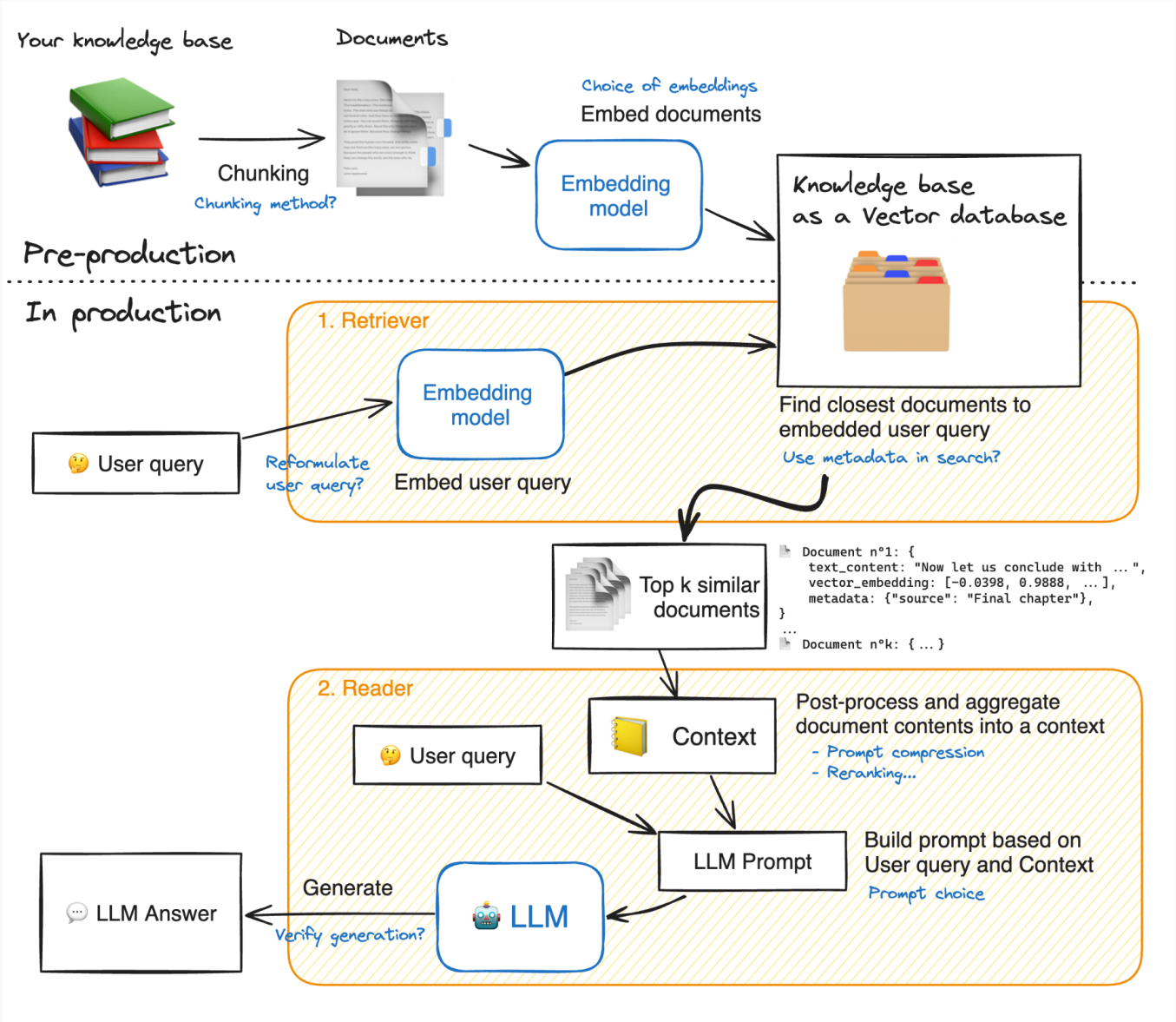

Figure 7: Ilustración Modern RAG. Fuente: [3]

Stage 1: Embeddings Densos Modernos

Los embeddings modernos han evolucionado significativamente desde DPR (2020).

