University of Ljubljana, Faculty of Computer and Information Science

Neural information retrieval

Prof Dr Marko Robnik-Šikonja

Natural Language Processing, Edition 2026

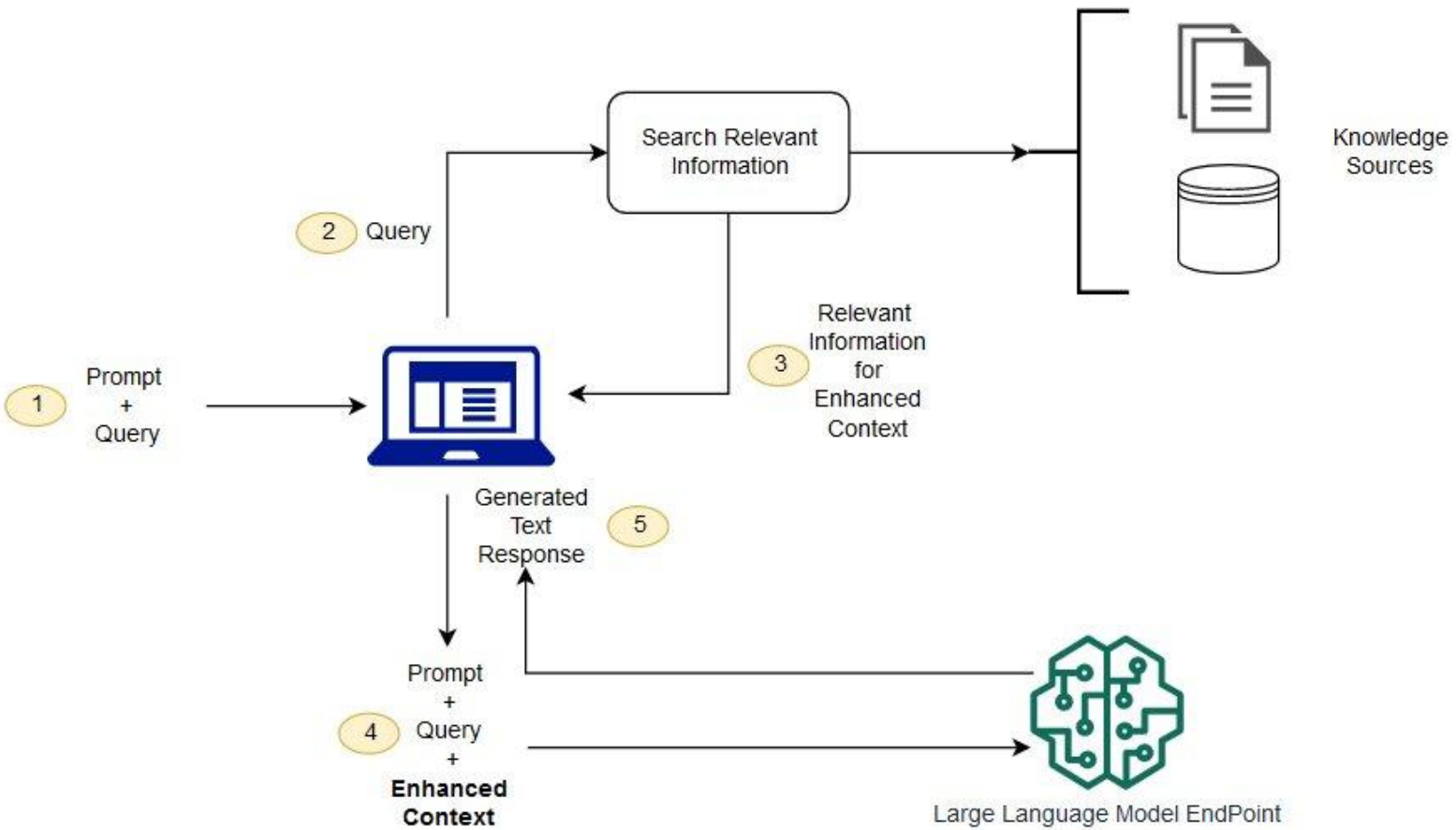
Contents

- information retrieval for LLMs
- neural text ranking

Retrieval Augmented Generation (RAG)

- For complex and knowledge-intensive tasks, LLM accesses external knowledge sources to complete tasks.
- Improves factual consistency, and reliability of the generated responses, reduces hallucinations
- RAG takes input and retrieves a set of relevant/supporting documents given a source (e.g., Wikipedia or internal documents). The documents are concatenated as context with the original input prompt and used as the input to LLM which produces the final output.
- RAG adapts to dynamic situations (facts could evolve over time)
- successful in QA

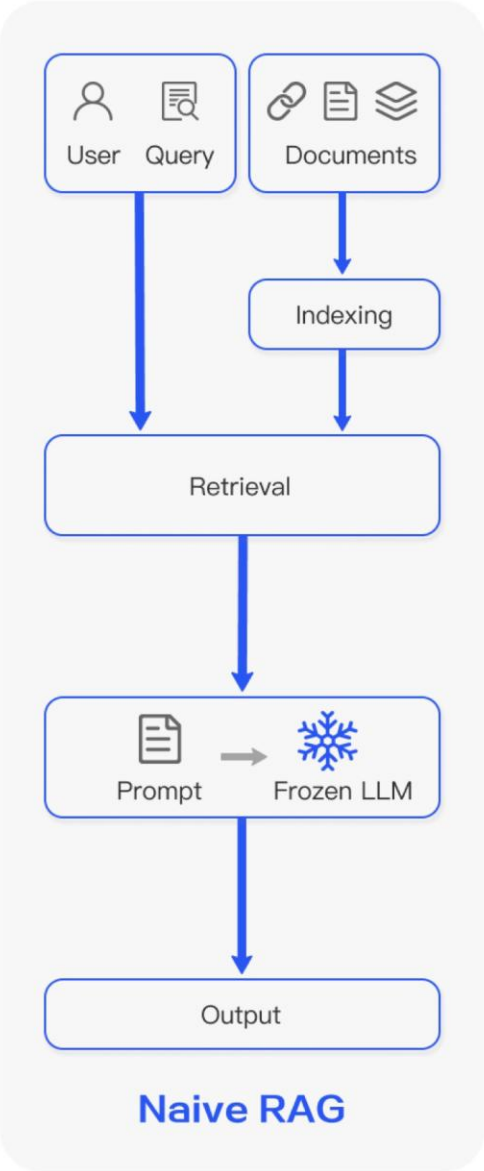
RAG schema

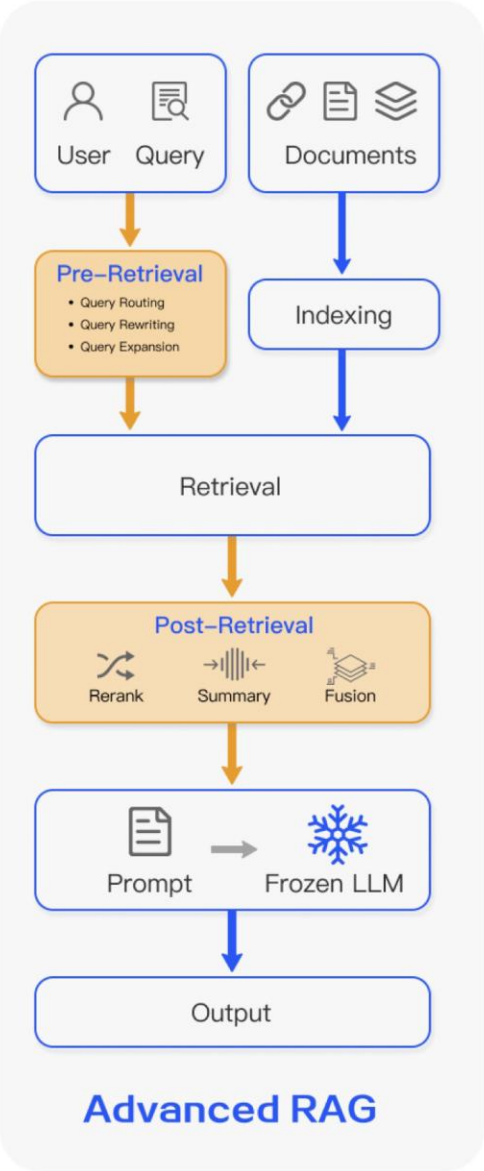

Search Relevant Information
Knowledge Sources
2 Query
Relevant Information for Enhanced Context
3
1 Prompt + Query
Generated Text Response
5
Prompt + Query + Enhanced Context
4
Large Language Model EndPoint

