BIP LanguageModels and KnowledgeGraphs

From Information Retrieval to Retrieval Augmented Generation

Davide Buscaldi

LIPN, Université Sorbonne Paris Nord

DIX, Ecole Polytechnique

Cagliari, BIP Summer School “LLMs and Knowledge Graphs”

Introduction

- What is Information Retrieval?
  - **Information Retrieval** (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers). —*Chris Manning*
- Computer Science changed the context
  - From specialists/professionals (librarians, lawyers,...)
  - To everyday users, searching the web or their e-mails

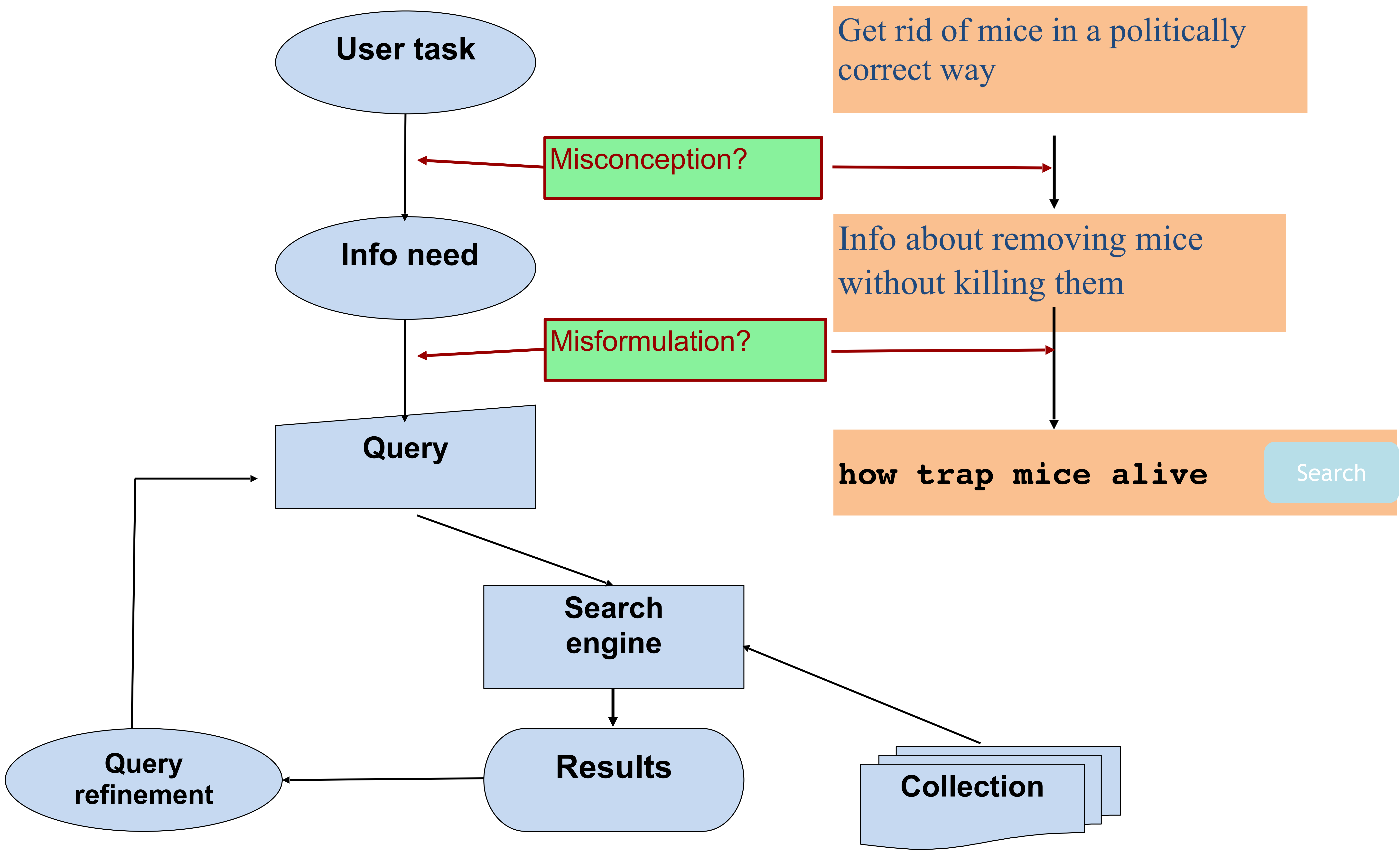
Some (classic) search engines

- Lucene (Java, Apache)
  - http://lucene.apache.org/core/
- Solr (web SE based on Lucene) -> ElasticSearch
  - http://lucene.apache.org/solr/
  - https://www.elastic.co/fr/elasticsearch
- Terrier (Java, Glasgow University)
  - http://terrier.org/
- Xapian (various languages, independent)
  - https://xapian.org/
- Whoosh! (Python, independent)
  - https://pypi.python.org/pypi/Whoosh

Information need

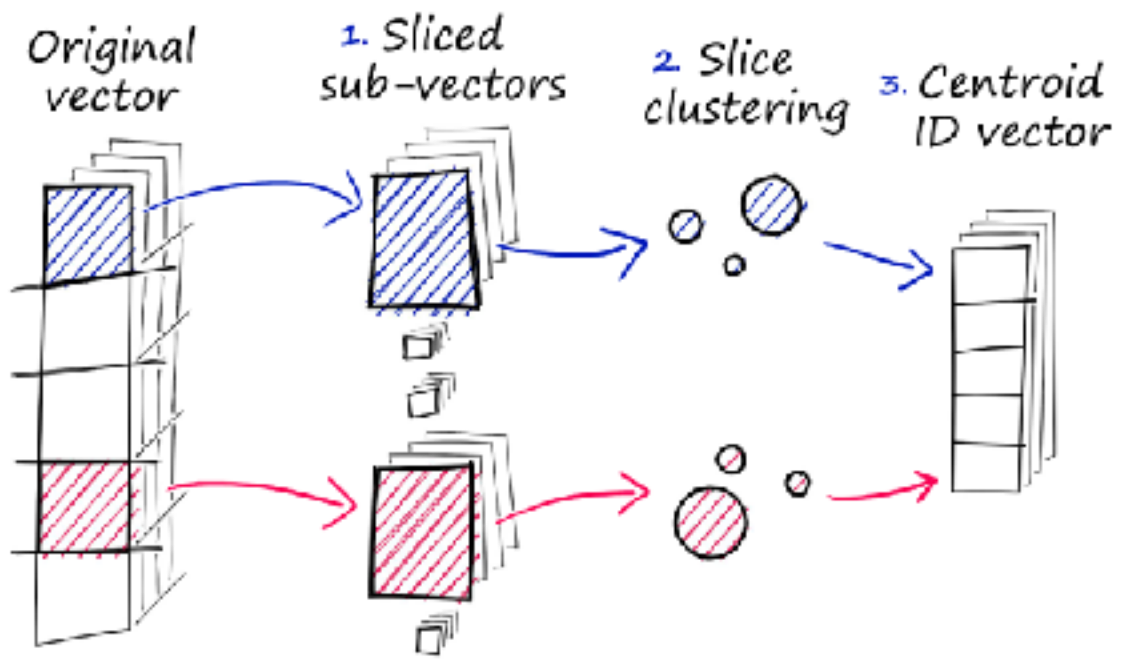
- **Goal**: Retrieve documents with information that is relevant to the user's information need and helps the user complete a task
- How to specify one's information need?
  - **Query**: list of keywords or a question in natural language
    - Example: *"U.S. presidential elections"*, *"list of football players who played both in Milan AC and Inter Milan"*

The classic search model



Indexing

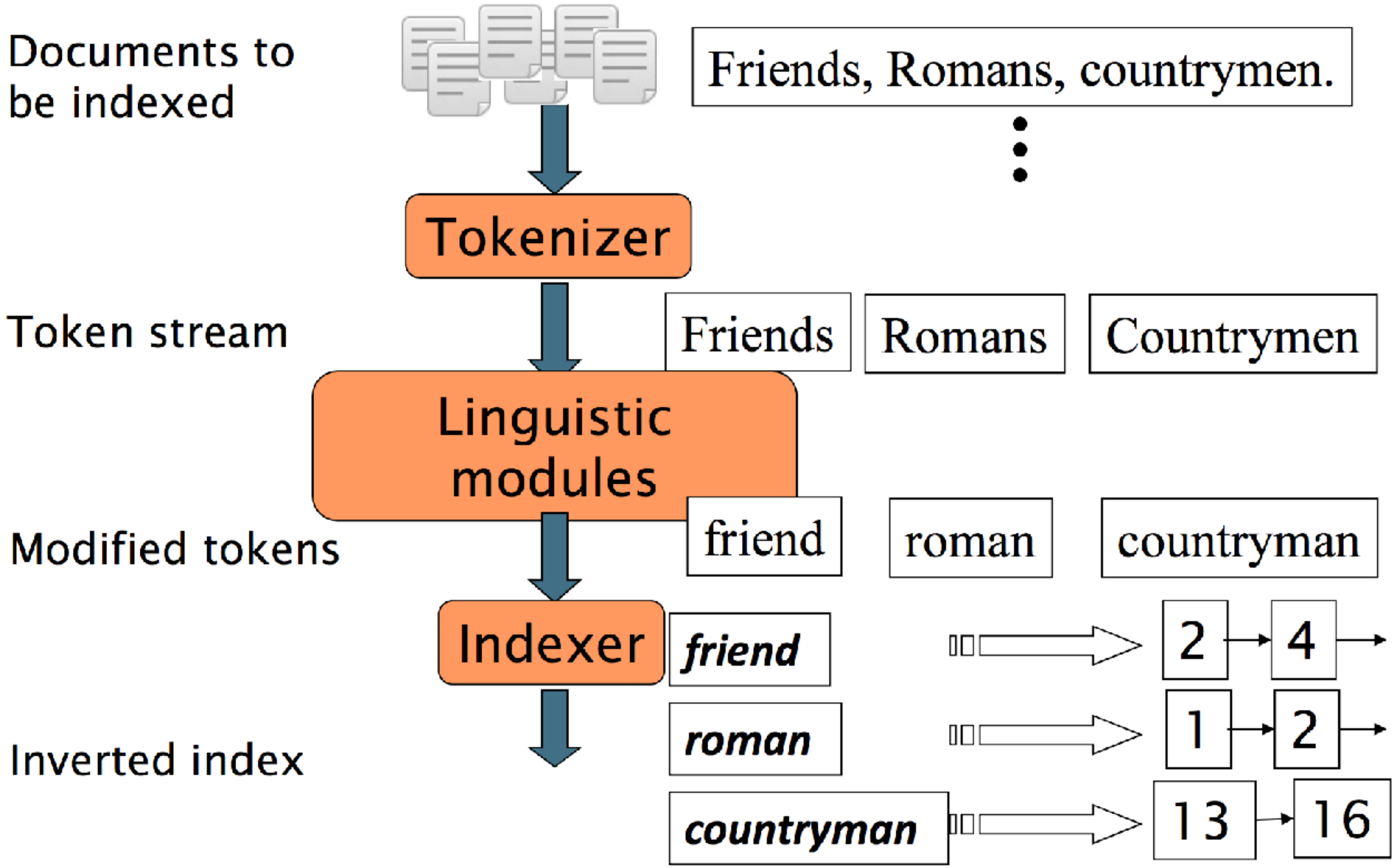
- Preliminary step to be able to search a collection
  - Transformation of documents into structures that could be searched **efficiently**
- Main types:
  - B-trees, inverted indexes (classic indexing)
  - Structured vector repositories (dense retrieval)
    - Example: FAISS (Facebook)



Inverted index

Term	Doc_1	Doc_2
Quick		X
The	X	
brown	X	X
dog	X	
dogs		X
fox	X	
foxes		X
in		X
jumped	X	
lazy	X	X
leap		X
over	X	X
quick	X	
summer		X
the	X	

Classic Indexing Pipeline



Stemming

- Reduce terms to their “roots” before indexing
- “Stemming” suggests crude affix chopping
- Language dependent
- e.g., automate(s), automatic, automation all reduced to automat.

for example compressed and compression are both accepted as equivalent to compress.	for exampl compress and compress ar both accept as equival to compress

Stemming Algorithms

- Porter stemmer (English)
  - sses → ss
  - ies → i
  - ational → ate
  - tional → tion
  - Weight of word sensitive rules
    - (m>1) EMENT →
    - replacement → replac
    - cement → cement
- Paice-Husk
- Lovins
- Snowball (http://snowball.tartarus.org/)
  - Used in Lucene, many languages

Sparse Retrieval

Classic Retrieval Models
(“Sparse” Retrieval)

- Boolean Model
- Vector Models
  - tf.idf, BM25
- Graph-based models
  - PageRank, HITS…

The Boolean model

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
- Boolean Queries are queries using AND, OR and NOT to join query terms
  - It views each document as a **set of words**
  - It is **precise**: document matches condition or not.
- Perhaps the simplest model to build an IR system on
- Many search systems you still use are Boolean: Email, library catalog, Mac OS X Spotlight
- <u>Example</u>: All the Shakespeare works containing “Caesar” and “Brutus” but not “Calpurnia”

Term-document Incidence Matrix

- Example: Shakespeare’s works

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

110100
110111
101111

100100

Ranked Retrieval

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query
- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
    - We just show the top k (≈ 10) results
- Idea —> Assign to documents a score [0, 1]
  - How to score documents?