La nueva generación (2023-2024)

Modelos entrenados con cientos de millones de pares query-documento, optimizados específicamente para retrieval.

Modelos estado del arte:

- **BGE (BAAI)**: Excelente rendimiento multilingüe, 100M+ pares de entrenamiento
- **E5 (Microsoft)**: Líder en benchmarks inglés (BEIR, MTEB)
- **Instructor**: Embeddings adaptables mediante instrucciones textuales
- **OpenAI text-embedding-3**: Alta calidad, API comercial

Mejora clave

Los embeddings modernos superan consistentemente a DPR en 10-20% en métricas de retrieval, especialmente en queries complejas.

Stage 2: BM25 - El Clásico que Aún Funciona

BM25 sigue siendo relevante en sistemas RAG modernos debido a sus fortalezas únicas:

Ventajas de BM25 en 2024:

- **Exactitud léxica**: Excelente para nombres propios, IDs, códigos
- **Términos técnicos**: Captura acrónimos y terminología especializada
- **Sin modelos neuronales**: No requiere GPUs ni embeddings pre-computados
- **Interpretabilidad**: Scores explícitos basados en frecuencias

Caso de uso típico

Query: "K-means clustering algorithm"
BM25 captura "K-means" exactamente, mientras embeddings pueden confundir con "K-nearest neighbors".

Hybrid Retrieval: Lo Mejor de Dos Mundos

Combinar retrieval semántico (embeddings) con léxico (BM25) supera a cada método individual.

¿Por qué combinar ambos?

BM25 y embeddings tienen fortalezas complementarias: BM25 captura matches exactos, embeddings capturan similitud semántica.

Estrategias de fusión:

- **Reciprocal Rank Fusion (RRF)**: Combina rankings sin necesidad de normalizar scores

$$\text{score}_{\text{RRF}}(d) = \sum_{r \in \{r_{\text{dense}}, r_{\text{sparse}}\}} \frac{1}{k + r(d)}$$

- **Weighted Combination**: Promedio ponderado de scores normalizados

$$\text{score}(d) = \alpha \cdot \text{dense}(d) + (1 - \alpha) \cdot \text{sparse}(d)$$

- **Cascade**: BM25 primero (rápido, top-1000) $\rightarrow$ Dense después (preciso, top-100)

Hybrid Retrieval: Ejemplo Práctico

Escenario: Búsqueda en biblioteca de recetas chilenas

Query del usuario

"Receta con zapallo para el otoño"

Resultados por método:

- **Solo BM25 (léxico)**: Busca palabra exacta "zapallo"
  - Top-1: Pastel de zapallo
  - Top-2: Sopa de zapallo
  - Top-3: Ensalada de tomate (menciona "otoño" pero no zapallo)
- **Solo Embeddings (semántico)**: Captura idea de "comida de temporada"
  - Top-1: Cazuela de vacuno (plato de otoño)
  - Top-2: Pastel de zapallo
  - Top-3: Humitas (plato de otoño con choclo)
- **Hybrid (RRF)**: Combina exactitud + contexto
  - Top-1: Pastel de zapallo (tiene palabra exacta + es plato de otoño)
  - Top-2: Sopa de zapallo (ídem)
  - Top-3: Cazuela (plato típico de otoño)

Evidencia Empírica: Hybrid > Individual

Resultados en benchmark BEIR (Thakur et al., 2021):

Dataset	BM25	Dense	Hybrid
MS MARCO	0.228	0.330	**0.387**
Natural Questions	0.329	0.525	**0.564**
HotpotQA	0.603	0.603	**0.676**
FiQA	0.236	0.329	**0.359**

Conclusión

Hybrid retrieval consistentemente supera métodos individuales. Los beneficios son complementarios: BM25 captura exactitud léxica, embeddings capturan similitud semántica.

