Information retrieval: dense retrieval

SciFact, and the same judgements as last week

LECTURE 20 LECTURE NOTES

Applications of Machine Learning

BSc Mathematics of Artificial Intelligence · Third year

Where we left off

The problem	rank 5,183 SciFact abstracts for each of 300 claims. A cheap first stage searches everything and is judged on recall at depth; whatever it never returns, no later stage can recover
BM25	idf, saturation and length normalisation over an inverted index, no training: **0.6611** NDCG@10
The metrics	precision and recall at a cutoff, reciprocal rank, AP, NDCG — each reads the order of a ranking differently
The judgements	a person's decision on a claim–abstract pair. SciFact's come from the abstracts each claim's source sentence cited, are binary and sparse, and anything unjudged is scored as irrelevant

AND THE FAILURE NO PARAMETER COULD FIX

One claim's only relevant abstract shares two words with it — `in` and `to` — and BM25 ranks it **4,999** of 5,183. Recall@10 is 77.8%, recall@100 88.5%: some documents are found and misplaced, the rest are never found at all.

Today

1. Vocabulary mismatch, and why an index of strings cannot solve it
2. **Bi-encoders** — encode once, retrieve by inner product; the training objective, and where its negatives go wrong
3. **Measured** against BM25, on the same judgements
4. **Combining them** — hybrid retrieval; cross-encoders, re-ranking, and the ceiling the first stage sets
5. **Design exercise** — a compliance-software company's search over its customers' policies, in nine steps, with the pitfalls and how to address them

No derivation today. The mathematics of this lecture is the dot product, and you have had it since Lecture 8.

Open Lecture 20 in Colab

The mismatch, restated as a requirement

Measured on SciFact's judged pairs: a relevant abstract is missing **40.9%** of its claim's words on average, and 28.6% of judged pairs share fewer than half.

- "heart attack" and "myocardial infarction"
- "lower numbers" and "reduced inoculum"
- A mechanism described rather than named

> **WHAT IS NEEDED**
>
> A representation in which **two texts that mean the same thing are close**, whether or not they share a word. Which is precisely what Lectures 17 and 18 built, and why Part V comes after Part IV.

The three repairs that do not work

Repair	Why it falls short
Stemming	merges word forms, not concepts — nothing links *infarction* to *attack*
A synonym dictionary	someone must write it, per domain, and it cannot cover paraphrase
Query expansion	adds terms from the top results, so it needs the answer to already be near the top — exactly what fails here

Each is a patch on the same assumption: that relevance is visible in shared strings. The assumption is what has to go.

This is the shape of every move from feature engineering to learned representations in this course. The patches are not stupid; they are bounded, and the bound is the assumption.

From tokens to one vector

Lecture 18 left us with one vector per token. A retrieval score needs one vector per *text*, so they must be pooled:

Pooling	What it does
[CLS]	take the reserved first token's vector — only meaningful if the model was trained to put something there
Mean	average the token vectors, masking padding — robust, and what our model uses
Max	component-wise maximum — rarely better, and discards how often a thing was said

✗ Masking the padding is not optional. Mean-pooling *over* the padding tokens makes a text's vector depend on the batch it was encoded in — the same abstract gets two different vectors, and nothing raises an error.

Why the vectors are normalised

Normalise to unit length and the inner product becomes the cosine, which throws away vector *magnitude*. That is deliberate.

- Magnitude in these encoders varies with length, not relevance — under mean pooling, longer abstracts get shorter vectors, so an unnormalised dot product quietly prefers short documents: a length bias of the kind BM25's b was invented to correct
- Normalised scores are bounded in $[-1, 1]$, which makes a threshold transferable between corpora

The same trap as Lecture 19's length normalisation, arriving from a different direction. Here the model does something about it: all-MiniLM-L6-v2 ends in a normalisation layer, so its vectors come out unit length whether or not you ask.

Encode both sides into one space

$$\text{score}(q, d) \;=\; E(q) \cdot E(d), \qquad E : \text{text} \to \mathbb{R}^{384}$$

One encoder, applied separately to the query and to the document. Vectors normalised, so the inner product is a cosine.

> **WHY "SEPARATELY" IS THE WHOLE DESIGN**
>
> $E(d)$ does not depend on the query. So **every document can be encoded once, offline, and stored**; at request time you encode one short query and take 5,183 inner products. The expensive part happens before anyone asks.

Hold on to that sentence. It is also the reason the bi-encoder is the *weaker* of the two neural models in this lecture.

One encoder or two?

	Shared encoder	Two encoders
Parameters	one set	two sets
Suits	symmetric tasks — sentence similarity	asymmetric — short query, long document
Risk	a short query and a long abstract are not the same kind of text	twice the parameters on the same labels

We use a shared encoder today, because the model we borrow was trained that way. It is a real modelling choice and it should be reported — a paper that says “bi-encoder” without saying which has told you less than it thinks.