Term Frequency

- The first idea is to capture the “strength” of a word in a document by its frequency in a document:
  - The term frequency $tf_{t,d}$ of term **t** in document **d** is defined as the number of times that **t** occurs in **d**.
- We replace the boolean values in the incidence matrix with $tf_{t,d}$:

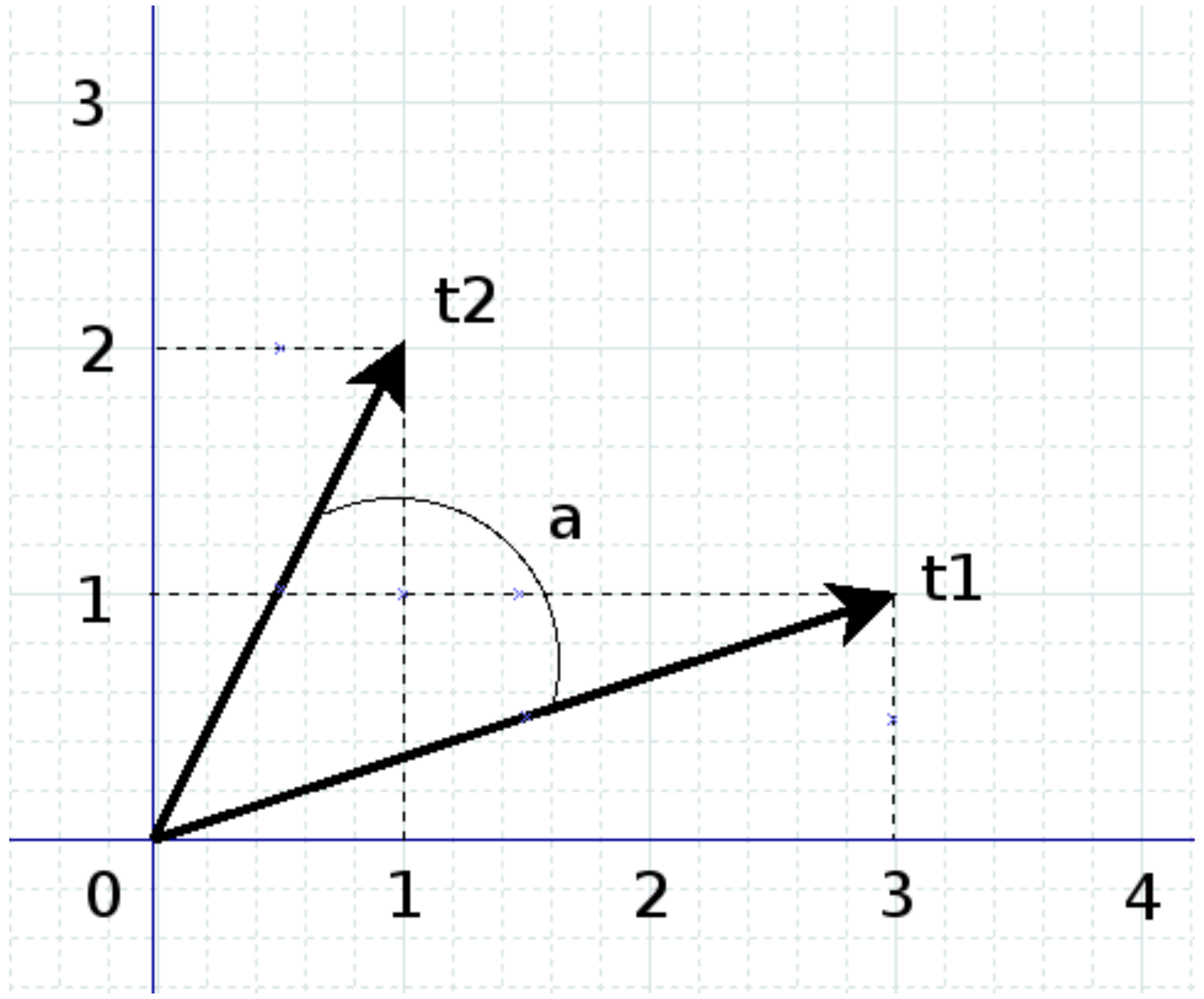
Term-document count matrices

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Documents as vectors
(Salton, 1968)

- Every document is represented by a vector in the word space
- Example: space of 2 words (a1, a2) and 2 documents (t1, t2):

	t1	t2
a1	3	1
a2	1	2



-

Comparing Documents

- Comparing Documents in a vector space can be done easily using vector distance or similarity measures
- Distance measures:
  - The more the differences in documents, the higher the score
- Similarity measures:
  - The more similar the documents, the higher the score

Distance measures

- Manhattan (*city block*) distance: $|t_1 - t_2| = \sum_{k=1}^{n} |v_{1,k} - v_{2,k}|$
  - for previous example: $|t_1 - t_2| = |3-1| + |1-2| = 2+1 = 3$
- Euclidean: $||t_1 - t_2|| = \sqrt{\sum_{k=1}^{n} (v_{1,k} - v_{2,k})^2}$
  - for previous example:

$$||t_1 - t_2|| = \sqrt{\sum_{k=1}^{n} (3-1)^2 + (1-2)^2} = \sqrt{2^2 + 1^2} = \sqrt{5}$$

Distance is a bad idea . . .
because distance is large for vectors of different lengths

Similarity Measures

- Dot product $t_1.t_2 = \sum_{k=1}^{n}(v_{1,k} * v_{2,k})$
  - dans notre cas, (3*1)+(2*1)
- Cosine similarity: $\frac{t_1.t_2}{||t_1||*||t_2||}$ où $||t_1|| = \sqrt{\sum_{k=1}^{n} v_{1,k}^2}$

Example

texte 1	"Le cinéma est un art, c'est aussi une industrie." (phrase célèbre d'André Malraux)
texte 2	"Personne, quand il est petit, ne veut être critique de cinéma. Mais ensuite, en France, tout le monde a un deuxième métier : critique de cinéma !" (citation approximative de deux phrases de François Truffaut)
texte 3	"Tout le monde a des rêves de Hollywood."
texte 4	"C'est la crise, l'économie de la France est menacée par la mondialisation."
texte 5	"En temps de crise, reconstruire l'industrie : tout un art !"
texte 6	"Quand une usine ferme, c'est que l'économie va mal."

Example

- To simplify we remove stop-words and apply a dictionary-based normalisation:
  - cinéma <— art, hollywood, cinéma, critique
  - économie <— crise, économie, mondialisation, industrie, usine
- So we can work just on 2 dimensions, “cinéma” and “économie”