Stage 3: Reranking

¿Por qué necesitamos reranking?

- **Limitación de bi-encoders**:
  - Codifican query y documento independientemente
  - No hay interacción directa entre query y documento
  - Optimizados para velocidad, no precisión máxima
- **Ventaja de cross-encoders**:
  - Procesan query y documento juntos
  - Cross-attention completa entre todos los tokens
  - Mucho más preciso pero también mucho más lento

Pipeline óptimo:

Bi-encoder recupera top-100 (rápido) $\rightarrow$ Cross-encoder reordena a top-5 (preciso)

Costo computacional

Cross-encoder es 100-1000x más lento que bi-encoder. Solo es viable en un conjunto pequeño de candidatos.

Cross-Encoder: Arquitectura

Diferencias arquitecturales fundamentales:

Bi-encoder (retrieval):

- Input: `[CLS] query [SEP]` $\rightarrow E_Q(q) \in \mathbb{R}^d$
- Input: `[CLS] doc [SEP]` $\rightarrow E_D(d) \in \mathbb{R}^d$
- Score: $E_Q(q)^T E_D(d)$ (producto punto)

Cross-encoder (reranking):

- Input: `[CLS] query [SEP] doc [SEP]` $\rightarrow$ BERT $\rightarrow h_{[\text{CLS}]}$
- Score: $\text{MLP}(h_{[\text{CLS}]}) \in [0, 1]$ (probabilidad de relevancia)
- Full cross-attention entre query y documento

Ejemplo de mejora

Query: "tratamiento diabetes tipo 2"

Bi-encoder puede confundir tipo 1 y tipo 2.

Cross-encoder analiza la interacción exacta y distingue correctamente.

Pipeline RAG Moderno Completo: Ejemplo LOTR

Query: "¿Qué personajes llevaron el Anillo además de Frodo?"

Stage 1 - Hybrid Retrieval (Recall):

- **BM25**: 50 docs con "Anillo"
- **Embeddings**: 50 docs semánticamente similares
- **RRF Fusion**: Combina a top-100 únicos

Stage 2 - Reranking (Precision):

- Cross-encoder evalúa cada (query, doc) del top-100
- Reordena por relevancia exacta
- Selecciona top-5 finales

Pipeline RAG Moderno Completo: Ejemplo LOTR (cont.)

Top-5 después de reranking:

1. "Bilbo encontró el Anillo en las Montañas Nubladas..." (0.94)
2. "Isildur cortó el Anillo del dedo de Sauron..." (0.89)
3. "Sméagol mató a Déagol para obtener el Anillo..." (0.87)
4. "Sam llevó brevemente el Anillo cuando rescató a Frodo..." (0.82)
5. "Gandalf se negó a tocar el Anillo..." (0.78)

Stage 3 - Augmented Generation:

Prompt al LLM

`Contexto:` [Top-5 passages del reranking]

`Pregunta:` ¿Qué personajes llevaron el Anillo además de Frodo?

`Instrucción:` Responde basándote SOLO en el contexto proporcionado.

Respuesta Generada por el LLM

Respuesta

Según el contexto, además de Frodo, los siguientes personajes llevaron el Anillo Único:

- **Bilbo**: Lo encontró en las Montañas Nubladas
- **Isildur**: Lo cortó del dedo de Sauron tras su derrota
- **Sméagol/Gollum**: Lo obtuvo matando a Déagol
- **Sam**: Lo portó brevemente al rescatar a Frodo en Cirith Ungol

Outline : Optimización de Retrieval: Más Allá del Modelo Base

La Importancia Crítica del Retrieval

Principio fundamental de RAG

Sin buen retrieval, no hay buen RAG. La calidad del LLM es secundaria si no recuperamos los documentos correctos.