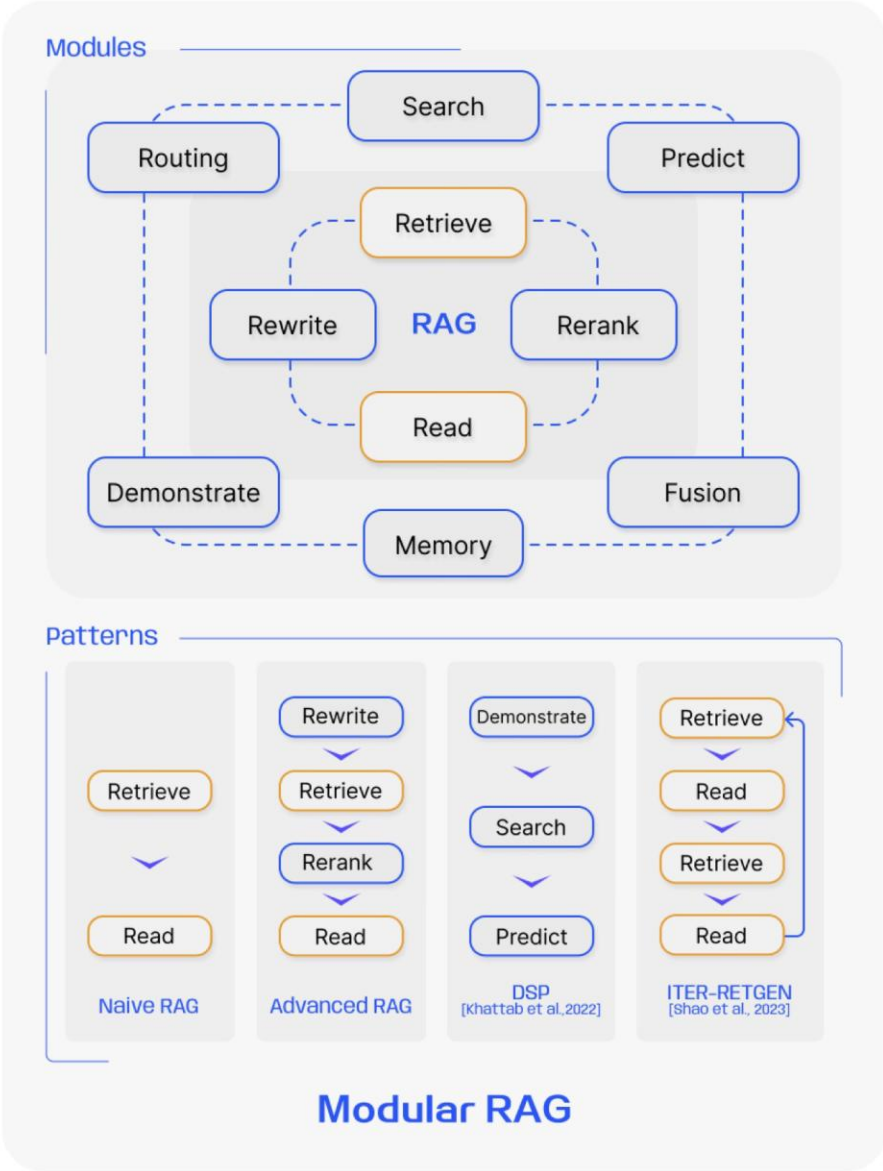
RAG summarized

- **Why RAG?** Ground LLM reasoning in *external* knowledge to improve factuality, recency, provenance
  - **How it works:** *Embed → Retrieve → (Re-rank) → Generate → Cite*
  - **Encode documents** → store in vector DB (ANN index)
  - **Encode query** → search top-k nearest vectors via ANN (Approximate NN)
  - **Return candidates** → re-rank or feed to generator
- **Key levers:** Chunking, embedding choice, index type (HNSW/IVF-PQ), filters, hybrid retrieval, re-ranking (cross-encoders), answer grounding
- **Measure what matters:** retrieval quality (Recall@k, nDCG), answer groundedness & citation quality (RAG-specific evals)