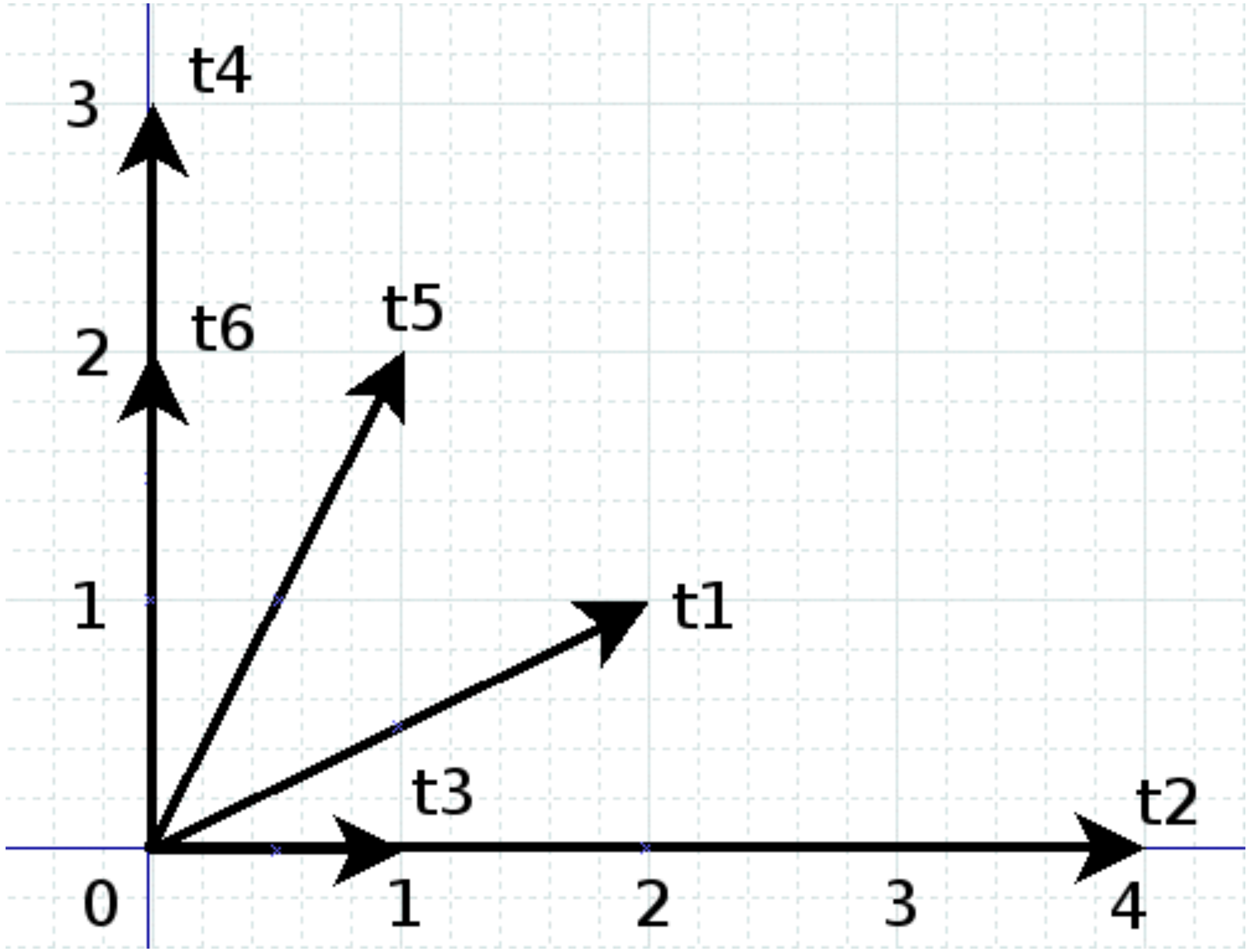
Example

Term-document matrix

	t1	t2	t3	t4	t5	t6
cinéma	2	4	1	0	1	0
économie	1	0	0	3	2	2

Example

Projection in vector space

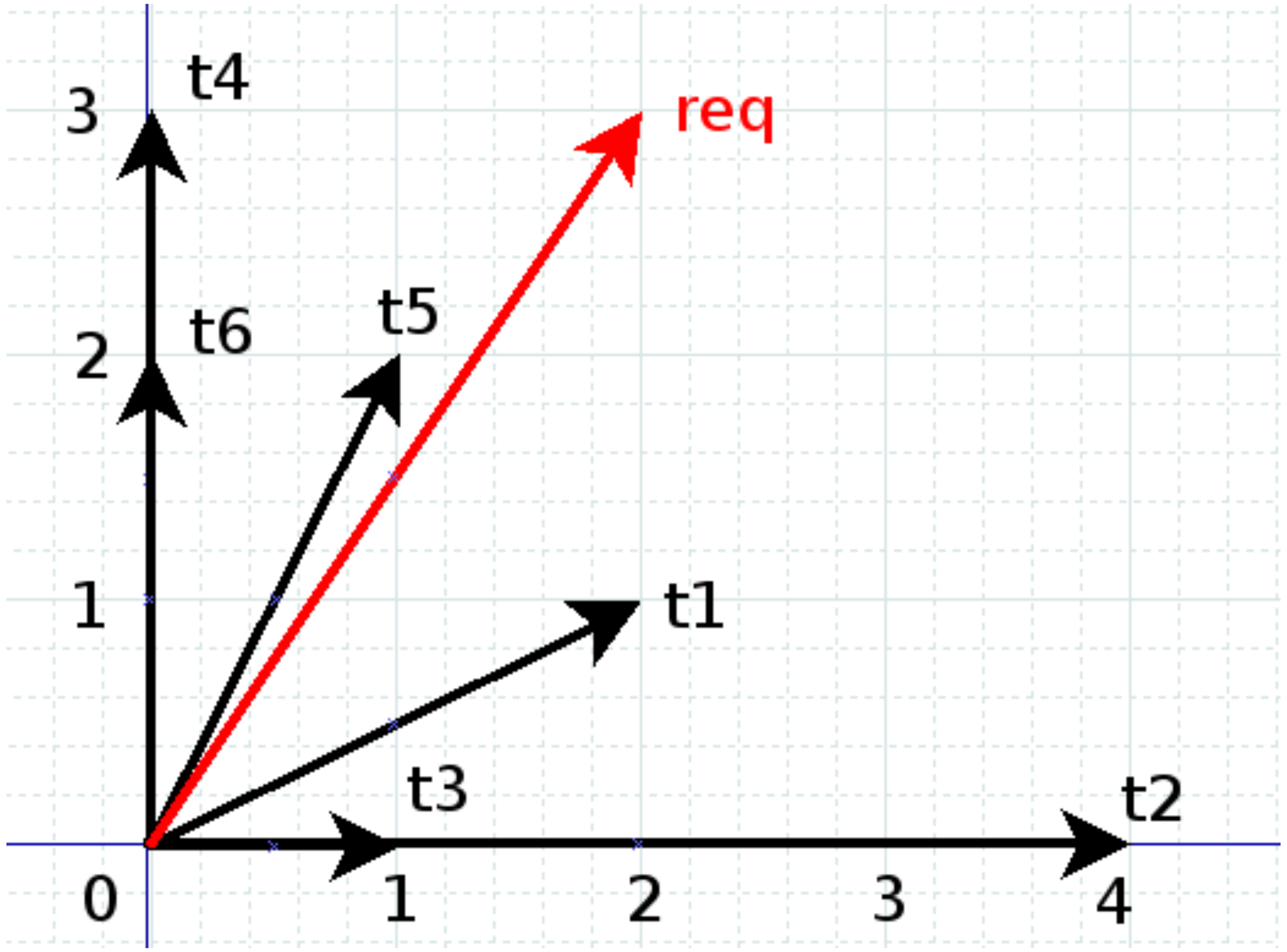


Example

Querying the database

- With the query:
  - "Pendant la crise, l'usine à rêves Hollywood critique le cynisme de l'industrie."

	r
cinéma	2
économie	3



BIP LanguageModels and KnowledgeGraphs

Example

Results of scoring using different measures

	t1	t2	t3	t4	t5	t6
cinéma	2	4	1	0	1	0
économie	1	0	0	3	2	2
Manhattan	2	4	4	2	2	3
Euclidean	2	3,6	3,16	2	1,41	2,24
dot	7	8	2	9	8	6
cosine	0,86	0,55	0,55	0,83	0,99	0,83

Log-frequency weighting

- Sometimes, especially for large documents, ***tf*** is too important
  - A document with 1000 occurrences of word w is not 10 times more relevant than a document with 100 occurrences of word w
- Solution: use log-frequency weighting

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Inverse Document Frequency

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., high, increase, line)
  - A document containing such a term is more likely to be relevant than a document that doesn't - but it's not a sure indicator of relevance.
- df_t is the document frequency of word t: the number of documents that contain t ($df_t \leq N$)
- We define the idf (inverse document frequency) of t by

$$\text{idf}_t = \log_{10} (N/\text{df}_t)$$

- N: size of document collection

Tf.idf

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

- Plusieurs variations (avec, sans log, type de normalisation…)
- Schéma de pondération le plus commun en RI

Term frequency		Document frequency		Normalization	
n (natural)	$\text{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\text{tf}_{t,d})$	t (idf)	$\log \frac{N}{\text{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \text{tf}_{t,d}}{\max_t(\text{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\text{df}_t}{\text{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\text{tf}_{t,d})}{1+\log(\text{ave}_{t \in d}(\text{tf}_{t,d}))}$				

Summary

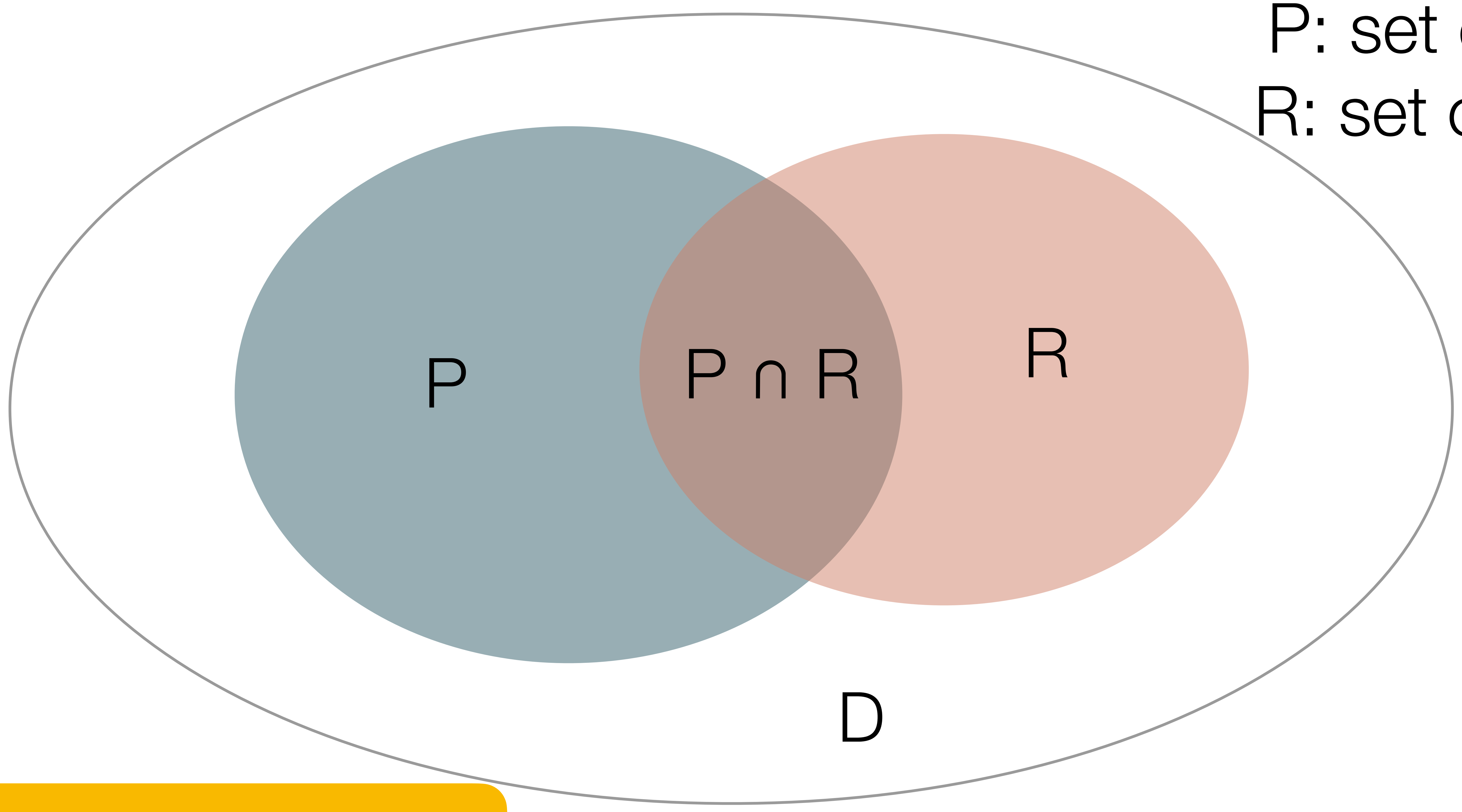
- Represent the query as a weighted tf.idf vector
- Represent each document as a weighted tf.idf vector
- Compute the cosine similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top K documents (e.g., K = 10) to the user

Measuring IR effectiveness

- To measure ad hoc information retrieval effectiveness in the standard way, we need a test collection consisting of three things:
- 1. A document collection
- 2. A test suite of information needs, expressible as queries
- 3. A set of relevance judgments, standardly a binary assessment of either relevant or non-relevant for each query-document pair.

