



CS 839 Next Generation Data Systems

Lecture 6: Advanced Retrieval

Tianyu Li
litianyu@cs.wisc.edu

Administrivia

- Project matchings have been sent
- Project Proposals due 10/8
 - TODO: discuss requirements

Recap: Vector DB

- Use embeddings to represent queries and documents as vectors, and retrieve relevant vectors using similarity/distance
- Vector DB productionizes and scales this paradigm to work under tight latency, throughput, and cost constraints
- What remains to be solved?

Motivating Example

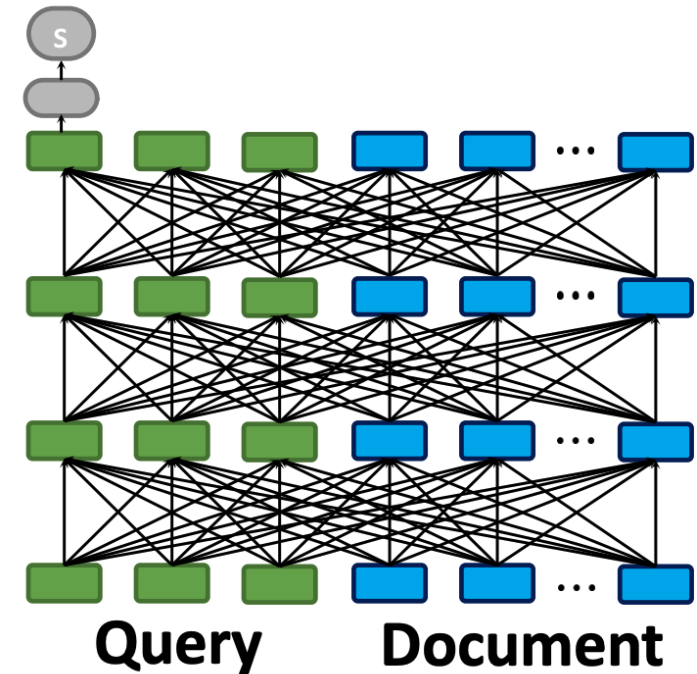
- User Question: “After watering my houseplants, water collects in the trays underneath the pots. Can I collect that water and use it to water the plants again?”
- Candidate Paragraphs (similarity from qwen3-embedding-8B)
 - “...These containers supply plants with water from a reservoir, making watering more convenient...” (0.5457)
 - “...recycling water can supplement existing water supplies and reduce demand for fresh water...” (0.5279)
 - “...Dissolved salts can accumulate in soil, damaging its structure and harming plant roots...” (0.3474)

Similarity vs. Relevance

- Relevance: does the passage help address the user's need?
 - Highly dependent on the question and domain
- Paradigm underlying most vector DB: bi-encoders
 - i.e., query and documents are separated embedded and compared
 - Representation must preserve information for many possible questions before seeing any of them under a tight representation budget
- Bigger/better models still operate under these architectural constraints and are unlikely to magically overcome them

Cross-Encoders

- Idea: evaluate query and document together
- For simplicity: think of it as embedding query + document and scoring it directly, allowing for query-dependent view of the document
- Problem: expensive, and limited opportunities for pre-computation
 - Typically used for re-ranking
 - Also, usually must be fine-tuned for the task