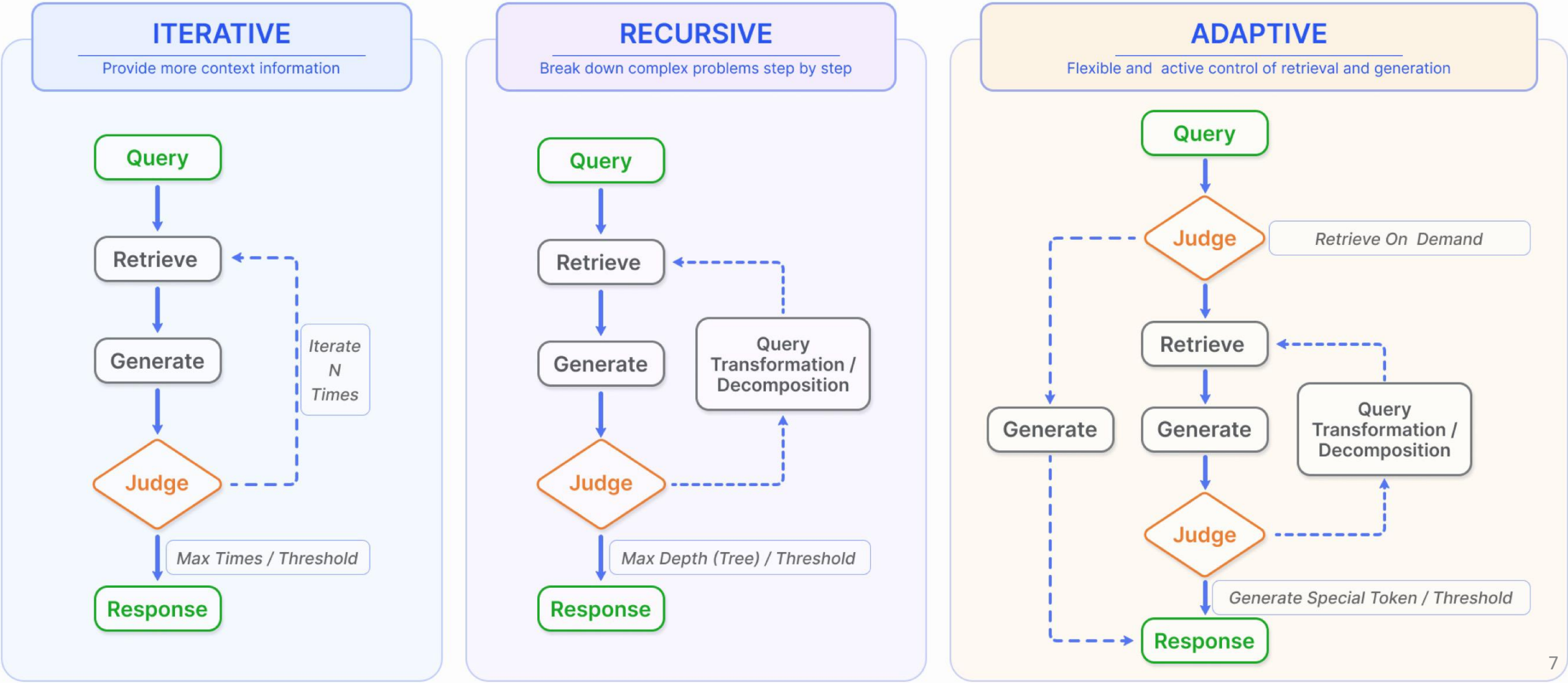
RAG improvements





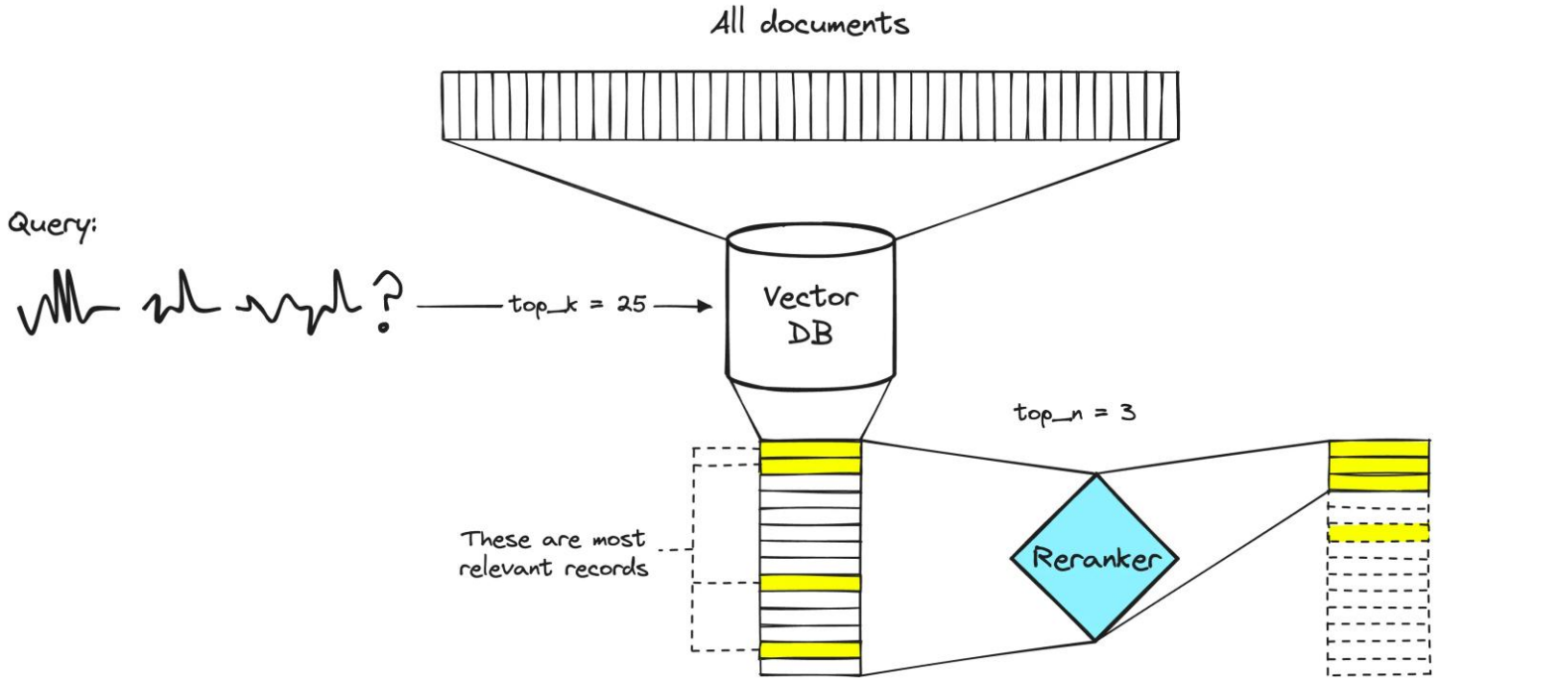


RAG variants



Re-ranking

- After a retriever returns the top-k candidate documents for a query, we may want to sort them by how relevant they truly are



Toolkits for Generation Evaluation

- **RAGAS – *Retrieval Augmented Generation Assessment Suite***
- Developed by Hugging Face & Intel Labs.
- Automated scoring of RAG outputs across:
  - *Faithfulness* (are answers grounded?)
  - *Answer relevance*
  - *Context recall* (did the retriever fetch needed info?)
  - *Context precision* (was retrieved info actually used?)
- Uses a smaller LLM or NLI (natural-language inference) model for evaluation.
- Produces interpretable scores per query.
- Integrates easily into LangChain, LlamaIndex, or Hugging Face pipelines

Beyond textual data

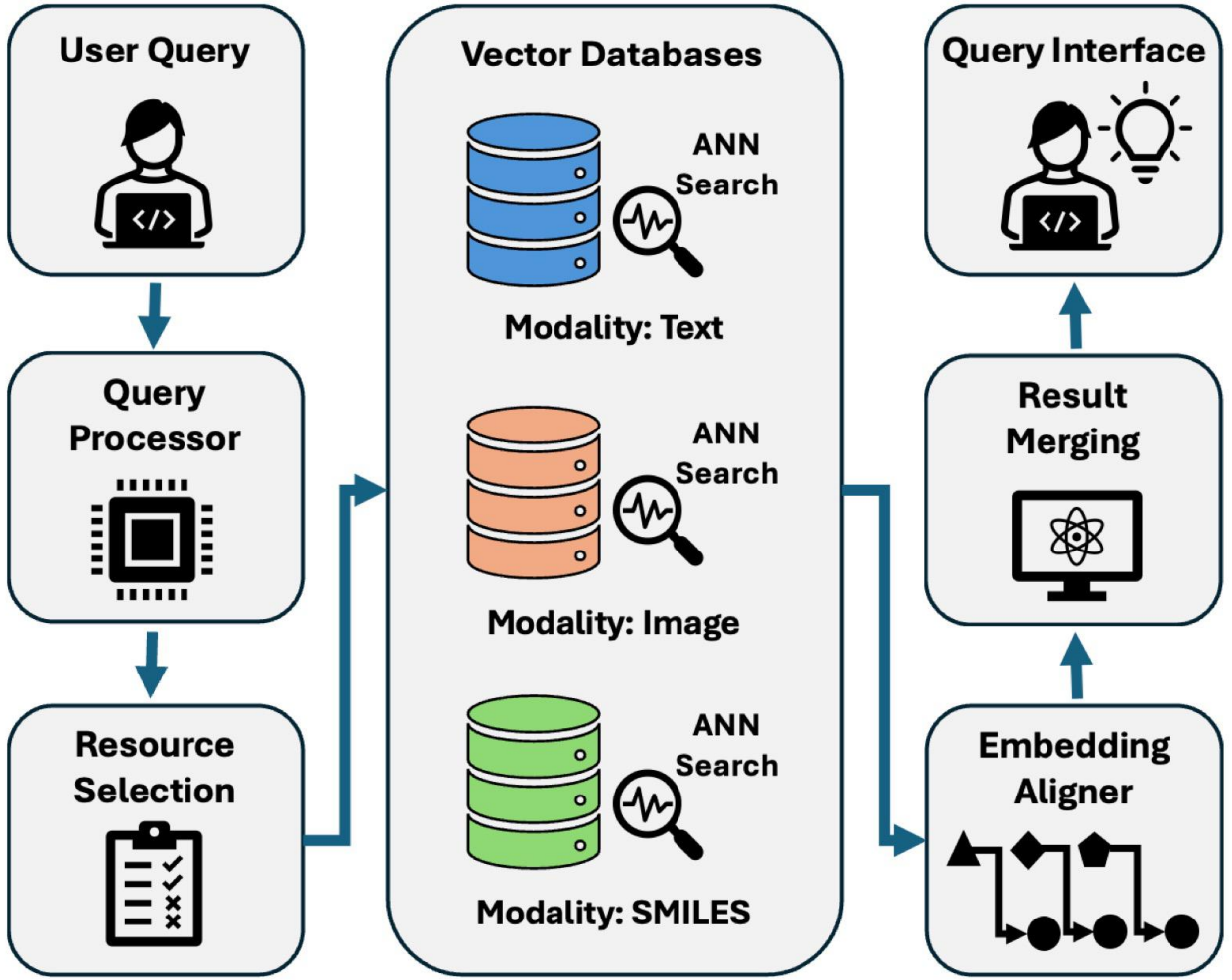
Data type	Examples	What the LLM must learn/do
Structured	Databases, CSVs, SQL engines	Generate and interpret queries, aggregate results
Temporal / time-series	Sensors, logs, climate records	Learn temporal reasoning; perform trend analysis
Spatial / geospatial	Maps, satellite imagery	Call GIS APIs; reason with coordinates and regions
Scientific / numeric	Simulation codes, lab data, HPC runs	Request parameters, launch computations, interpret results
Multimodal	Images, spectra, molecules, graphs	Use embeddings or model adapters per modality
Private / dynamic	Company or lab databases, instruments	Handle authentication, provenance, and freshness