Précision/Rappel

P: set of relevant documents
R: set of retrieved documents



Precision: P ∩ R / R

Recall: P ∩ R / P

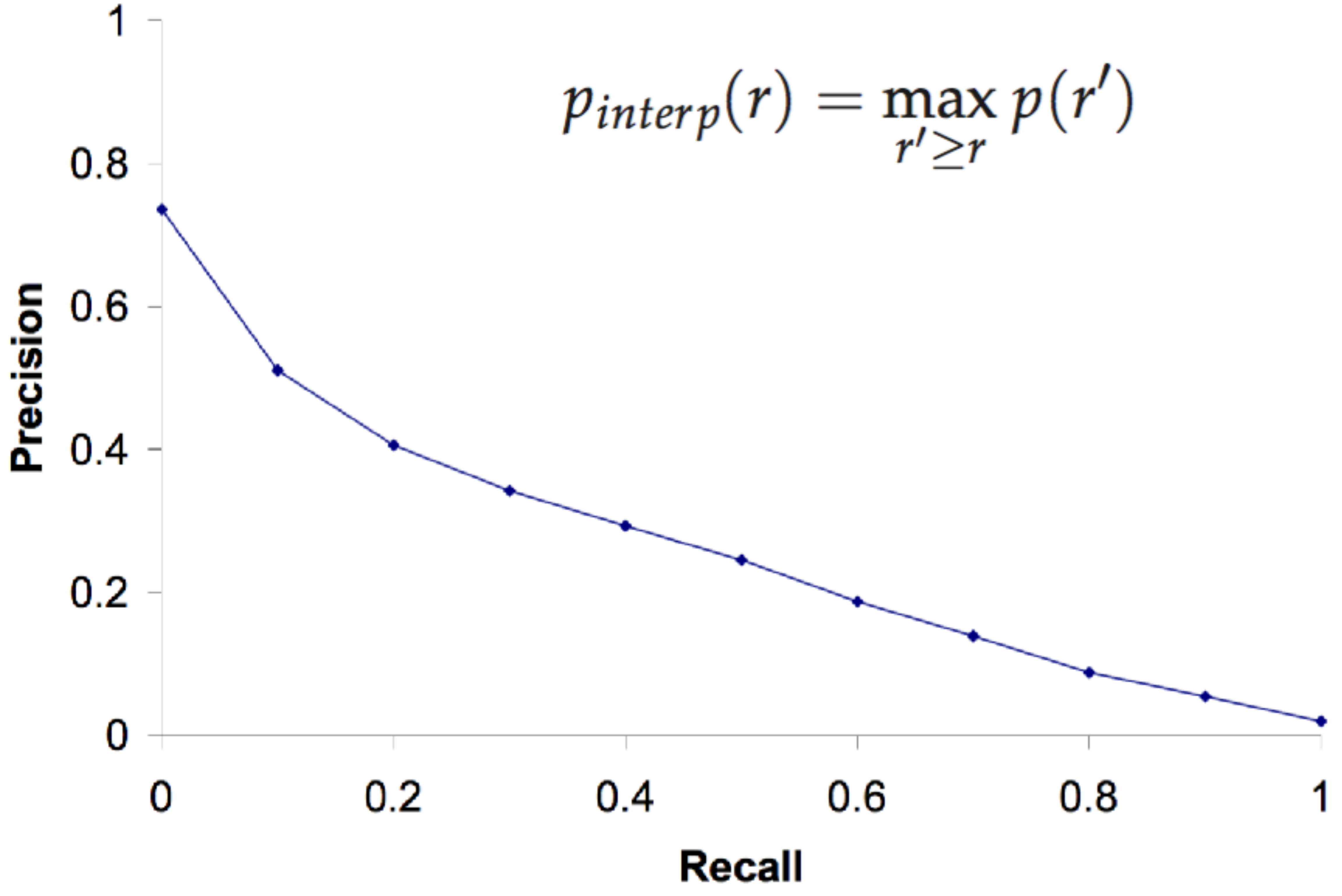
How to evaluate ranked lists?

Evaluating ranked lists

- **P@N**: evaluate precision after a certain number of results (N)
  - Simple to calculate, effective to measure user interest in the top k results
  - Problem: fails to connect to recall
- **R-Precision**: calculate precision after R (number of relevant documents in the collection) results
- **MAP**: Mean Average Precision
  - Calculate precision every time that we find a relevant result in the list, then do the average

Interpolated precision/recall graph

- Corrélation entre précision et rappel
  - 11 points de rappel
    - 0 < r < 1
- Idée: si la proportion de documents p
  mesure que j'analyse la liste, je peux a



Example

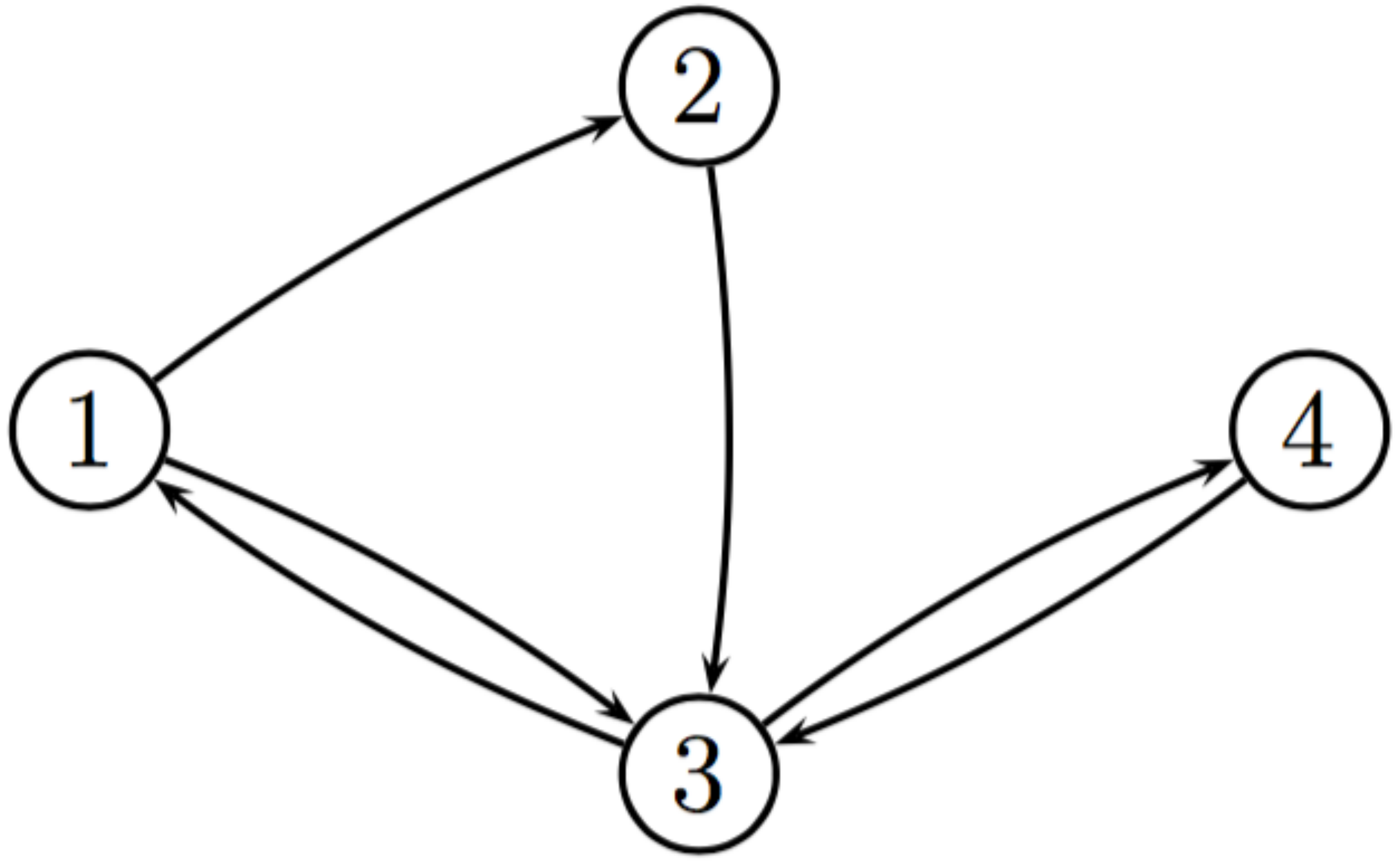
- List of retrieved documents (R: relevant, N: not relevant):
  - R R N R N R N N N R N N R N N R N N R N
  - Total number of relevant documents in the collection: 13
- P@5: 0,6
- P@10: 0,5
- R-Precision: 0,46
- MAP: (1 + 1 + 0,75 + 0,67 + 0,5 + 0,46 + 0,44 + 0,42)/8 = 0,655

PageRank

- Google “core” algorithm
  - Many more features on top of it in the actual Google search engine
- Every web page is a graph node and the links to other pages make the edges of the graph
- The objective is to weigh the pages depending on the connections of the page

Example:

- 4 pages



PageRank

- Every link between A and B is a “vote” of A for B
- The score (weight) of a page depends on the inbound links
  - But also: the weight of the pages that are “voting”
  - ->recursive function to weigh the nodes

PageRank Function

$$S(V_i) = \frac{1-d}{|V|} + d * \sum_{j \in In(V_i)} \frac{1}{|Out(V_j)|} S(V_j)$$

- **Out(V_j)** : Out-degree of node ***V_j***
- **In(V_i)** : In-degree of node ***V_i***
- **d**: smoothing factor (usually 0,85)
- **V** : number of nodes (documents)

PageRank example

- Example with 4 pages:

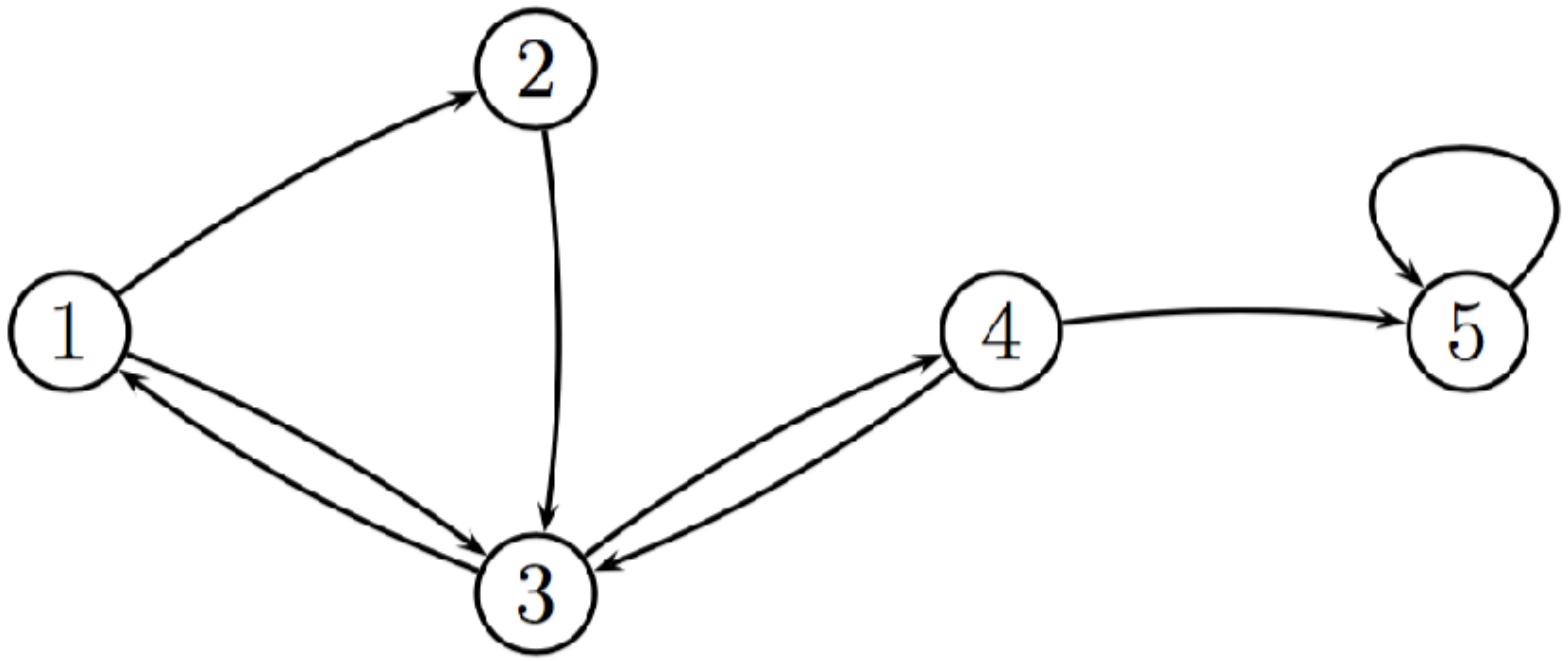
$$\begin{cases} c_1 = \frac{1}{2}c_3 \\ c_2 = \frac{1}{2}c_1 \\ c_3 = \frac{1}{2}c_1 + c_2 + c_4 \\ c_4 = \frac{1}{2}c_3 \end{cases}$$

	nœud 1	nœud 2	nœud 3	nœud 4
$t = 0$	0	1	0	0
$t = 1$	0	0	1	0
$t = 2$	$\frac{1}{2} = 0,5$	0	0	$\frac{1}{2} = 0,5$
$t = 3$	0	$\frac{1}{4} = 0,25$	$\frac{3}{4} = 0,75$	0
...	...	...	...	...
$t = 10$	0,228...	0,105...	0,437...	0,228...
...	...	...	...	...
$t = 20$	0,222...	0,111...	0,444...	0,222...