La cadena de calidad en RAG:

Retrieval (Recall) $\rightarrow$ **Reranking** (Precision) $\rightarrow$ **Generation** (Quality)

Ejemplo del problema

Si tu sistema recupera documentos sobre "diabetes tipo 1" cuando el usuario pregunta por "diabetes tipo 2", incluso GPT-4 generará una respuesta incorrecta porque el contexto está mal.

Por eso necesitamos:

- **Domain Adaptation**: Embeddings especializados para dominios específicos
- **Chunking inteligente**: Organizar documentos para retrieval óptimo
- **Query transformation**: Mejorar las queries antes de buscar

Outline : Domain Adaptation de Embeddings

El Problema: Out-of-Domain Degradation

Los embeddings genéricos (entrenados en Wikipedia/web general) fallan en dominios específicos.

Ejemplo: Documentos Legales

En textos legales chilenos, palabras como "conforme", "según", "establece", "dispone" aparecen constantemente pero no aportan significado específico. Un embedding genérico les dará peso, mientras que términos clave como "nulidad", "cosa juzgada" no serán bien representados.

Ejemplo: Documentos Médicos

Query: "inhibidores ECA para hipertensión"

Embedding genérico confunde:

- "ECA" (Enzima Convertidora de Angiotensina)
- "IECA" (Inhibidor de ECA)
- "ARA-II" (alternativa terapéutica)

Embedding especializado entiende la jerarquía y relaciones.

¿Cuándo hacer Domain Adaptation?

Señales claras de que necesitas domain adaptation:

1. Vocabulario altamente especializado

- Médico: "ECA", "IECA", "ARA-II", "betabloqueantes"
- Legal: "nulidad", "cosa juzgada", "recurso de casación"
- Financiero: "swap", "derivado", "cobertura delta"
- Técnico: Siglas, acrónimos, términos específicos

2. Performance pobre con embeddings genéricos

- nDCG@10 ¡ 0.6 en evaluación manual
- Usuarios reportan resultados irrelevantes frecuentemente
- Modelo general en corpus legal: nDCG baja de 0.82 a 0.53

3. Corpus suficiente para fine-tuning

Mínimo: 10,000 documentos. Ideal: 100,000+ documentos del dominio target.

Estrategia: Generación Sintética con LLMs

1. **Tomar documentos del dominio target** (legal, médico, técnico)
2. **Usar LLM** (GPT-4, Claude) para generar queries sintéticas
3. **Crear pares (query, documento)** para fine-tuning
4. **Fine-tunear embedding model** con contrastive learning

Prompt de generación

```
Dado el siguiente fragmento de documento legal chileno:

[DOCUMENTO]

Genera 5 preguntas específicas que este fragmento
podría responder. Formatea como JSON con campos:
question, difficulty (easy/medium/hard), keywords.
```

Ventajas: Escalable (100k+ pares), diversidad de queries, bajo costo ($50-200).

Outline : Chunking: Organizando Documentos para Retrieval

El Problema del Chunking

¿Por qué necesitamos chunking?

- Documentos largos (libros, papers, manuales) no caben en un embedding
- Embeddings de documentos completos son "promedio difuso" de todo el contenido
- LLMs tienen límites de context window (necesitamos chunks manejables)

Problema del chunking naive

Dividir cada 512 tokens sin considerar estructura rompe el contexto:

- Corta oraciones a la mitad
- Separa información relacionada
- Pierde contexto de sección/capítulo

¿Qué buscamos con el chunking?

- Chunks semánticamente coherentes
- Tamaño adecuado (ni muy grande, ni muy pequeño)
- Mantener contexto importante

Estrategias de Chunking: Overview

Tenemos 4 estrategias principales para dividir documentos inteligentemente:

1. **Chunking por Estructura**: Respetar la organización natural del documento (capítulos, secciones)
2. **Overlapping Chunks**: Mantener solapamiento entre chunks para no perder contexto en los límites
3. **Semantic Chunking**: Dividir cuando cambia el tema, no por tamaño fijo
4. **Hierarchical Chunking**: Mantener el documento a múltiples niveles de granularidad

Objetivo común

Crear chunks que mantengan contexto coherente y faciliten encontrar información relevante.