Multimodal RAG

- Example:

SMURF: Scientific MUltimodal Retrieval in Federations

- Show me examples of catalysts that perform well at 200 °C for CO_2 conversion, and include any microscopy images showing the catalyst surface.”



RAG pitfalls

- **Context window overflow / truncation**
  When too many retrieved tokens exceed the LLM's context limit, older or less-recent chunks get silently dropped, causing loss of critical information or incoherent answers
- **Irrelevant retrieval hurting generation**
  Poorly matched passages can mislead the model's reasoning, pulling the answer toward unrelated topics or introducing factual noise
- **Citation hallucination**
  The model fabricates references or associates statements with the wrong source, creating false confidence and undermining provenance
- **Embedding drift when corpora evolve**
  Updating or re-embedding documents with a new model or altered data distribution changes their vector geometry, breaking previous similarity relationships and degrading retrieval accuracy

Obtaining relevant context for a query

- a part of traditional information retrieval
- But still relevant for LLMs, e.g., in RAG
- The context can constitute a part of the prompt to LLM
- Well-known approaches to be presented
  - BM25 (Best match 25)
  - DPR (Dense Passage Retrieval)
  - Dot product on sentence encoders, e.g., LaBSE or Sentence-BERT
  - CovBERT

Ranking documents with BM25

- Okapi BM25 (Best match 25)
- uses bag-of-words document representation, works similarly to TF-IDF weighting
- Given a query Q, with words q_1,..., q_n the BM25 score of a document D is:

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

- $f(q_i,D)$ is the number of times that q_i occurs in D,
- avgdl is the average document length in the text collection
- k_1 and b are parameters, usually chosen from $k_1 \in [\ 1.2\ ,\ 2.0\ ]$ and $b = 0.75$

IDF variant in BM25

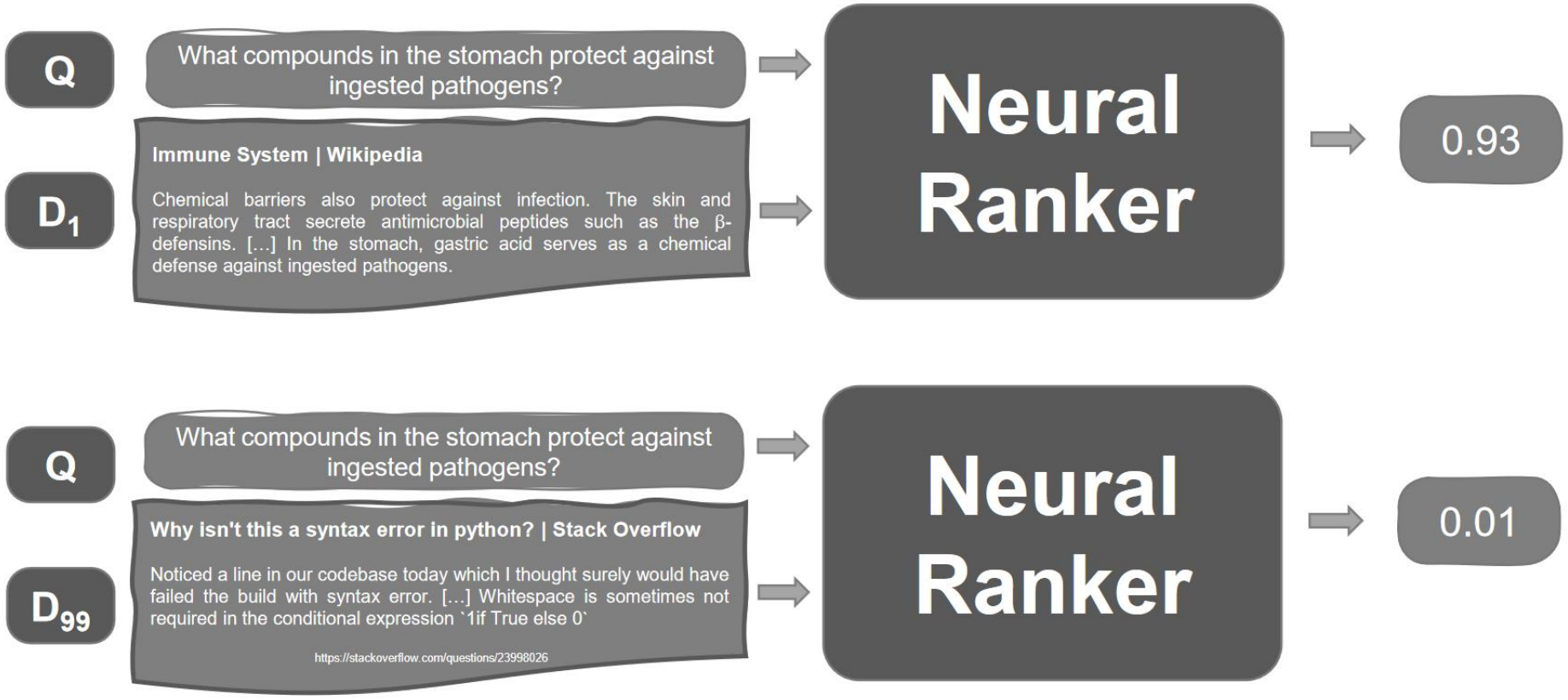
- IDF (inverse document frequency) weights the query term q_i

$$\mathrm{IDF}(q_i) = \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

- where N is the total number of documents in the collection, and $n(q_i)$ is the number of documents containing q_i