- 20 iterations with an initial configuration 0 1 0 0
- Without taking into account **d**

Smoothing factor motivation

- We need to “jump out” of loops:



- We can consider **1-d** as the probability of choosing another page without following the links (“teleporting” probability)

Deep Learning + Sparse Retrieval

Limits of Sparse Models

- Main problem: **similar or even the same concepts can be expressed in different ways**
  - Affects recall principally (if I can find a result where the same words are used, OK!)
- This has been a similar problem in language modelling
  - (see the limits of n-gram models)
- Idea: apply Neural Language Models to IR!

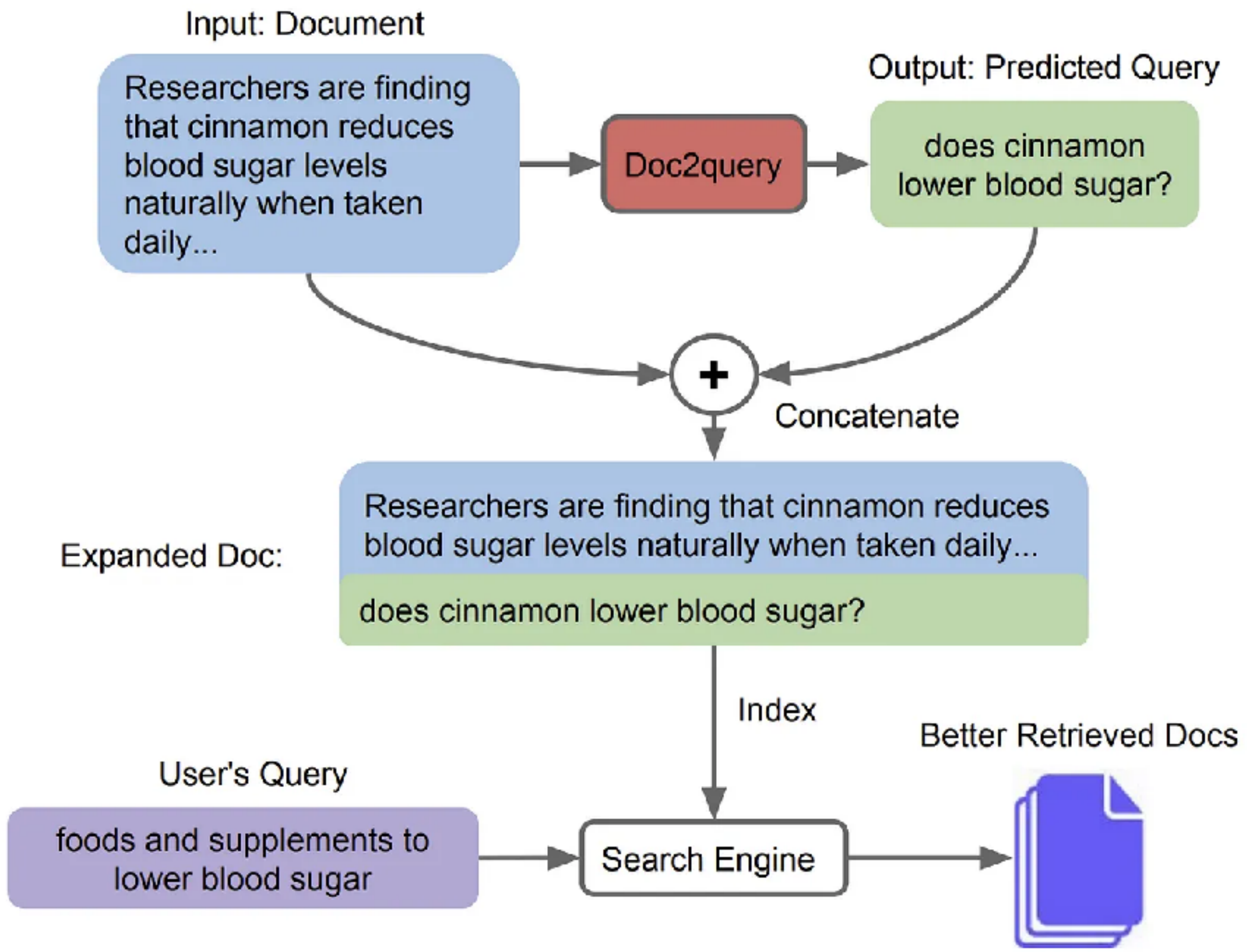
First Attempts

- Bag-of-Embeddings approach:
  - Intuitively, replace words with their embeddings (word2vec, GloVe…)
  - Bad idea because words may have different meanings while these embeddings are unique
- Solution: exploit contextual embeddings
  - DeepCT, HDCT : learn to map BERT's contextualized text representations to context-aware term weights
    - So the weight terms come from BERT and are stored in a "classic" index

Predicting Queries from Documents

- Doc2Query is another method that uses deep learning (seq2seq models)
- Idea: train a seq2seq model on known IR collections to see what users would ask about a document
- Store the hypothetical queries with the documents (indexed in the classical way)

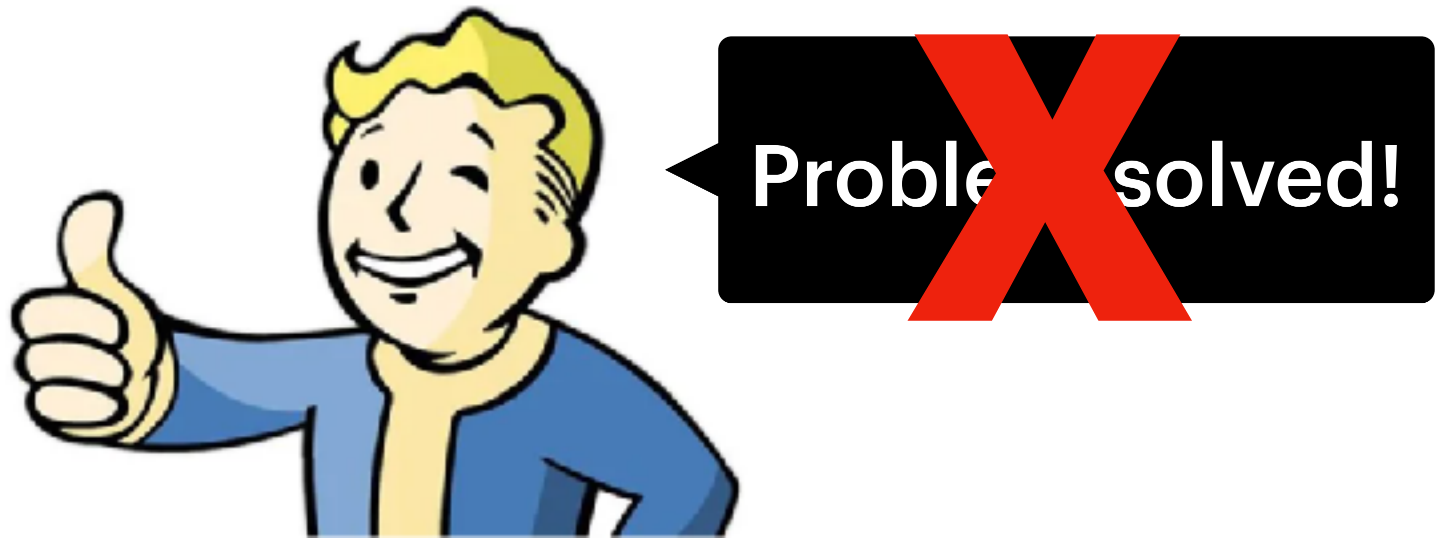
These solutions do not solve the main problem of Sparse Retrieval

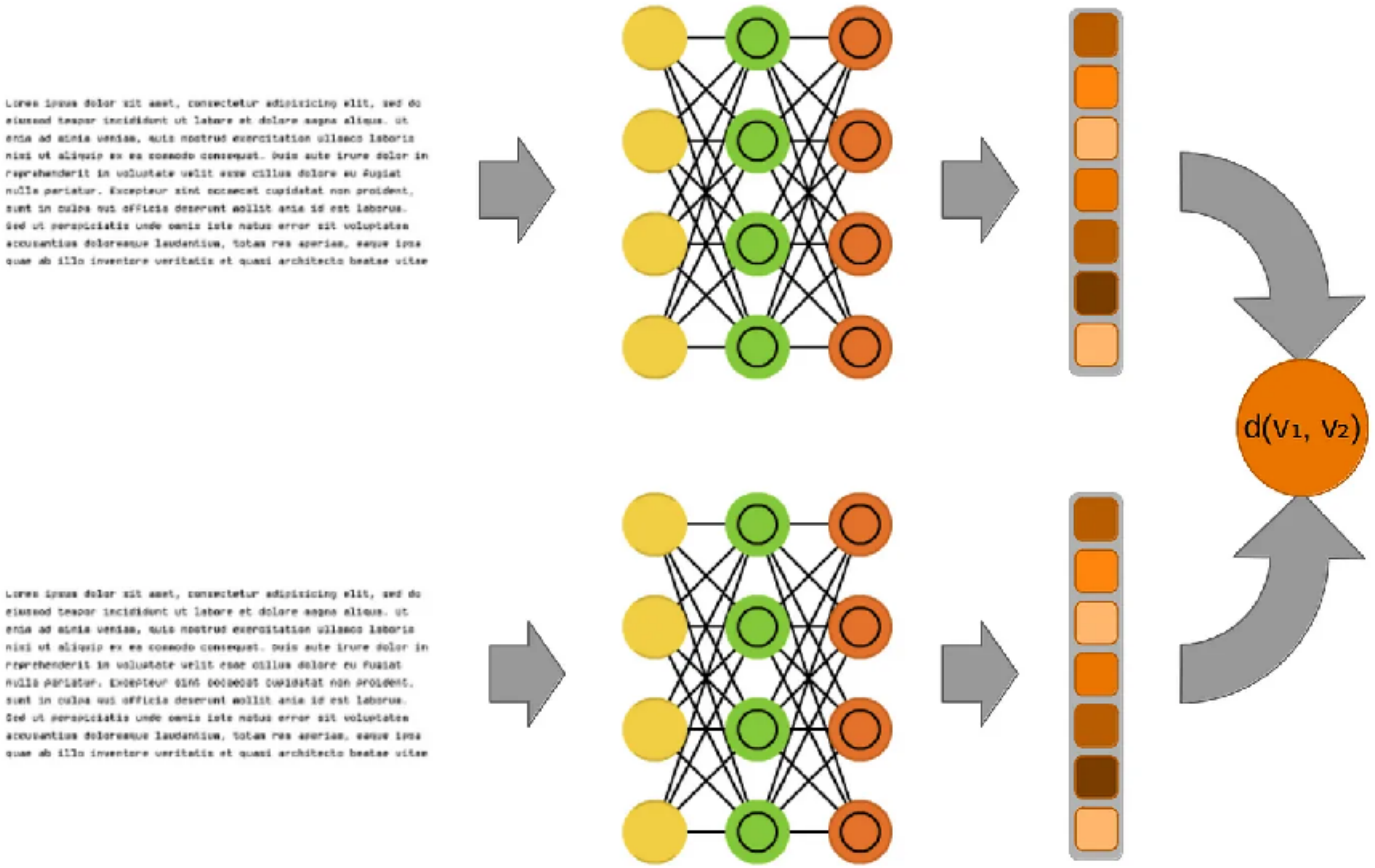


Dense Retrieval

Dense Retrieval Paradigm

- Use LLMs to encode both queries and documents into dense vectors
- Retrieval is then just finding the document vectors that have the highest similarity to the query vector





BIP LanguageModels and KnowledgeGraphs

Dense Retrieval Issues

- Averaging Word2Vec embeddings scores better than using BERT to encode the sentences:

Model	STS12	STS13	STS14	STS15	STS16	STSb
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50

Table 3: BERT embedding similarity performances on STS tasks [10]