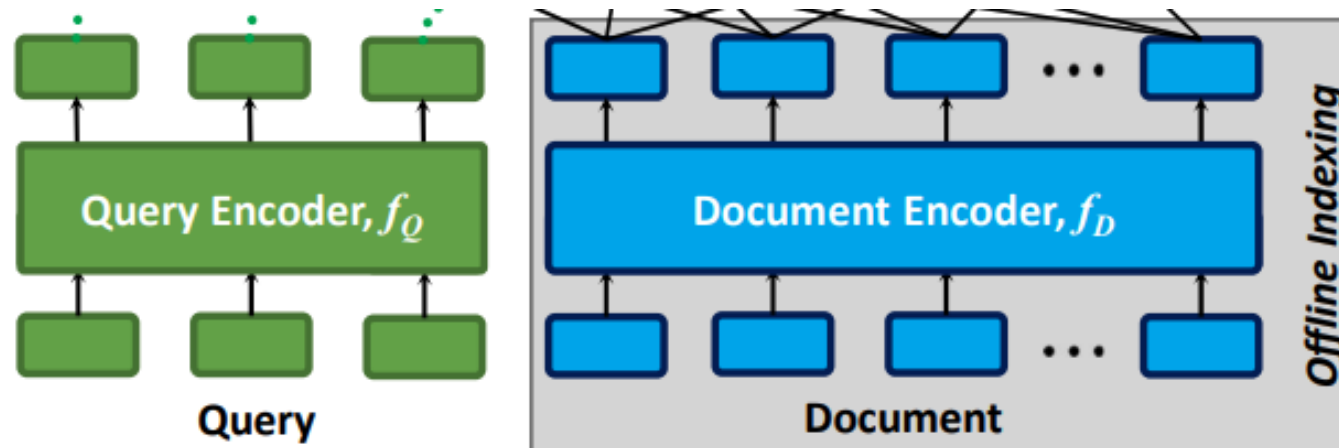


Contextualized Late Interaction (ColBERT)

- Goal: keep precomputation but still allow for fine-grained interactions between query and document
- Idea: pre-compute documents, but delay query-document interaction until after encoding. Use lightweight interaction at query time.

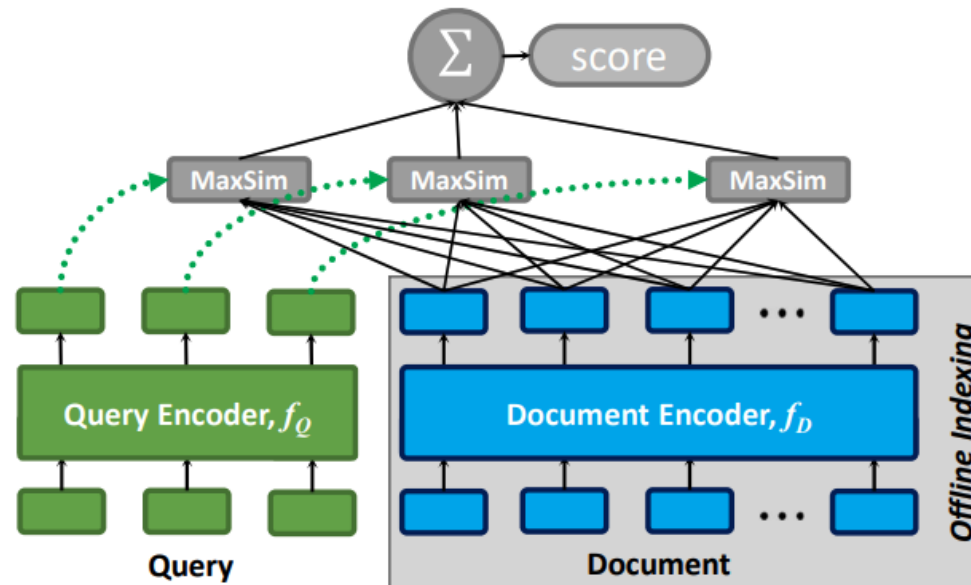
ColBERT

- Encode query & document separately with a shared encoder and retain contextualized per token embedding
 - Document encodings are pre-computed