Veamos cada estrategia con ejemplos concretos...

Estrategia 1: Chunking por Estructura

Aprovechar la estructura natural que el autor ya creó.

Ejemplo: Libro "Gran Libro de Cocina Chilena" (500 páginas)

Chunking naive (cada 500 palabras):

- Chunk 1: Mitad de "Pastel de Choclo" + inicio de "Empanadas"
- Chunk 2: Fin de "Empanadas" + mitad de "Cazuela"

Chunking por estructura (por receta completa):

- Chunk 1: Receta completa de "Pastel de Choclo" (ingredientes, preparación, tips)
- Chunk 2: Receta completa de "Empanadas de Pino"
- Chunk 3: Receta completa de "Cazuela"

Query: "¿Cómo hacer empanadas?" $\rightarrow$ Recupera chunk completo y coherente.

Ventaja: Cada chunk es una unidad semántica completa.

Estrategia 2: Overlapping Chunks

Mantener solapamiento para no perder información en los límites Chunk 500 tokens, overlap 100 tokens (20%).

Ejemplo: Manual de Usuario de Smartphone

Texto original:

- Párrafo A: "Cómo tomar fotos con modo nocturno..."
- Párrafo B: "El modo nocturno funciona mejor en ambientes oscuros. Para activarlo..."
- Párrafo C: "Además del modo nocturno, puedes usar el modo retrato..."

Sin overlap: [A] — [B] — [C]

- Si corto entre A y B, pierdo contexto de cómo A y B se relacionan

Con overlap de 20%: [A + inicio B] — [fin A + B + inicio C] — [fin B + C]

- Cada chunk tiene contexto del anterior y siguiente
- Query "activar modo nocturno" encuentra el chunk del medio completo

Estrategia 3: Semantic Chunking

Dividir cuando cambia el tema, no por tamaño arbitrario.

Ejemplo: Guía Turística de Chile

Contenido del documento:

- Párrafos 1-3: Hablan de Santiago (Plaza de Armas, Cerro Santa Lucía, Barrio Lastarria)
- Párrafos 4-5: **Cambio de tema** $\longrightarrow$ Hablan de Valparaíso (ascensores, cerros, puerto)
- Párrafos 6-8: Hablan de Atacama (Valle de la Luna, geysers, lagunas altiplánicas)

Semantic chunking detecta cambios:

- Chunk 1: Todo sobre Santiago (párrafos 1-3)
- Chunk 2: Todo sobre Valparaíso (párrafos 4-5)
- Chunk 3: Todo sobre Atacama (párrafos 6-8)

Query: "¿Qué hacer en Valparaíso?" $\longrightarrow$ Recupera solo el chunk relevante, sin mezclar Santiago o Atacama.

Ventaja: Chunks coherentes por tema, no por tamaño.

Estrategia 4: Hierarchical Chunking

Mantener el documento a múltiples niveles de detalle.

Ejemplo: Artículo de Noticias

Artículo: "Nueva Ley de Pensiones en Chile" (2000 palabras) **Nivel 1 - Resumen (200 palabras)**:

- "El Congreso aprobó reforma de pensiones que aumenta cotización y crea nuevo pilar solidario..."

Nivel 2 - Secciones (500 palabras c/u):

- Chunk A: "Aumento de cotización del 10% al 11.5%..."
- Chunk B: "Nuevo pilar solidario para pensiones bajas..."
- Chunk C: "Cronograma de implementación gradual..."

Nivel 3 - Párrafos detallados (100 palabras c/u):

- "El aumento del 1.5% se distribuirá en cuentas individuales..."

Query: "¿Cuánto subirá la cotización?" $\rightarrow$ Busca nivel 2, luego 3 si necesita

Metadata Enrichment: Mejorando los Chunks

Agregar información contextual a cada chunk mejora significativamente el retrieval.

