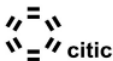


MICROCREDENCIAL ALGORITMOS VERDES PARA LA IA DISEÑO E IMPLEMENTACIÓN

Tema: Natural language processing

Sesión 3: Retrieval

Ponente: Olga Zamaraeva



UNIVERSIDADE DA CORUÑA



Financiado por la
Unión Europea
NextGenerationEU



GOBIERNO
DE ESPAÑA

MINISTERIO
PARA LA TRANSFORMACIÓN DIGITAL
Y DE LA FUNCIÓN PÚBLICA

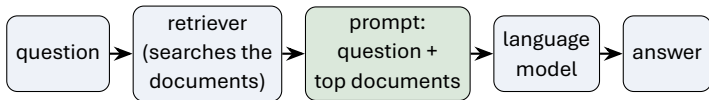


2026

CONTENT WARNING

- The translation example today uses sentences from the novel *Speechwriter*
- Two of these sentences contain strong swearing, as in the original
- We keep them: translating swearing is a real problem for translation systems

RETRIEVE, THEN READ: RAG

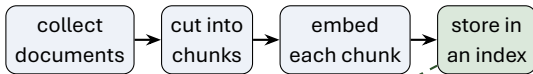


- **RAG** = retrieval-augmented generation
- Step 1: find the relevant documents. Step 2: put them into the prompt, next to the question
- The model answers from the prompt *and* from what it learned in training
- So a small model can answer about facts it never saw in training

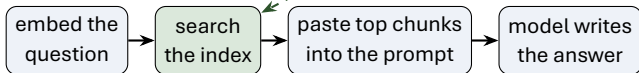
Source: Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," NeurIPS 2020 (arXiv:2005.11401)

WHERE DO THE DOCUMENTS COME FROM?

1. Before any question (once)



2. For every question

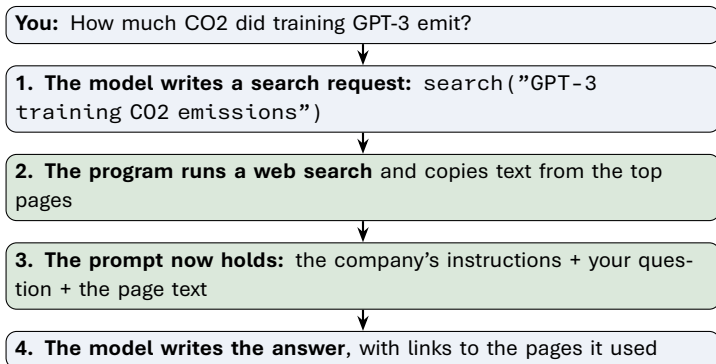


- The **system builders** choose the documents: help-desk articles, contracts, regulations, ...
- The model does not choose where to search. It only sees the final prompt.

RETRIEVAL IN REAL SYSTEMS

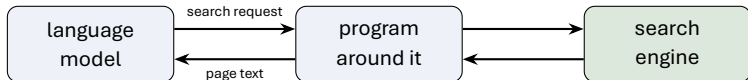
- **Scale:** millions of chunks, so the search is approximate: it checks only the most promising chunks
- **Hybrid search:** word-based and embedding-based search run together, and their results are merged
- **Reranking:** a second, more careful model reorders the top 20–50 results; the best few go into the prompt
- **Web search:** the retriever is a search engine, and the collection is the web
- **The model asks for a search:** it writes a search request; the surrounding program runs it and returns the results. The builders still decide what can be searched.

EXAMPLE: ASKING CHATGPT A QUESTION



Gemini, Claude, Copilot and Perplexity work the same way.

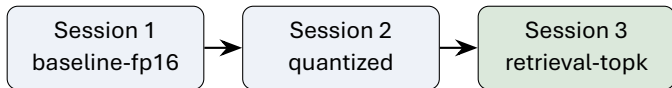
HOW DOES THE MODEL “DECIDE” TO SEARCH?



- The model only writes text. A search request is text in a special format.
- The model was trained on examples where questions about specific or recent facts are followed by a search request
- Uploaded files work the same way: a long file is cut into chunks, embedded and searched, as in “Where do the documents come from?”

WHERE WE LEFT OFF

- Session 1: `baseline-fp16`: the model reads all the documents for every question
- Session 2: `quantized`: the same documents, a smaller model
- Today: **retrieval**: the same model, only the relevant documents



WHAT THE BASELINE DID

Question: “What does the acronym PUE stand for?”



the answer
is here

- For every question, the model read all 8 documents: about 1,350 tokens
- The answer to each question is in one document
- So most of what the model read was irrelevant to the question

WHAT SESSIONS 1 AND 2 MEASURED: READING COMPREHENSION

- **Reading comprehension:** the model gets a text that contains the answer, and must find it
- **Question answering:** the model gets only the question, and must know or find the answer
- In Sessions 1 and 2, every prompt contained all 8 documents, with every answer. That was reading comprehension.
- Today we redo it as real question answering. With a small model, the results are much worse.