- Anisotropy of high dimensional embeddings

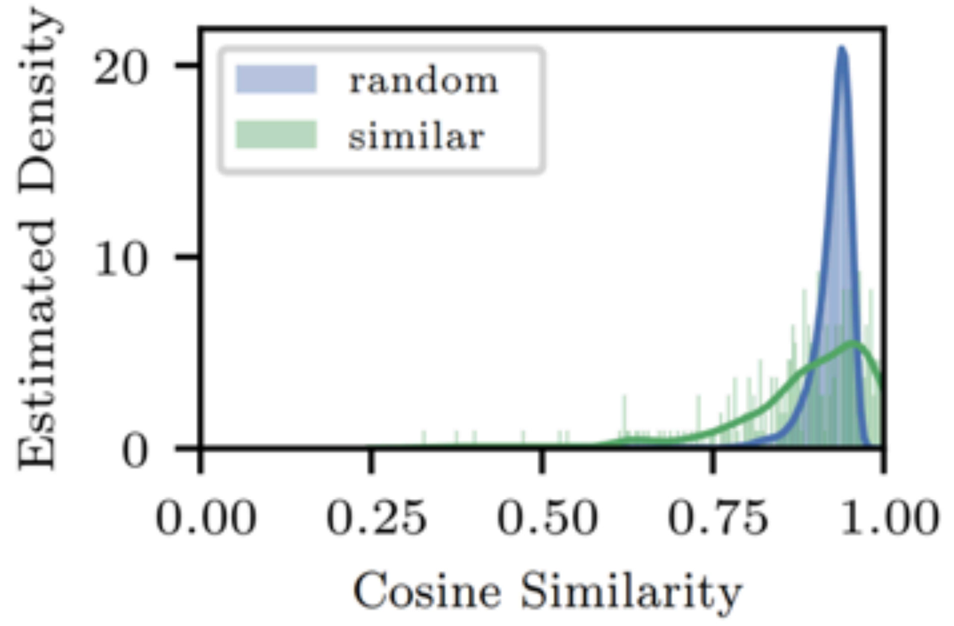


Figure 11: Similarity of RoBERTa $\overrightarrow{[\mathrm{CLS}]}$ on semantically similar and random pairs from STS-S [11]

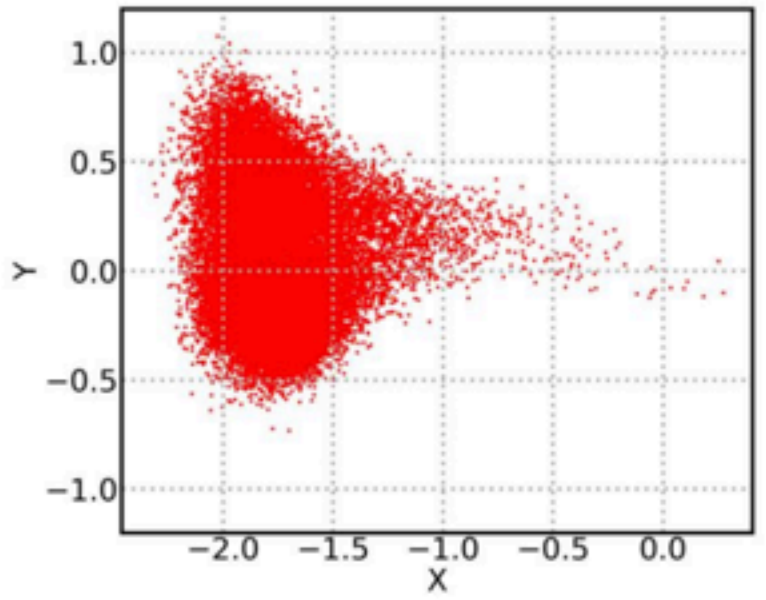
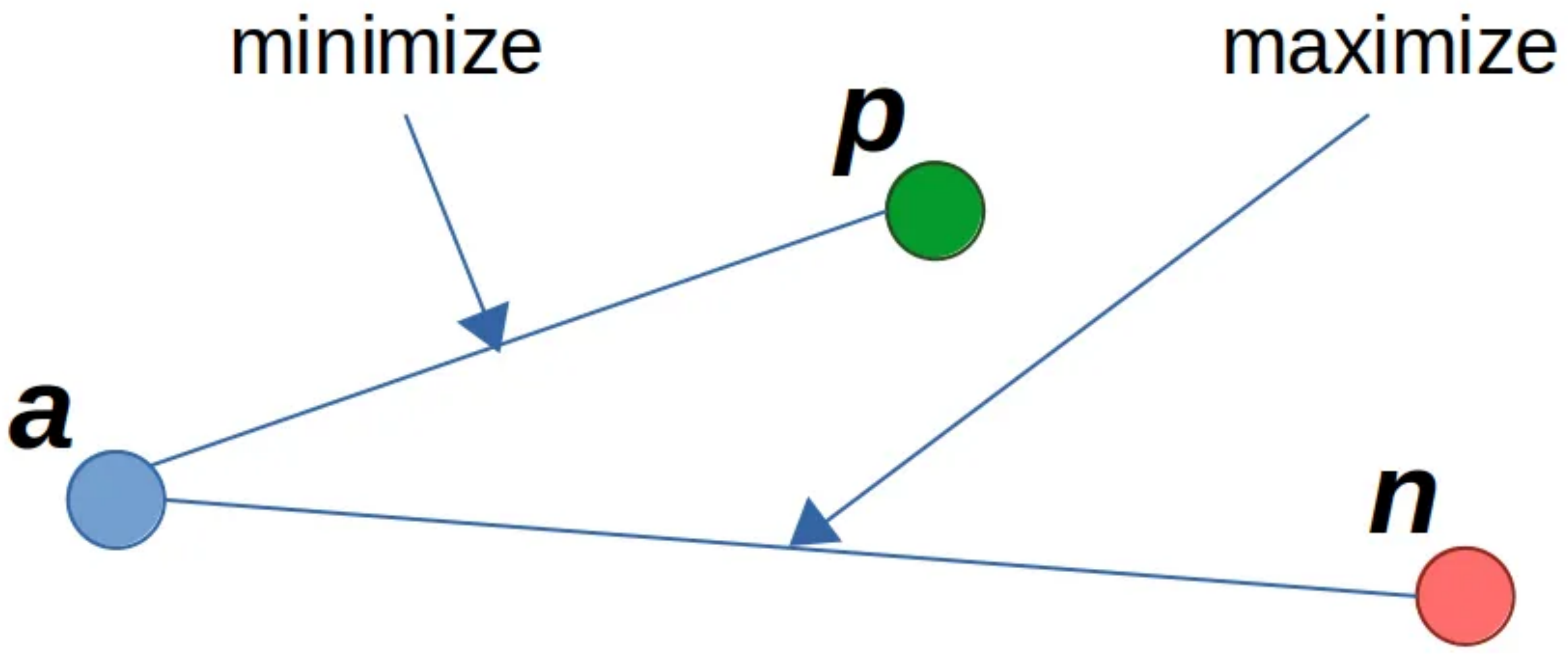


Figure 12: SVD 2-D mapping of word embeddings from Transformer trained on EN→DE [12]

Contrastive Learning

- Idea: train embeddings such as embeddings of similar sentences/ paragraphs are closer and embeddings of dissimilar sentences/ paragraphs are pushed farther away

$$Loss = max(d(a,p) - d(a,n) + \lambda, 0)$$

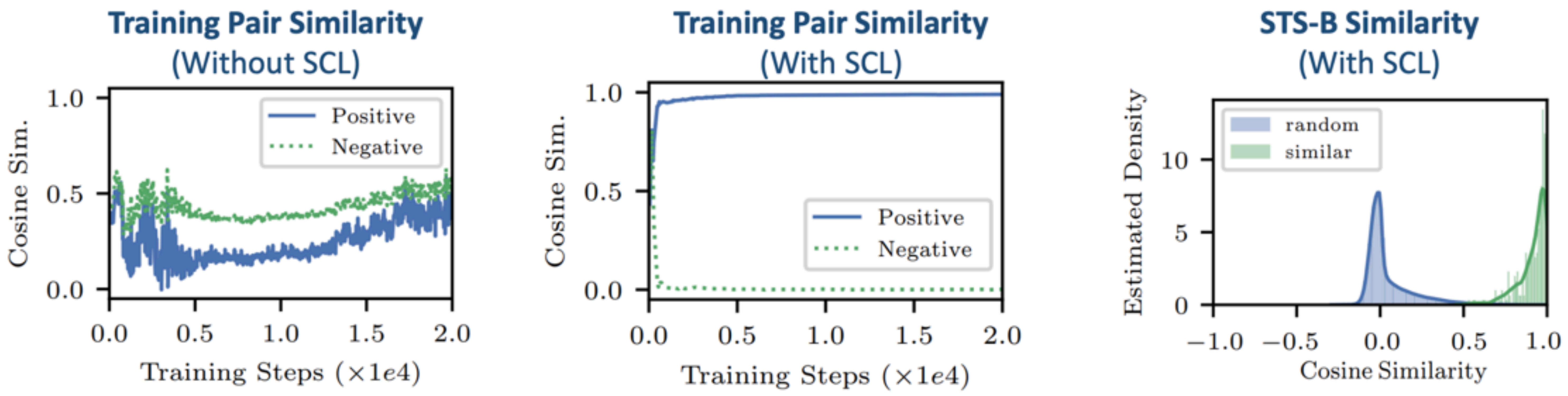


Example : S-BERT (Sentence - BERT) https://sbert.net/

BIP LanguageModels and KnowledgeGraphs

Contrastive Learning

- Contrastive Learning has huge beneficial effects on document similarity
- Non-random results in retrieval



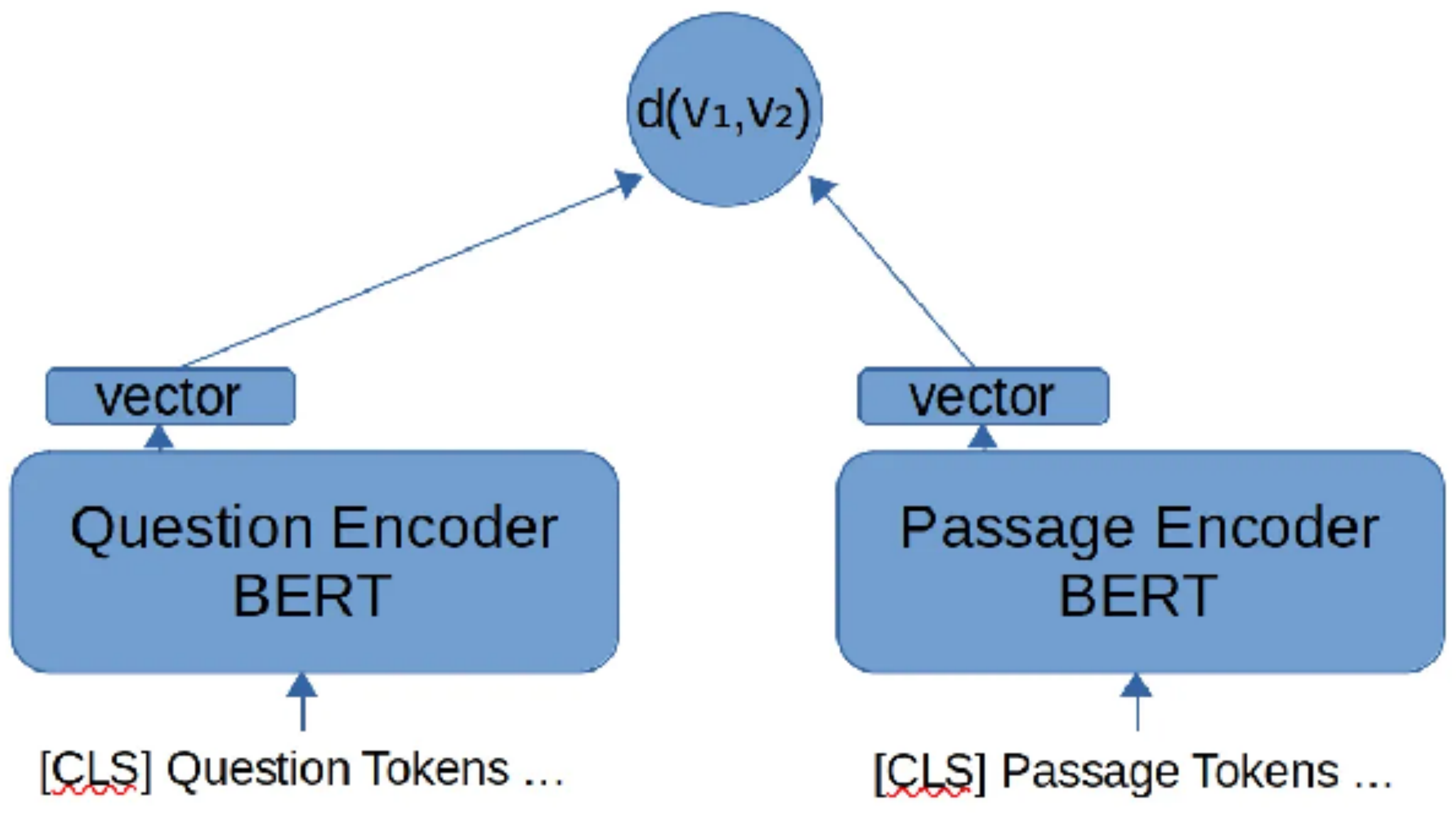
Failed without SCL
(Although 90% overlap!)