Ejemplo: Biblioteca Digital de Universidad

Chunk de texto: "La fotosíntesis es el proceso mediante el cual las plantas..."

Metadata agregada:

```
{
  "titulo": "Biología General - Campbell 11va Ed.",
  "capitulo": "Capítulo 10: Fotosíntesis",
  "pagina": 185,
  "tema": "Procesos celulares",
  "nivel": "Pregrado",
  "fecha": "2021",
  "idioma": "español"
}
```

Mejorando chuncking

Uso práctico:

- Query: "fotosíntesis pregrado" $\rightarrow$ Filtra por nivel="Pregrado" antes de buscar
- Query: "capítulo 10 Campbell" $\rightarrow$ Filtra por metadata específica

Outline : Embeddings: Aspectos Técnicos

¿Cómo se entrenan los embeddings modernos?

Los embeddings modernos se entrenan mediante **contrastive learning**.

Objetivo

Aprender representaciones donde queries y documentos relevantes estén cerca en el espacio de embeddings, mientras que queries y documentos irrelevantes estén lejos.

InfoNCE Loss:

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(q, d^+)/\tau)}{\sum_{d_i \in \{d^+\} \cup D^-} \exp(\text{sim}(q, d_i)/\tau)}$$

donde q es query, d^+ es documento relevante, D^- son documentos irrelevantes, y τ es temperatura.

Hard Negative Mining: Negativos difíciles (semánticamente similares pero no relevantes) fuerzan al modelo a aprender distinciones finas.

Problemas Comunes de los Embeddings

1. Out-of-Domain Degradation

Embeddings entrenados en Wikipedia/News fallan en dominios específicos (legal, médico). Solución: Domain adaptation.

2. Semantic Mismatch

Query corta (5-10 palabras) vs. Documento largo (500 palabras). Embeddings optimizados para textos similares en longitud. Solución: Query expansion.

3. Lexical Gap

Embeddings puros fallan en términos técnicos exactos, IDs, códigos (e.g., "K-means" vs "K-nearest"). Solución: Hybrid retrieval con BM25.

4. Multilingüismo

Modelos multilingües sacrifican performance por idioma. Cross-lingual retrieval es más desafiante. Solución: Modelos específicos por idioma cuando sea posible.

Outline : Técnicas Avanzadas de RAG

Query Transformation

Problema: La query original del usuario puede no ser óptima para retrieval.
Técnica 1: Query Expansion Generar múltiples variantes de la query y recuperar con todas:

Ejemplo

- Original: "tratamiento diabetes"
- Expandida: ["tratamiento diabetes tipo 2", "terapia diabetes mellitus", "manejo hiperglucemia"]

Recuperar con las 3 y combinar resultados mediante fusion.

Técnica 2: HyDE (Hypothetical Document Embeddings)

1. Generar respuesta hipotética con LLM (sin documentos)
2. Embedear respuesta hipotética (no la query original)
3. Buscar documentos similares a la respuesta hipotética

Intuición: Documentos relevantes se parecen más a respuestas que a preguntas.

Step-back Prompting

Para queries complejas, primero hacer una pregunta más general para obtener contexto.

Ejemplo: Reforma Tributaria Chile

Query específica: "¿Cuál fue el impacto económico de la Ley de Reforma Tributaria de 2022 en Chile?"

Step-back query: "¿Qué son las reformas tributarias y cómo afectan la economía?"

Proceso:

1. Recuperar documentos con step-back query (contexto general)
2. Recuperar documentos con query original (detalles específicos)
3. Combinar ambos conjuntos
4. LLM genera respuesta usando contexto general + específico

Beneficio: Proporciona fundamentos conceptuales antes de detalles específicos.

Contextual Compression

Problema: Chunks recuperados contienen información irrelevante que desperdicia context window del LLM.