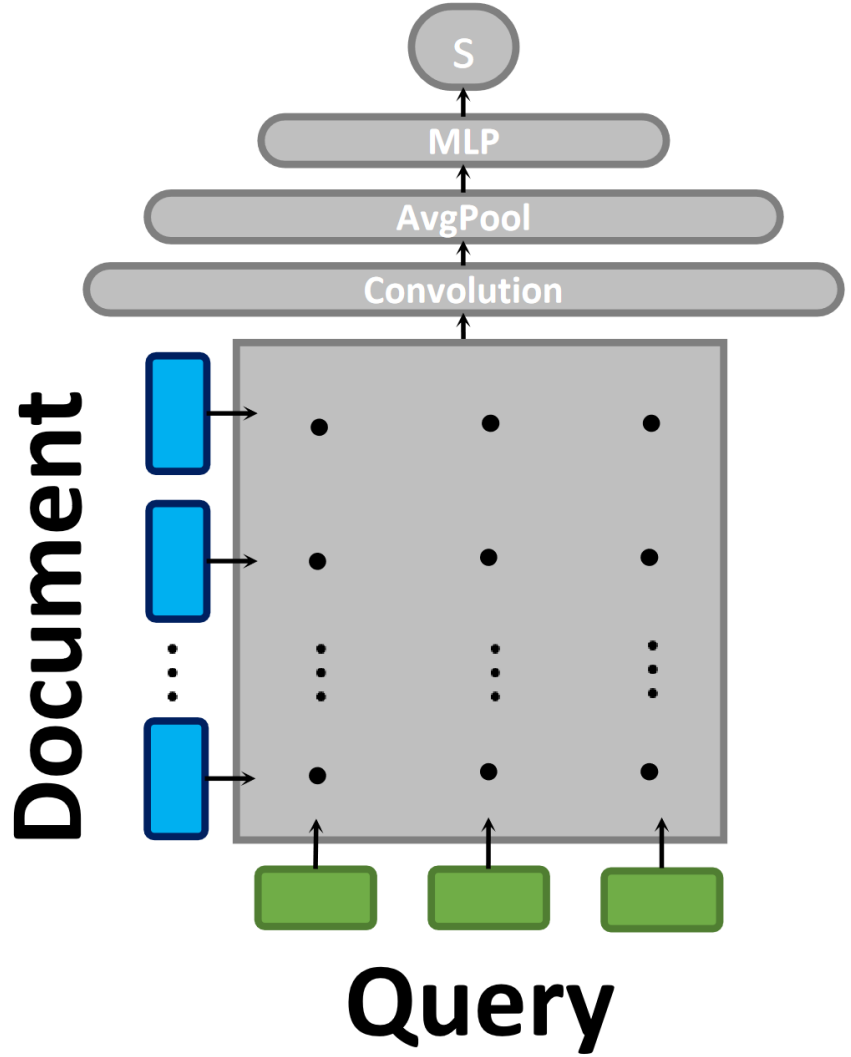
Neural Ranking

- Use neural representations



Query-Document Interaction Approach

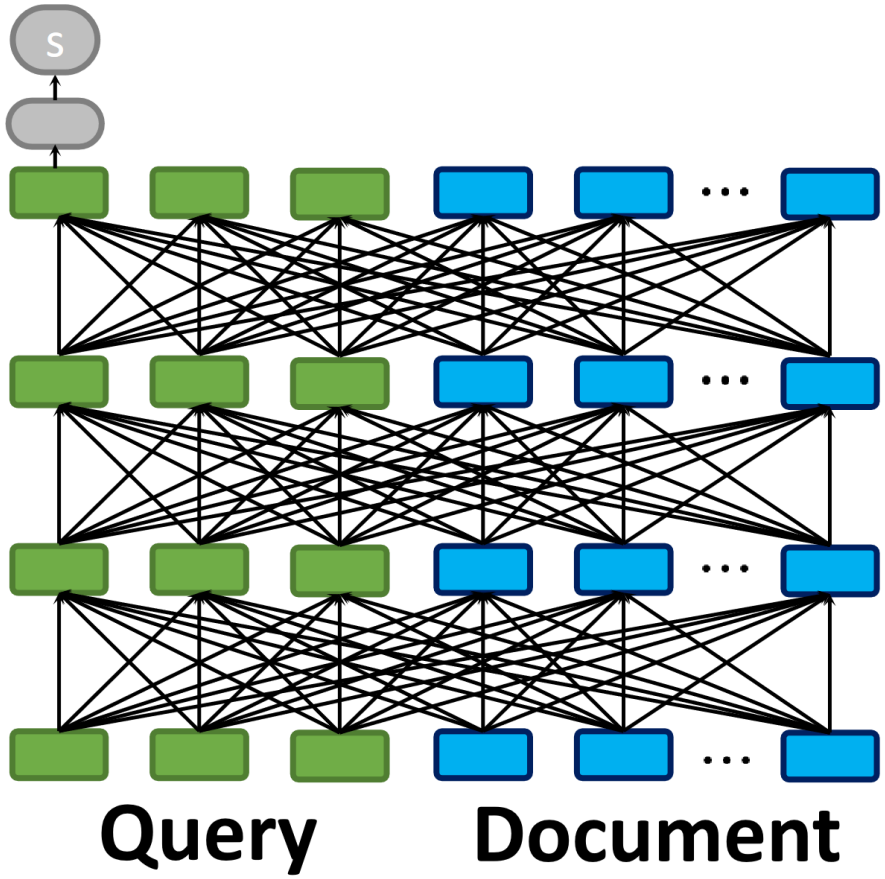
- IR Ranking refers to scoring query-document pairs, sorting them in descending order, and then getting the top K results:
  - Tokenize query and documents
  - Embed tokens to vector
  - Make query-document interaction matrix and compute cosine similarity for each pair of words.
  - Compress the matrix into a score. Use a neural layer (convolution, linear layers)
- considerably better than non-neural methods but computationally expensive
- why?



All-to-all Interaction with BERT

- 1. Feed BERT “[CLS] Query [SEP] Document [SEP]”
- 2. Run this through all the BERT layers
- 3. Extract the final [CLS] output embedding
- 4. Reduce to a single score through a linear layer

- This is essentially a standard BERT classifier, used for ranking passages.
- We must fine-tune BERT for this task with positives and negatives to be effective
- Much better quality—but also a dramatic increase in computational cost
- How to get a better query latency?

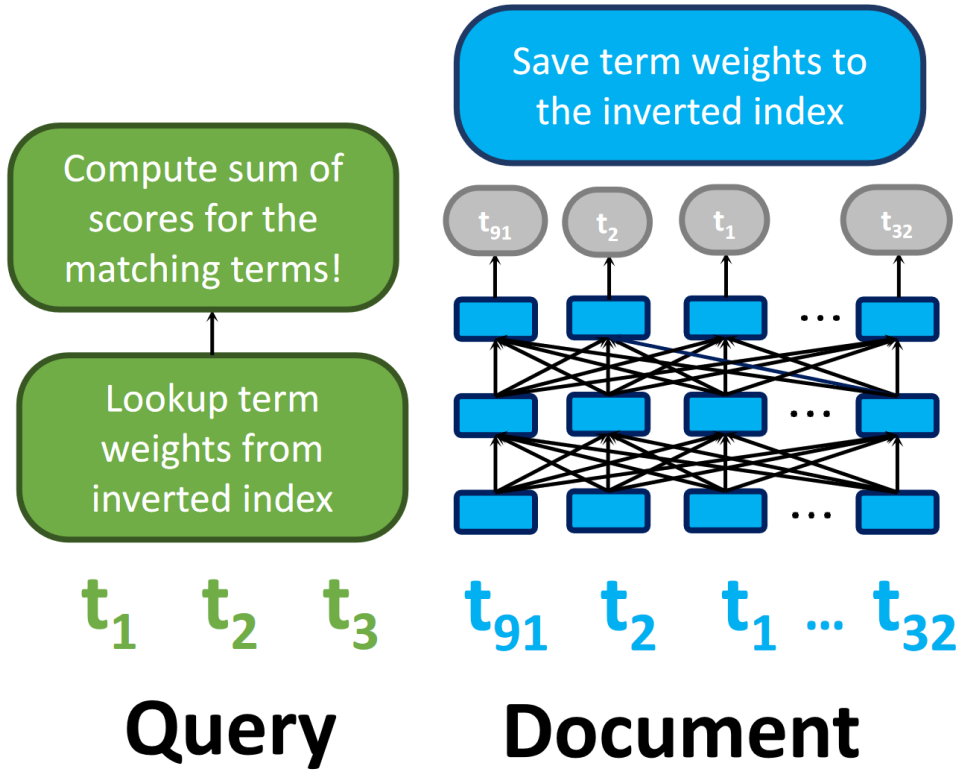


Faster IR: precomputing

- Is there a value in jointly representing queries and documents?
- BERT rankers are slow because their computations can be redundant:
  - Represent the query (1000 times for 1000 documents)
  - Represent the document (once for every query!)
  - Conduct matching between the query and the document
- We have the documents in advance.
  - Can we pre-compute the document representations?
  - And “cache” these representations for use across queries