REAL QUESTION ANSWERING, NO DOCUMENTS: CLOSED BOOK

- Same 8 questions, question only, no documents (“closed book”)
- In our test run: **2 of 8** correct, against 8 of 8 with the documents
- The wrong answers sounded just as sure as the right ones:

Question	Model, no documents	Document says
GPT-3 training emissions	“approx. 1,168 tonnes”	552 tonnes
Data centres’ share of world electricity, 2024	“about 1%”	1.5%
Name of the technique (RAG)	“Retrieval-Augmented Learning”	Retrieval-Augmented Generation

HOW THE RETRIEVER FINDS DOCUMENTS

- Turn the question into a vector (an embedding). Do the same for every document.
- Compare the question vector with each document vector, using **cosine similarity**
- Keep the k most similar documents. Only those go into the prompt.

A WORKED EXAMPLE

- Question: “How many tonnes of CO2 did training GPT-3 emit?”
- Toy 2-D vectors: question = (2, 1)
- Documents: patterson_doc = (4, 1), iea_doc = (1, 4), rag_doc = (0, 3)

$$\cos(q, \text{patterson}) \approx 0.98 \quad \cos(q, \text{iea}) \approx 0.65 \quad \cos(q, \text{rag}) \approx 0.45$$

- With $k = 1$, the model sees only patterson_doc, the document about training emissions

THE OLDER METHOD: SPARSE RETRIEVAL

- **TF-IDF**: rank documents by the words they share with the question
- Shared words count more when they are rare in the collection
- No embedding model, only counting. **BM25** is an improved version used in search engines.

NOT EVERY WORD IS EQUALLY USEFUL

- If everyone in a room has brown hair, “has brown hair” does not help you find one person
- If only one person has a pet parrot, “has a pet parrot” finds them at once
- Words work the same way: a word in every document tells you nothing; a word in only one document is a strong clue

TF-IDF: THE FORMULA, IN PLAIN WORDS

- **TF** (term frequency): how many times does the word appear in *this* document?
- **IDF** (inverse document frequency): how rare is the word across *all* documents?
- Multiply them: a word scores high only if it is present *and* rare

$$\text{idf}(\text{word}) = \log \left(\frac{\text{number of documents}}{\text{documents containing the word}} \right)$$

The full score

$$\text{tf-idf}(\text{word}, \text{document}) = \text{tf}(\text{word}, \text{document}) \times \text{idf}(\text{word})$$

TF-IDF: A WORKED EXAMPLE

- Five toy documents (the same ones as in the Retrieval Playground)
- The word `model` appears in 3 of them; `energy` appears in only 1 (`doc_energy`)

Word	In how many docs?	$\text{idf} = \log(5/\text{that number})$	
<code>model</code>	3 of 5	$\log(5/3) \approx 0.51$	common $\rightarrow$ low score
<code>energy</code>	1 of 5	$\log(5/1) \approx 1.61$	rare $\rightarrow$ high score

- Both words appear *once* in `doc_energy`, so $\text{tf} = 1$ for both
- $\text{tf-idf}(\text{model}, \text{doc_energy}) = 1 \times 0.51 = 0.51$
- $\text{tf-idf}(\text{energy}, \text{doc_energy}) = 1 \times 1.61 = 1.61$

The point

`energy` counts about 3 times more than `model`, though both appear once: it is rarer.

WHY IDF USES A LOGARITHM

- Imagine a large collection: a news archive with $N = 1,000,000$ articles

The word appears in	ratio $N \div$ that number	$\log(\text{ratio})$
1 article	1,000,000	13.8
1,000 articles	1,000	6.9
every article	1	0

- Ratio alone: the rarest word counts **1,000 times** more than a word in 1,000 articles. One rare word would decide the ranking.
- With the log: **13.8 vs. 6.9**. Rare words still count more, by a reasonable amount.
- In a small collection the log matters little: with 5 documents, the ratio is at most 5

SPARSE VS. DENSE RETRIEVAL

sparse (TF-IDF)



dense (embedding)



the same picture as Session 1's sparse and dense word vectors, now for whole documents

- Sparse: one number per word in the vocabulary, mostly zeros. Finds exact word matches only.
- Dense: a few hundred learned numbers. Can match a question that uses different words with the same meaning. Needs an embedding model.

Activity: Retrieval Playground (20–25 min, individual)

- Open `03b_retrieval_playground`. No GPU needed.
- Part A: rank five toy documents by shared words, then by TF-IDF
- Part B: rank them by cosine similarity of toy embeddings
- Compare: which method finds the right document for a question with different wording?