Solución: Usar un LLM pequeño para extraer solo información relevante antes de pasar al LLM principal.

Ejemplo

Chunk original (500 tokens): "PostgreSQL es un sistema de base de datos... [párrafo largo sobre historia]... MVCC (Multi-Version Concurrency Control) permite que múltiples transacciones... [más contenido]..."

Query: "¿Cómo funciona MVCC en PostgreSQL?"

Chunk comprimido (150 tokens): "MVCC (Multi-Version Concurrency Control) permite que múltiples transacciones accedan concurrentemente..."

Ventajas: Reducción 40-60% en tokens, mejora precisión del LLM, permite incluir más chunks.

Self-RAG: Retrieval Adaptativo

Self-RAG entrena el LLM para decidir autónomamente cuándo necesita información externa. **Tokens especiales de control:**

- `<retrieve>`: Modelo solicita recuperación de documentos
- `<no_retrieve>`: Modelo confía en su conocimiento interno
- `<relevant>` / `<irrelevant>`: Evalúa calidad de docs recuperados

Ejemplo de flujo

User: "¿Cuál es la capital de Francia?"

LLM: `<no_retrieve>` "La capital de Francia es París" (conocimiento interno)

User: "¿Cuál fue el precio de cierre de Bitcoin ayer?"

LLM: `<retrieve>` [busca info actualizada] "Según datos de ayer, Bitcoin cerró en $..."

Ventaja: Reduce latencia y costo al no recuperar cuando no es necesario.

Outline : RAG Multimodal

Extendiendo RAG a Imágenes y Texto

Motivación: Muchos dominios requieren integrar texto, imágenes, tablas, gráficos.

Embeddings multimodales clave:

- **CLIP (OpenAI)**: Joint embedding de imágenes y texto en mismo espacio
- **BLIP-2 (Salesforce)**: Embeddings para queries complejas multimodales
- **SigLIP (Google)**: CLIP mejorado

Tipos de retrieval multimodal:

1. **Text** $\rightarrow$ **Image**: Query textual recupera imágenes relevantes
2. **Image** $\rightarrow$ **Text**: Query imagen recupera documentos relevantes
3. **Image** $\rightarrow$ **Image**: Query imagen recupera imágenes similares

Propiedad clave de CLIP

Imágenes y texto se alinean en el mismo espacio de embeddings, permitiendo similitud directa.

RAG Multimodal: Casos de Uso

Arquitectura completa:

Query (texto/imagen) $\rightarrow$ **Embedding multimodal** $\rightarrow$
Retrieve (imágenes + texto) $\rightarrow$ **Multimodal LLM** $\rightarrow$ **Respuesta**

Ejemplos de aplicación:

- **Diagnóstico médico**: "Buscar casos similares a esta radiografía" $\rightarrow$ Recupera imágenes + historiales clínicos de casos parecidos
- **E-commerce**: "Buscar zapatos similares a esta imagen pero en color azul" $\rightarrow$ Recupera productos visuales similares
- **Educación**: "Explica fotosíntesis" $\rightarrow$ Recupera diagramas + explicaciones textuales
- **Soporte técnico**: "Mi pantalla muestra este error [imagen]" $\rightarrow$ Recupera manuales con diagramas relevantes

Outline : Evaluación de RAG

Métricas de Evaluación

Evaluar RAG requiere medir tanto retrieval como generation:

Métricas de Retrieval:

- **Recall@K**: ¿Están los docs relevantes en top-K?
- **Precision@K**: ¿Qué fracción de top-K es relevante?
- **nDCG@K**: Penaliza docs relevantes en posiciones bajas
- **MRR**: Posición del primer doc relevante

Métricas de Generation:

- **Faithfulness**: ¿La respuesta es fiel a docs recuperados?
- **Answer Relevance**: ¿La respuesta aborda la query?
- **Context Precision**: ¿Se usó la información relevante?
- **Context Recall**: ¿Se recuperó todo el contexto necesario?