Easy-to-Learn Task
(90% overlap, after all)

Effective Calibration
& Good Zero-Shot Ability

Dense Passage Retrieval (DPR)

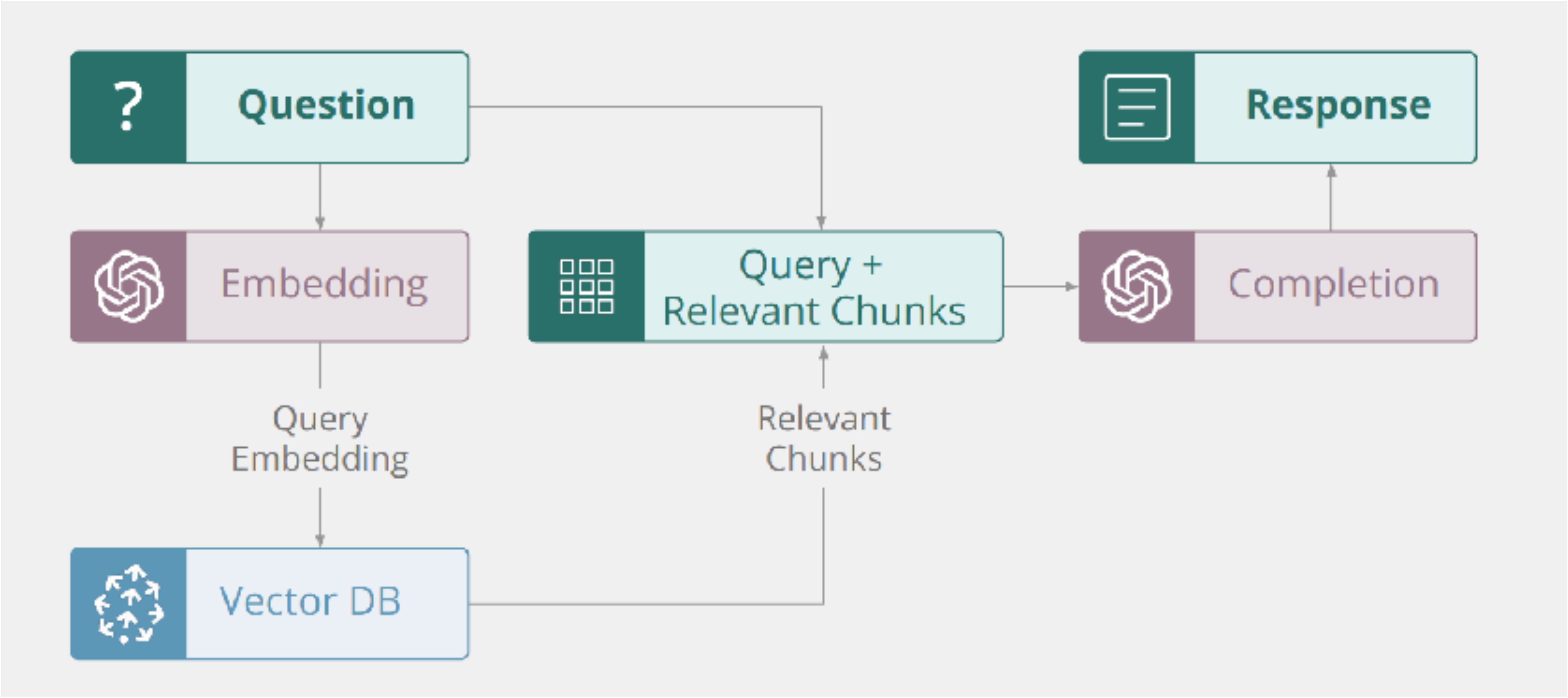
- Idea: split documents into passages, encode passages with S-BERT or similar
- Index passages with an efficient vector store such as FAISS (https://github.com/facebookresearch/faiss)
- Problems:
  - sometimes the relevant information is distributed over more passages
  - similarity sometimes is high even if the main entities are missing (style over substance)



Retrieval Augmented Generation (RAG)

RAG principle

- Data Ingestion: creating an index (dense) using embedded representations -> Vector DB or Vector Store
- Querying: a LLM is used either to build/help building the user query and/or to summarise the results



Example: HyDE

https://python.langchain.com/v0.1/docs/templates/hyde/

- A LLM is used to generate a document that respond to the query
- The vectorized document is compared to the content of the index
- The background idea is that answers look more similar between them than queries and answers

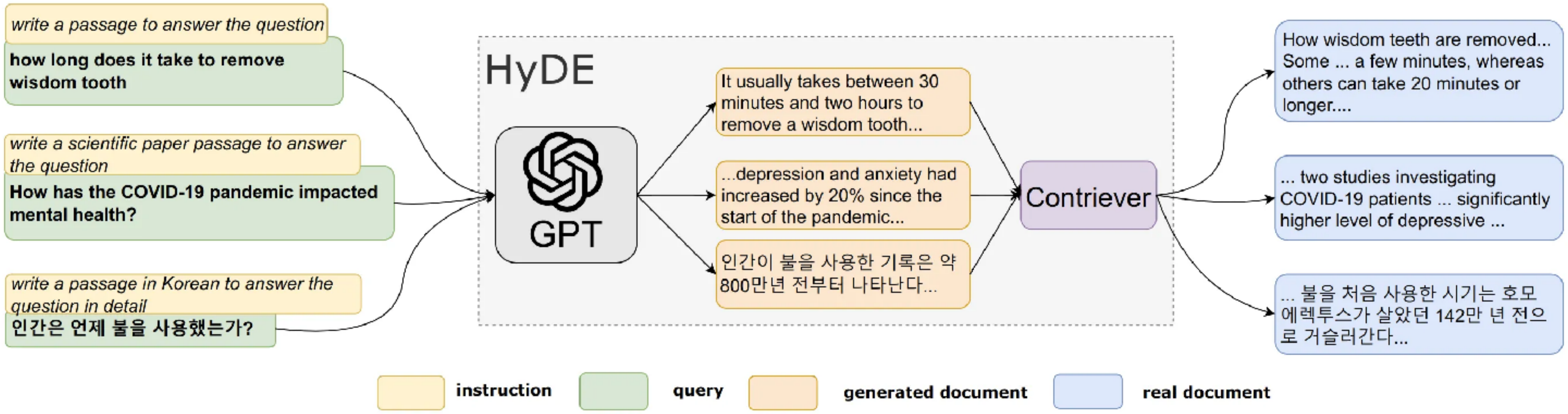


Figure 1: An illustration of the HyDE model. Documents snippets are shown. HyDE serves all types of queries without changing the underlying GPT-3 and Contriever/mContriever models.

Example: LLamaIndex

https://www.llamaindex.ai/

Data Ingestion
(LlamaHub 🦙)

- Connect your existing data sources and data formats (API's, PDF's, docs, SQL, etc.)

Data Structures

- Store and index your data for different use cases. Integrate with different db's.

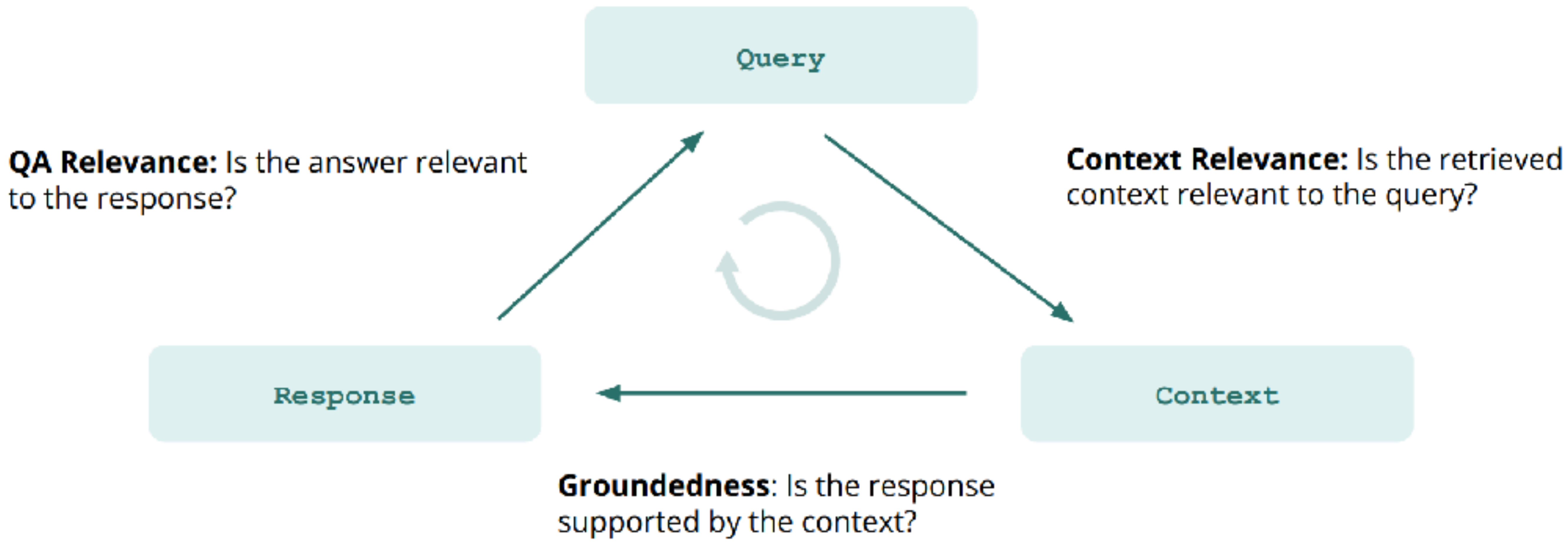
Retrieval and Query Interface

- Given an input prompt, retrieve relevant context and synthesize a knowledge-augmented output.