ColBERT

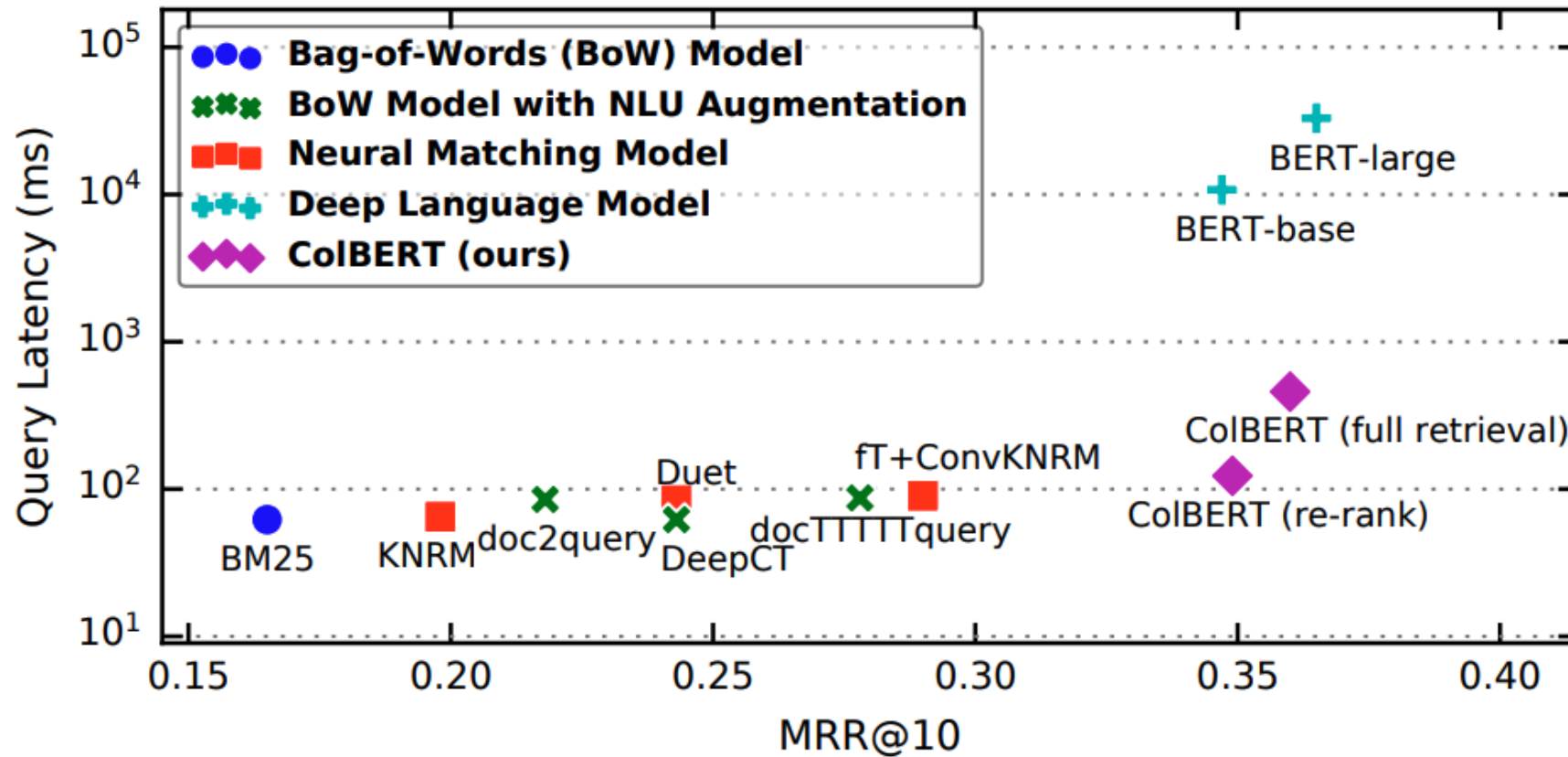
- At query time, compare every query token vector with every document token vector
 - For each query vector, take max-sim match
 - Sum to obtain the final score



ColBERT – Usage

- First idea: use as a cheap re-ranker
 - Fine, but would not help if earlier stages already eliminates gold document due to limitations of traditional dense retrieval
- End-to-end retrieval using ColBERT is possible
 - Idea: build a vector index on all document tokens
 - For each query vector, issue ANN search returning top-k candidates
 - Union and deduplicate resulting document IDs
 - Fetch full representation and evaluate

ColBERT – Results



A word on fine-tuning

- Note: ColBERT modifies the BERT model for retrieval to work
 - Explicit [Q]/[D] markers to distinguish input
 - Learned Projection/Normalization of per-token embedding to save space
 - Short queries padded with [MASK] tokens to serve as additional matching vectors
 - Discard punctuation on documents
- ColBERT is fine-tuned on the task
 - Backpropagate through maxsim on (query, relevant document, irrelevant document) triples

ColBERT v2

- One key challenge with ColBERT: space efficiency
 - One document goes from one vector to dozens or even hundreds
- ColBERT v2 reduces index footprint by 6 – 10x while improving retrieval quality
 - Residual compression represents each token vector as a centroid + quantized residual
 - Better training technique distilling from a true cross-encoder

Multi-vector Retrieval

- ColBERT represents a family of techniques that represent each searchable object using a collection of vectors
 - Relevance is scored through interaction between two vector sets
 - Active area of research!
- Active Area of Research!
 - Can we pick vectors not based on token-level embeddings?
 - How to reduce space/storage overhead?
 - How to improve search performance and index efficiency?

More Vectors == Better?