Evaluación manual es crítica

Métricas automáticas son aproximaciones. Evaluación humana sigue siendo gold standard.

Debugging RAG: Checklist Práctico

Cuando tu sistema RAG no funciona bien, diagnosticar sistemáticamente:

1. Problema de Retrieval

- ¿Los chunks tienen tamaño adecuado? (ni muy grandes ni muy pequeños)
- ¿Recuperas suficientes documentos? (prueba $k = 50$ en lugar de $k = 5$)
- ¿El embedding model está en el idioma correcto?
- ¿Hybrid retrieval (semántico + BM25) mejora resultados?

2. Problema de Generation

- ¿El prompt instruye claramente cómo usar el contexto?
- ¿El LLM alucina a pesar del contexto? (instrucciones más estrictas)
- ¿El contexto es demasiado largo/ruidoso? (probar compression)

3. Problema de Reranking

- ¿El reranker mejora o empeora? (a veces overfitting)
- ¿Cross-encoder fue entrenado en dominio similar?

RAG vs. Fine-tuning: ¿Cuándo usar cada uno?

Criterio	RAG	Fine-tuning	Hybrid
Conocimiento actualizado			
Datos privados			
Costo inicial	Bajo	Alto	Medio-Alto
Mantenimiento	Fácil	Difícil	Medio
Latencia	Media	Baja	Alta
Precisión dominio	Media	Alta	Alta
Interpretabilidad	Alta	Baja	Alta

Recomendación general

Empezar con RAG, luego considerar fine-tuning solo si necesitas:

- Extrema especialización de dominio
- Latencia crítica (¡100ms)
- Cambio fundamental de comportamiento del modelo

Outline : Herramientas del Ecosistema

Frameworks y Vector Databases

Frameworks RAG principales:

- **LangChain**: Framework completo, alta abstracción, gran comunidad
- **LlamaIndex**: Especializado en RAG, excelente para prototyping
- **Haystack**: Pipeline-based, muy flexible, production-ready

Vector Databases:

- **FAISS**: In-memory, muy rápido, no persistente (prototyping)
- **Pinecone**: Managed service, escalable, comercial ($$)
- **Weaviate**: Open-source, production-ready, multimodal
- **Qdrant**: Rust-based, muy rápido, filtering avanzado
- **Chroma**: Python-native, simple, bueno para empezar

Embedding Models:

- **Sentence-Transformers**: Librería estándar (HuggingFace)
- **OpenAI Embeddings**: API comercial, alta calidad
- **Cohere Embed**: Multilingüe, optimizado para retrieval

Conclusiones: El Estado de RAG en 2024-2025

Lo que aprendimos:

1. **RAG es fundamental**: Sin buen retrieval, no hay buen sistema
2. **Hybrid es superior**: BM25 + Embeddings supera métodos individuales
3. **Multi-stage es clave**: Retrieval (recall) $\rightarrow$ Reranking (precision) $\rightarrow$ Generation
4. **Domain adaptation importa**: Embeddings genéricos fallan en dominios específicos
5. **Chunking inteligente**: Organización de documentos impacta directamente calidad

El futuro de RAG

- RAG multimodal (texto + imagen + audio)
- Self-RAG y retrieval adaptativo
- Integración más profunda con LLMs (retrieval durante generación)
- Mejor evaluación automática de calidad

Questions?

References i

X.-S. Nguyen.
Code a simple rag from scratch.
https://huggingface.co/blog/ngxson/make-your-own-rag, oct 2024.
Blog post, Hugging Face.

J. Parmar, S. Satheesh, M. Patwary, M. Shoeybi, and B. Catanzaro.
Reuse, don't retrain: A recipe for continued pretraining of language models, 2024.

A. Roucher.
Advanced rag on hugging face documentation using langchain.
https://huggingface.co/learn/cookbook/advanced_rag, oct 2024.
Blog post, Hugging Face.