Evaluating RAG

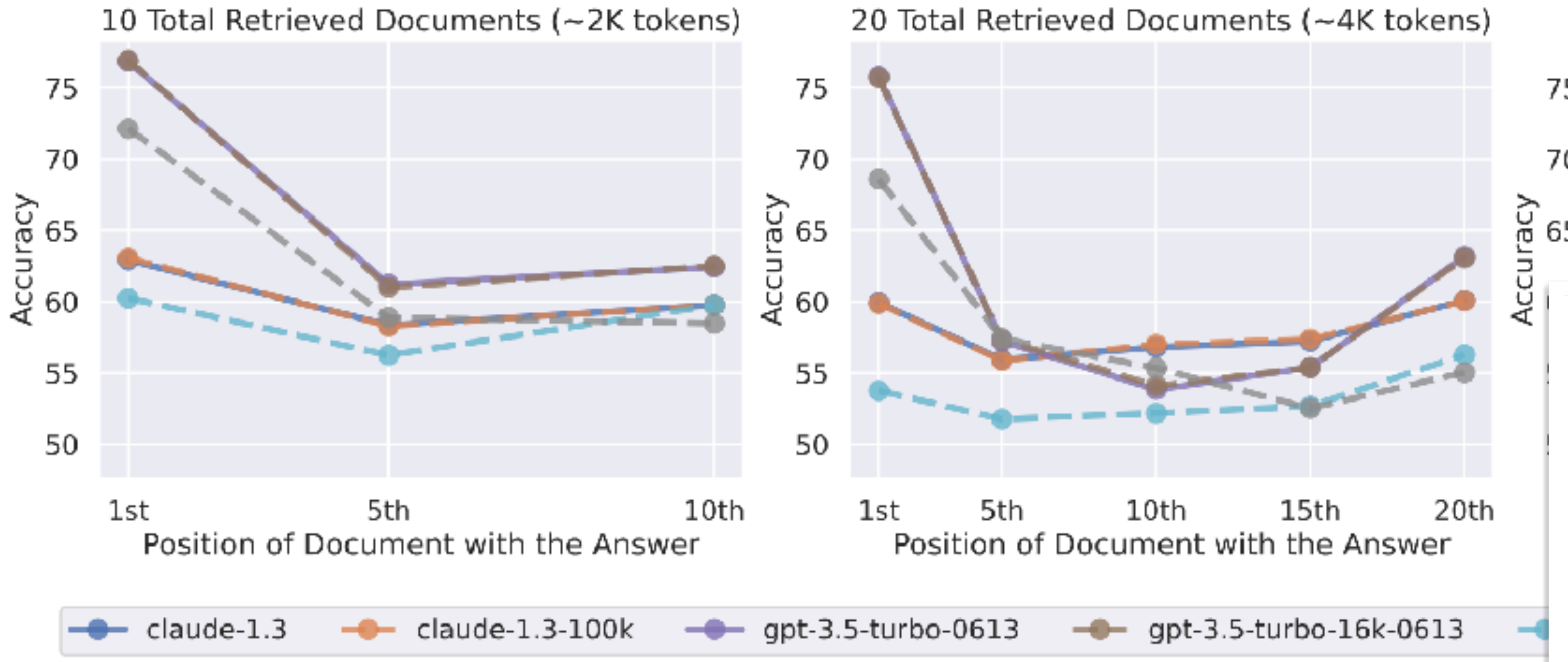
- Makes it difficult to evaluate in the “standard” IR way (no rankings)
- Risk of hallucinations (content that is not in the results)
- Can we use LLMs as rankers?
  - That is, can LLMs evaluate relevance in a reliable way?

The RAG Triad

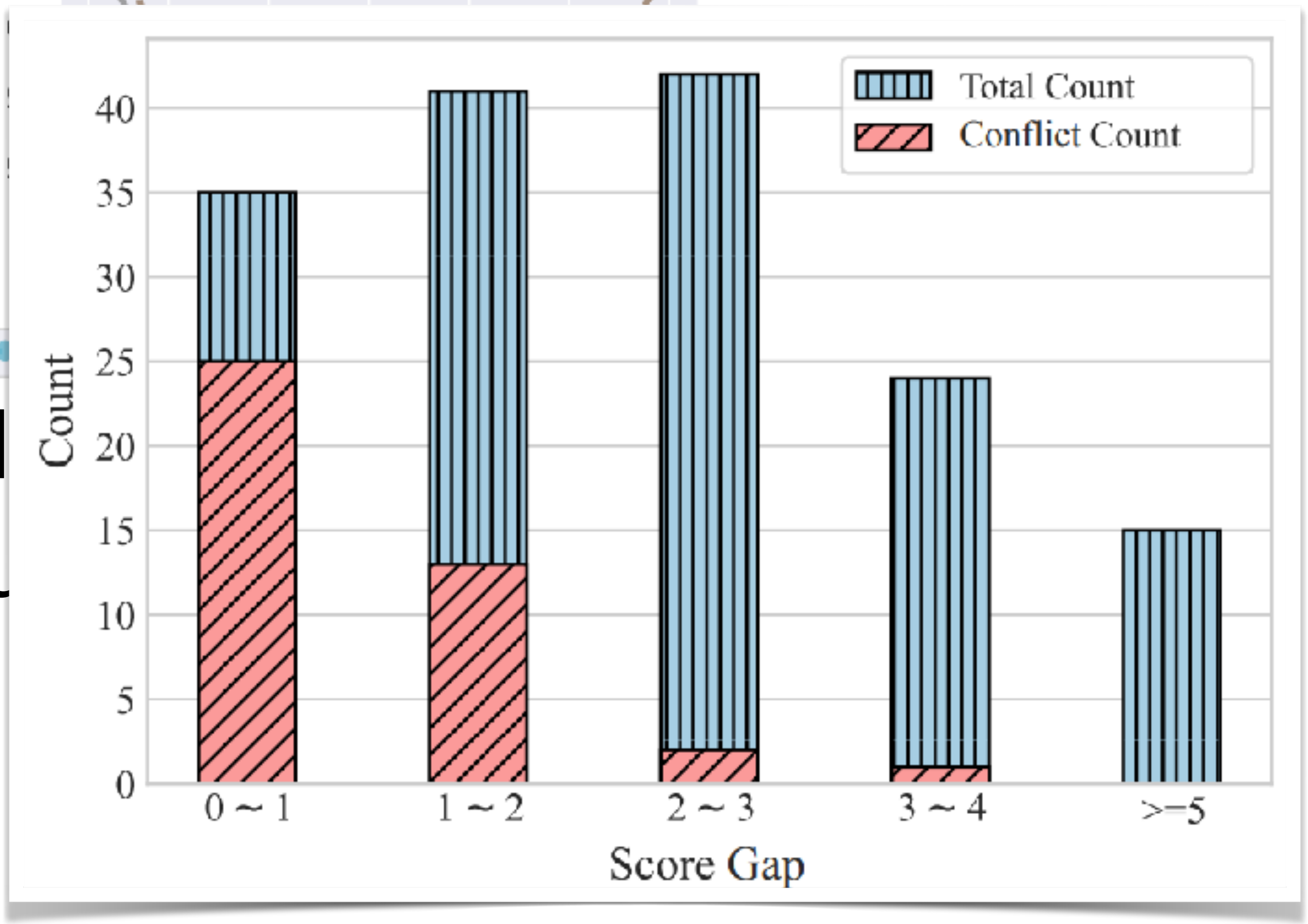


Positional Bias

- RAGs tend to have a positional bias towards the first positions of answers



- Also: swapping the slots of two responses and
most likely produce conflicting evaluation resu

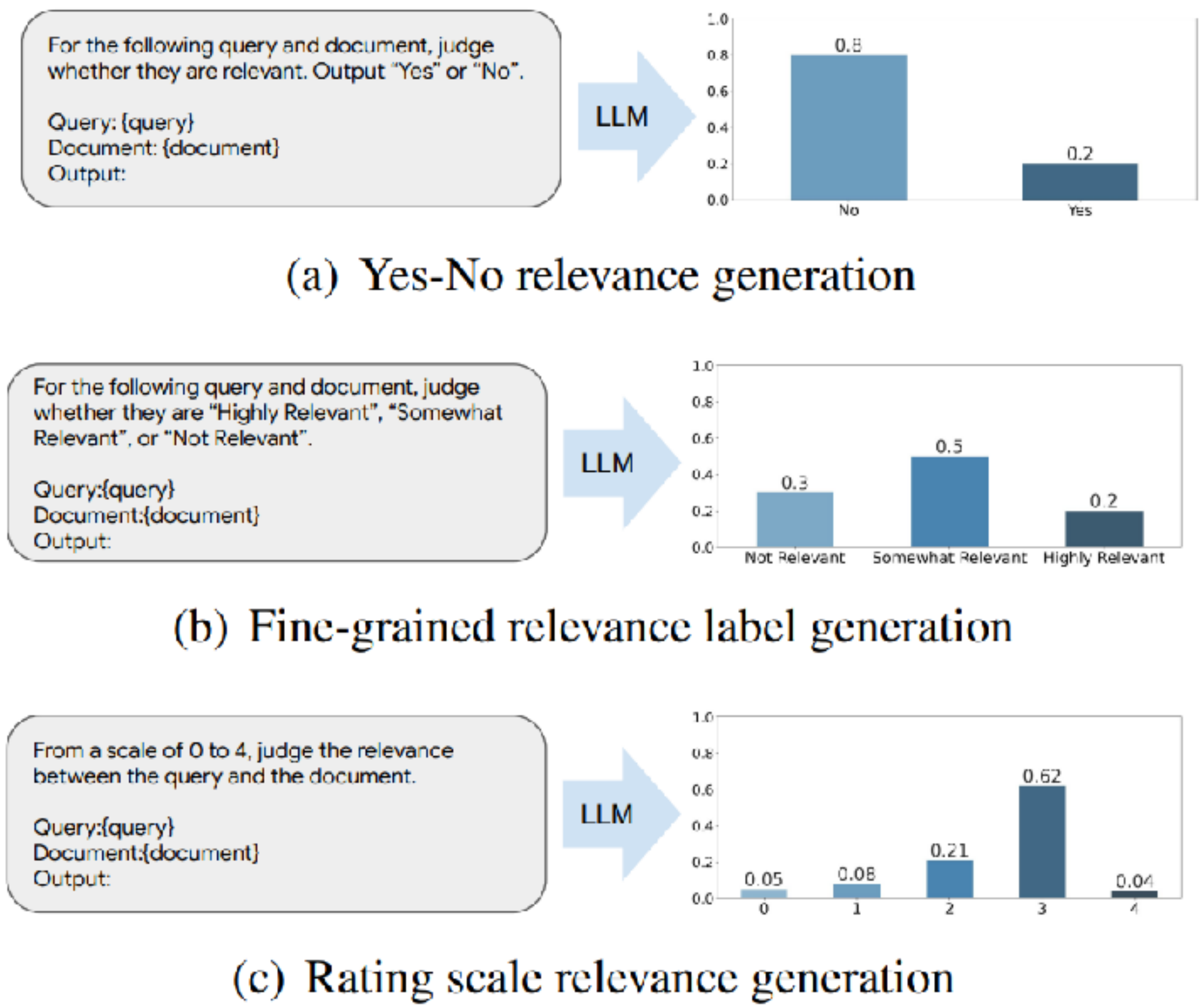


Potential Causes of Sensitivity to Order

- Model Architecture/Positional Embeddings
  - These may affect where the models pay more attention
- Model size
  - Larger models have a larger context window and seem to “lose focus” in the middle of large documents
- Human bias
  - Supervised models such as chatGPT may suffer from bias of human evaluators that limit to the first proposed solutions

Solutions to Positional Bias

- Fine-Tuning of models on datasets for ranking (e.g. MSMARCO)
  - RankLLama, RankingGPT
- Fine-grained relevance assessments:
- Query Decomposition
  - Having more aspects to judge relevance



(a) Yes-No relevance generation

(b) Fine-grained relevance label generation

(c) Rating scale relevance generation

Conclusions

- LLMs are not trustworthy for retrieval and ranking
  - Risk of hallucinations
  - Sensitivity to position of information in documents
- They may add some insights but classic IR is still a “safer” option in critical scenarios
- However RAGs are currently among the most promising industrial applications of LLMs
  - Health, Education, Finance…

Thanks !