- Dense retrieval is often weak at handling precise constraints
 - E.g., “Horror novels published in 1947” will have high semantic similarity to a horror novel set in the 1940s published in 2026
- Traditional approaches often complement dense retrieval
 - E.g., keyword search, BM25, etc.
 - Often also called “sparse retrieval”

Classical IR

- Same interface and setup as vector retrieval, but without dense embeddings
 - This often means relying on signals from keyword matches
- Strawman: let's just count occurrence of every word per document and sum for every queries term!

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
against	0	0	0	1	0	0	3	2	3	0
age	0	0	0	1	0	3	1	0	4	0
agent	0	0	0	0	0	0	0	0	0	0
ages	0	0	0	0	0	2	0	0	0	0
ago	0	0	0	2	0	0	0	0	3	0
agree	0	1	0	0	0	0	0	0	0	0
ahead	0	0	0	1	0	0	0	0	0	0
ain't	0	0	0	0	0	0	0	0	0	0
air	0	0	0	0	0	0	0	0	0	0
aka	0	0	0	1	0	0	0	0	0	0

Classical IR

- Obviously, some adjustments are needed
 - E.g. throw away “the”, concatenate hyphens, etc.
- Frequency is the primary signal but normalization is also very important
 - Rarer term matches are much more impactful
 - Document size matters as longer document tend to have more words and therefore higher frequency for matched words, etc.

TF-IDF

- Term Frequency, Inverse Document Frequency

$$TF(term, doc) = \log(1 + Freq(term, doc))$$

$$IDF(term) = \log \frac{N}{df(term)}$$

$$TF.IDF(term, doc) = TF(term, doc) \times IDF(term)$$

BM25

- Tweaked version of TF-IDF that also penalizes longer documents
- Very widely used and efficiently implementable with an inverted index (i.e., term lookup table)

$$IDF(term) = \log\left(1 + \frac{N - df(term) + 0.5}{df(term) + 0.5}\right)$$

$$TF(term, doc) = \frac{Freq(term, doc) \times (k + 1)}{Freq(term, doc) + k \times (1 - b + b \times \frac{|doc|}{avgdoclen})}$$

Hybrid Search

- Idea: retrieve candidates with multiple methods (e.g., BM25 + vector) and combine their lists
- Question: how do we merge results from scores that are not from the same scale?
 - Idea: use reciprocal rank fusion (RRF)
 - $RRF(d) = \frac{1}{c + \text{rank}_{BM25}(d)} + \frac{1}{c + \text{rank}_{dense}(d)}$ for 2 retrievers (c is a tunable constant)
- Why not use a cascading pipeline like before?
 - The two search methods are complementary

SPLADE

- Do we need to abandon vocabulary-based representations to learn semantic matching?
 - It's fast, cheap, interpretable
- Can we use embeddings to improve classical retrieval rather than replacing it/complementing it in hybrid search

SPLADE

- Idea: Use a language model to produce a sparse vector that represents weighted keywords on the document, including words that may never occur in the document
 - E.g., a document that keeps talking about physicians, hospitals, patients, etc. should be indexed on “doctor” even if it never appears in the text
- Details are beyond the scope of this class, but SPLADE essentially modifies BERT for this purpose
 - Take the original masked token prediction head to obtain logits of possible words at each location and pool to obtain representation
 - Fine-tuned on retrieval examples

Crazy Idea

- Goal of a retrieval task: given a natural language task, return document ID
- Can we learn this task end-to-end?
 - i.e., learn to predict $P(\text{docID} \mid \text{query})$
- Would this even work?

Differentiable Search Index

- It's a form of learned index: model predicts $P(\text{docID} \mid \text{query})$
 - Trained models specialize for a corpus
- Idea: Start from BERT embeddings of each document
 - Cluster recursively to obtain a hierarchical ID (e.g., 3-2-0)
- Then, start from a foundational model (paper uses T5), learn to generate IDs autoregressively
 - Details are beyond the scope of this class
 - A document's retrieval score is probability of generating its ID

DSI – Summary

- As you might expect, has remaining scaling issues, interpretability issues, maintainability issues after follow-up work
- Nevertheless, a landmark attempt at generative retrieval
 - Models don't just produce scores and embeddings
- Next class, we will discuss agentic search as the more practical and modern spiritual successor

Discussion

- Will the future be about increasingly sophisticated unified retrieval mechanisms, or should we expect multiple techniques to co-exist? What roles will systems play?
- Which problems in retrieval disappear with better models / new retrieval techniques, and which problems will remain fundamental?
- As retrieval systems increasingly serve AI agents, would metrics about retrieval systems performance shift? How?