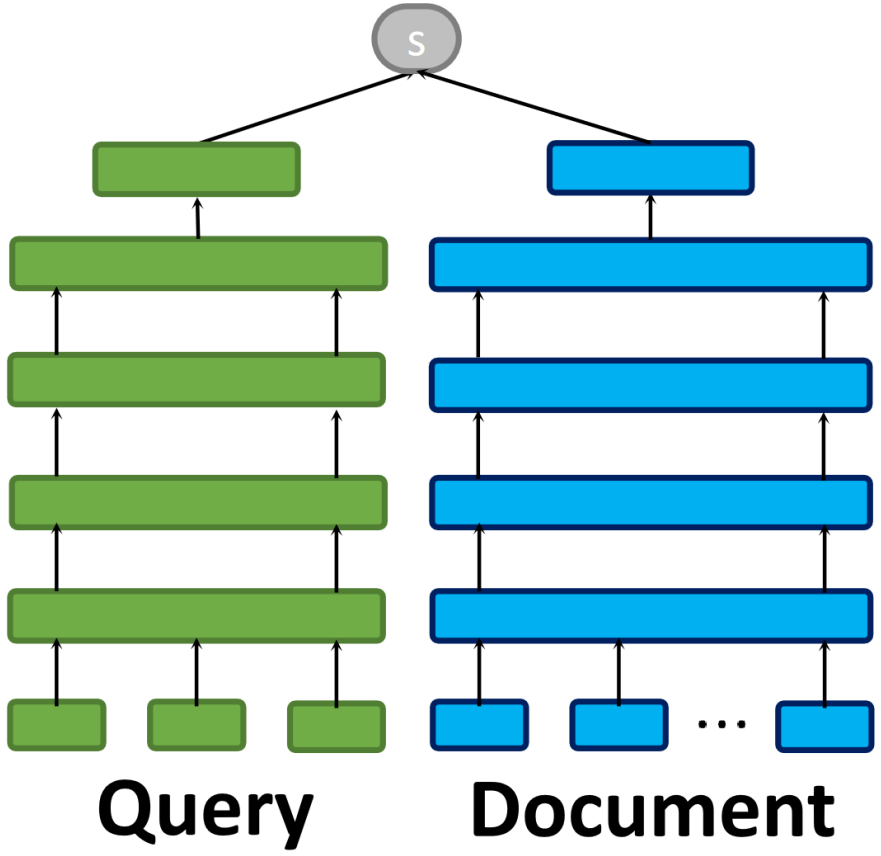
A bad solution: Neural bag-of-words

- BM25 decomposed a document's score into a summation over term–document weights. Can we learn term weights with BERT?
- Tokenize the query/document
- Use BERT to produce a score for each token in the document
- Add the scores of the tokens that also appear in the query



Neural IR: Representation similarity

- Tokenize the query and the document
- Independently encode the query and the document into a single-vector representation each
- Estimate relevance as a dot product or a cosine similarity
- Like learning term weights, this paradigm offers strong efficiency advantages:
  - Document representations can be pre-computed!
  - Query computations can be amortized.
  - Similarity computations are very cheap.



Example of representation similarity: Dense Passage Retrieval (DPR)

- BERT based passage retrieval
- Encodes each passage and each query into a 768-dimensional vector
- Ranks passages in the document collection relative to query *q* using dot product similarity
- BERT is additionally pretrained to maximize the similarity between *q* and correct passages and minimize the similarity between *q* and wrong passages using the loss:

$$
\begin{aligned}
& L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) \\
= \quad & -\log \frac{e^{\mathrm{sim}(q_i, p_i^+)}}{e^{\mathrm{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\mathrm{sim}(q_i, p_{i,j}^-)}}
\end{aligned}
$$

- A negative passage is sampled from BM25 top-100
- Passages and query are encoded with modified BERT (using the CLS token representation) and then ranked based on the dot product similarity

Karpukhin et al (2020) Dense passage retrieval for open-domain question answering. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 6769–6781.

Sentence-BERT

- Use a **contrastive loss**

```
L = max(0, d(anchor, positive) - d(anchor, negative) + margin)
```

- Given pairs (anchor, positive, negative):
  - anchor = "What is photosynthesis?"
  - positive = "The process by which plants make food using sunlight."
  - negative = "The capital of France is Paris."
- The model encodes all three and minimizes:
  - distance(anchor, positive) << distance(anchor, negative)

LaBSE sentence encoder

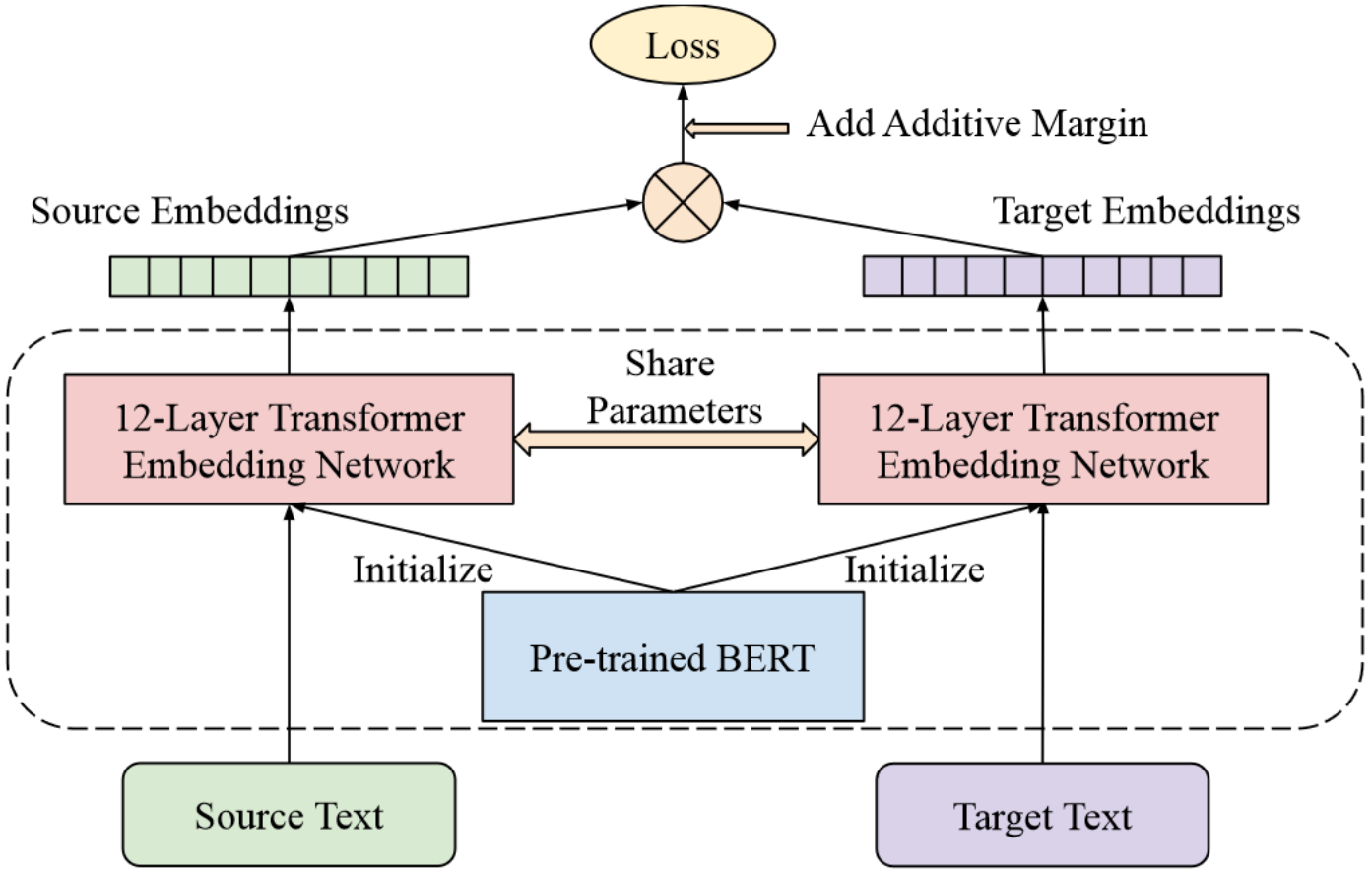
- LaBSE (Language-agnostic BERT Sentence Encoder)
- dual-encoder architecture, where source and target sentences (in different languages) are encoded separately using a shared BERT-based encoder
- pre-trained on masked language modeling and translated language modeling
- supports 109 languages
- allows finding similar sentences across different languages.
- loss