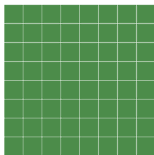
CHUNKING: LONG DOCUMENTS MUST BE SPLIT

- An embedding model reads a limited number of tokens. Ours reads at most **128**.
- Anything longer is cut off before embedding. The end of the text is not represented.
- 23 of our 26 documents are longer than 128 tokens. You will see one miss caused by this.
- Real systems split long documents into **chunks**, embed each chunk, and retrieve chunks

THE PAYOFF: FEWER TOKENS TO READ



4 tokens: 16 pairs



8 tokens: 64 pairs

- Today: all 26 documents $\approx 4,200$ tokens; top 2 ≈ 400 tokens
- Attention: twice the tokens, four times the pairs
- The rest of the model grows linearly, and writing the answer costs the same: expect a smaller energy saving

WHY CHECKING EVERY DOCUMENT DOES NOT SCALE

- Our retriever compares the question with all 26 documents. With 26, that is instant.
- A search engine or a company archive can hold millions or billions of documents
- Comparing a question with every one of them would take far too long

EVIDENCE 1: READING EVERYTHING GIVES BETTER ANSWERS

- Li et al. (2024): each question is about one long source text, e.g. a whole book or a long report
- **Long context:** the entire source text goes into the prompt
- **RAG:** the text is cut into chunks of 300 words; only the 5 most relevant chunks go into the prompt
- Three strong models: GPT-4o, GPT-3.5-Turbo, Gemini-1.5-Pro
- Long context scored higher on average, for all three models

Source: Li, Li, Zhang, Mei & Bendersky, "Retrieval Augmented Generation or Long-Context LLMs? A Comprehensive Study and Hybrid Approach," EMNLP 2024 Industry Track (arXiv:2407.16833)

EVIDENCE 2: RETRIEVAL READS FAR LESS

10,000–100,000 tokens



long context

≈1,500 tokens



RAG

- Long context: the entire source text, 10,000 to 100,000 tokens per question
- RAG: 5 chunks of 300 words, about 1,500 tokens per question
- Both computation and API prices grow with the number of input tokens

EVIDENCE 3: MOST OF THE TIME, THE ANSWERS ARE THE SAME

- For **63%** of questions, RAG and long context gave exactly the same answer
- Their combined method, **Self-Route**: try RAG first; the model says if the retrieved text is enough; if not, use long context
- Self-Route stayed close to long-context quality and cut cost by 65% (Gemini-1.5-Pro) and 39% (GPT-4o)

WHAT CAN GO WRONG: THE RETRIEVER MISSES

- The baseline read every document, so the right one was always in the prompt
- A retriever can fetch the wrong documents. Then the right answer is not in the prompt.
- A wrong answer now has two possible causes: the retriever, or the model

NOT THE SAME LEVER AS LAST SESSION

- Session 2 changed **precision**: the same text, fewer bits per number
- Today changes **how much text**: the same precision, fewer tokens
- The two are independent. A system can use both.

RETRIEVAL WORKS FOR TRANSLATION TOO: KNN-MT

- A trained translation model gets a **datastore** of earlier translations
- At each word it writes, it looks up the most similar contexts in the datastore
- It combines their suggested next word with its own prediction

Source: Khandelwal, Fan, Jurafsky, Zettlemoyer & Lewis, "Nearest Neighbor Machine Translation," ICLR 2021

OUR VERSION: A SMALL TRANSLATION MEMORY

- kNN-MT: millions of entries, a lookup at every word
- Ours: six Galician sentences with their English translations
- We retrieve the one sentence most similar to our passage, once, and show it to the model as an example
- Then we compare translations with and without the example, using four metrics

THE TRADE-OFF, PLAINLY

Cost	retrieval reads far fewer tokens, so it is much cheaper per question
Quality	with strong models, reading everything scored higher on average
Most questions	both give the same answer, so retrieval saves cost at no loss
New risk	the retriever can miss the right document

Activity: Discussion: when would you *not* retrieve? (10 min)

- When would you choose to give the model everything, on purpose?
- What would make you trust the retriever's single top document enough to show the model nothing else?
- Our 26 documents are short and each is about one topic. What goes wrong first with a large, messy collection?



THE SAME PIECES IN OUR NOTEBOOK

Real system	Notebook 3
knowledge base	CORPUS: the 26 documents
embedding model	paraphrase-multilingual-MiniLM
index	doc_embeddings: computed once
search	retrieve(): cosine similarity, keep the top 2
augmented prompt	build_prompt(): documents above the question
language model	Qwen2.5-1.5B-Instruct

WHAT WE'RE RUNNING TODAY

- 26 documents and 26 questions: Session 1's eight, plus 18 new
- The same model answers the quiz three times: **no documents, all 26 documents, the top 2 retrieved**
- Dense retrieval vs. TF-IDF: which finds more of the right documents?
- Translation with a retrieved example, scored with four metrics