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} log \frac{e^{\phi(x_i,y_i)}}{e^{\phi(x_i,y_i)} + \sum_{n=1,n\neq i}^{N} e^{\phi(x_i,y_n)}}$$

Feng, F., Yang, Y., Cer, D., Arivazhagan, N. and Wang, W., 2022. Language-agnostic BERT Sentence Embedding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 878-891). https://arxiv.org/abs/2007.01852 https://tfhub.dev/google/LaBSE

LaBSE architecture

- Dual encoder model with BERT based encoding modules.

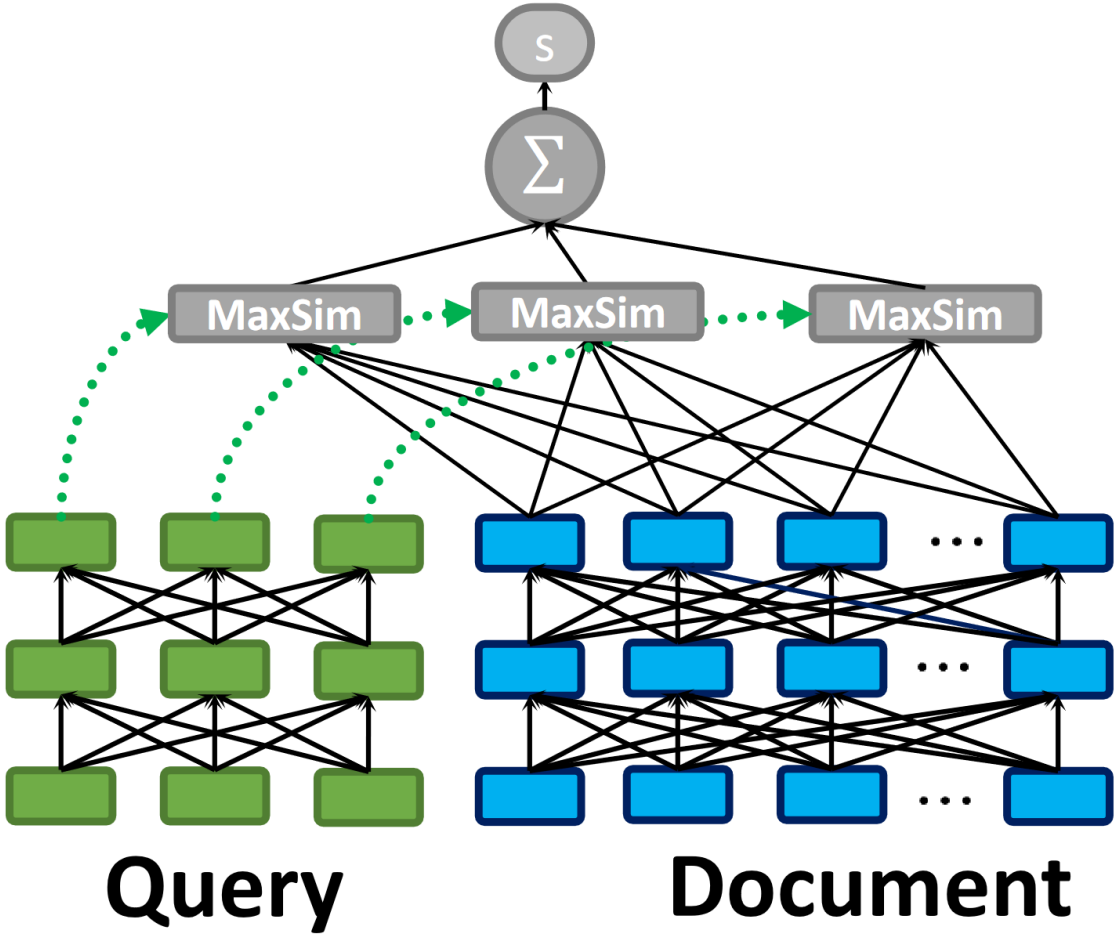


Representation Similarity: Downsides

- Single-Vector Representations “cram” queries and documents into a coarse-grained representation!
- No fine-grained interactions
- They estimate relevance as a single dot product!
- We lose term-level interactions, which we had in query–document interaction models (e.g., BERT) and even term-weighting models (e.g., BM25)
- Can we keep precomputation and still have fine-grained interactions?

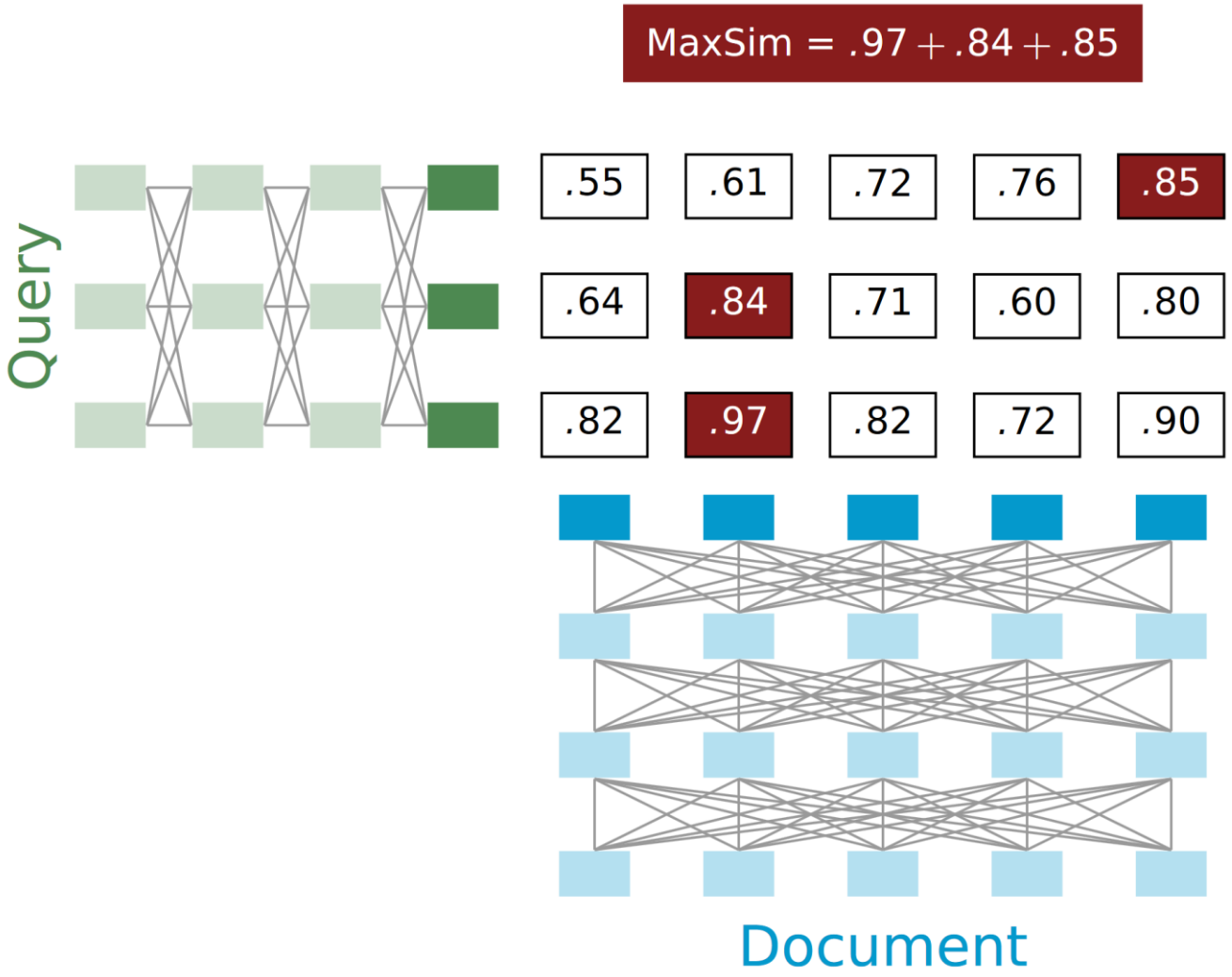
Neural IR: Late interactions

- Independent Encoding
- Fine-Grained Representations
- End-to-End Retrieval
- ColBERT represents the document as a MATRIX, not a vector



Omar Khattab and Matei Zaharia. “ColBERT: Efficient and effective passage search via contextualized late interaction over BERT." SIGIR’20

ColBERT: MaxSim



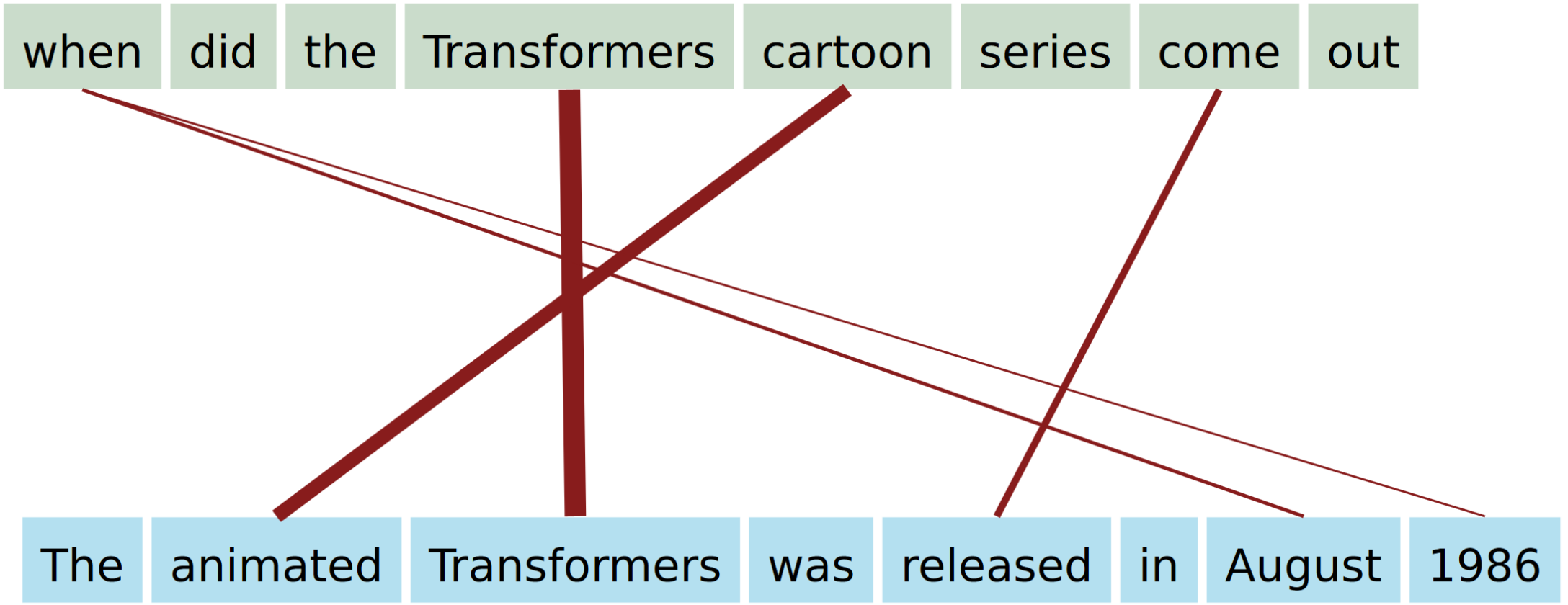
1. Examples: $\langle q_i, \text{doc}_i^+, \{\text{doc}_{i,k}^-\}\rangle$
2. Loss: negative log-likelihood of the positive passage, with **MaxSim** as the basis.

For a BERT-style encoder with N layers:

$$\textbf{MaxSim}(q, \text{doc}) = \sum_i^L \max_j^M \textbf{Enc}(q)_{N,i}{}^\top \textbf{Enc}(\text{doc})_{N,j}$$

with L is the length of q, M the length of doc.

Soft alignment with ColBERT



Common Evaluation Metrics for IR

1. *Accuracy* (does answer match gold-labeled answer?)
2. *Mean Reciprocal Rank (MRR)*
   - For each query return a ranked list of M candidate answers.
   - Query score is 1/Rank of the first correct answer
     - *If first answer is correct: 1*
     - *else if second answer is correct: ½*
     - *else if third answer is correct: ⅓, etc.*
     - *Score is 0 if none of the M answers are correct*
   - Take the mean over all N queries

$$MRR = \frac{\sum_{i=1}^{N} \frac{1}{rank_i}}{N}$$

IR evaluation datasets

- Text REtrieval Conference (TREC) has annual competitions for comparing IR systems.
- MS MARCO Ranking is the largest public IR benchmark.
  - It is adapted from a Question Answering dataset
  - It consists of more than 500k Bing search queries
  - Passage Ranking: 9M short passages; sparse labels
  - Document Ranking: 3M long documents; sparse labels
- Many others

FAISS: Facebook AI Similarity Search

- An open-source C++/Python library for large-scale vector search
- Supports multiple ANN algorithms (including IVF, HNSW, Product Quantization)
- Runs on CPU or GPU, enabling extremely fast batched searches
- Provides exact search for small datasets and compressed approximate search for billion-scale corpora.

Type	Description	When to use
Flat (brute-force)	Exact search; O(N)	Small datasets (≤100k vectors)
IVF (Inverted File Index)	Clusters vectors into “centroids”; search only nearby clusters	Millions of vectors
IVF-PQ (Product Quantization)	Compress vectors for memory efficiency	Billion-scale datasets
HNSW	Graph-based ANN for high recall	Mid-sized (up to 100M) datasets