The two architectures, side by side

B Bi-encoder

$E(q)$ and $E(d)$ computed apart, then a dot product. $E(d)$ is query-independent, so it can be stored. The query never touches the document's tokens.

X Cross-encoder

one pass over q and d together. Every query token attends to every document token. Nothing can be stored, because nothing exists until the query arrives.

The accuracy difference and the cost difference have the **same cause**: whether the query is present when the document is read.

Building the dense index

```
D = encode(model, documents)      # once, offline, 5,183 x 384
D = D / np.linalg.norm(D, axis=1, keepdims=True)

def search(query, k=10):
    q = encode(model, [query])[0]
    q /= np.linalg.norm(q)
    s = D @ q                     # every document, exactly
    return np.argsort(-s)[:k]
```

Six lines, and it is a complete retrieval system. Note what is *not* here: no index structure, no library, no database — the matrix product *is* the search, and on this corpus it is fast enough to be exact.

Everything a vector database adds is a way of not doing that matrix product in full, and it is bought with recall.

The dimension is a dial too

384 dimensions is a choice the model made for us. Its consequences are arithmetic:

Bytes per document	1,536
This corpus	8.0 MB
The same corpus at 768 dimensions	15.9 MB
Ten million documents at 384-d	15.4 GB

Every one of those bytes is read on every exact query. This is the sense in which a dense index is expensive: not to build, but to *use*. The inverted index skips every document that shares no term with the query; the exact dense scan cannot skip anything.

Training it: what the objective must say

We want $E(q) \cdot E(d^+)$ large and $E(q) \cdot E(d^-)$ small. Written as a softmax over one positive and a set of negatives:

$$\mathcal{L} = -\log \frac{\exp(E(q) \cdot E(d^+)/\tau)}{\exp(E(q) \cdot E(d^+)/\tau) + \sum_{d^-} \exp(E(q) \cdot E(d^-)/\tau}$$

Which is the cross-entropy of Lecture 17 with the documents as classes, and the contrastive objective you will meet again in Lecture 23 with images on one side. τ is a temperature: it scales the cosines before the softmax.

The positives come from the judgements. **Where do the negatives come from?** There are millions of documents and you cannot score them all, so a few are chosen — and the obvious way to choose them is where training goes wrong.

The false-negative problem, sized

A random abstract is a trivially easy negative. So **hard negatives** are mined instead: take a first stage's top results and call the unjudged ones negative. On this corpus:

Claim–abstract pairs in the grid	1.55 million
Pairs judged relevant	339
Relevant per claim, mean	1.13

BM25's top 10 for a claim contains, on average, **0.86 judged-relevant abstracts** and 9.14 others. Those others are *unjudged*, not *judged irrelevant* — and on a corpus this sparsely judged, some of them are relevant.

✗ Train on them as negatives and the gradient pushes the model away from correct answers. This is the single commonest reason a fine-tuned retriever underperforms the paper it copied.

What the bi-encoder gave up

A single 384-dimensional vector must stand for a 225-token abstract, and it is computed **before the query is known**.

- An abstract about three things compresses to one point, so it is somewhat about each and exactly about none
- Nothing in the encoder can emphasise the part of the document the query is asking about — it has not been asked yet
- Exact term matching is not guaranteed: two different gene names may land close together

✗ That is the price of indexability, and it is the whole reason a second, non-indexable model appears later in this lecture. Notice the trade is *structural*: it is not fixed by a bigger encoder.

Adding one document

	Inverted index	Dense index
New document	append to its terms' postings — microseconds	one encoder pass, then append the vector
New model version	nothing to do	✗ re-encode the entire corpus
Mixed versions	impossible to have	✗ silently wrong — vectors from two models are not comparable

✗ The last row is a real outage, and a quiet one: nothing errors, the scores are simply meaningless for the documents encoded by the old model. A dense index needs a model version stored beside it, and that is an operational cost the accuracy table never shows.

The protocol, stated before the numbers

- Same corpus: 5,183 abstracts. Same 300 test claims. Same judgements
- The dense model is `all-MiniLM-L6-v2`, trained on general web sentence pairs and never shown a scientific claim, used **zero shot**: no fine-tuning on SciFact
- Same metric code — the functions written in Lecture 19's notebook, unchanged
- Both retrievers scored **exactly**: every document for every query, no approximate index in either
- The metric was named before the run: **NDCG@10**, with recall@100 reported alongside because the first stage is judged on it

> **WHY THIS SLIDE EXISTS**
>
> Naming the metric after seeing the scores is the commonest dishonesty in this literature, and this lecture is about to produce a result where the temptation is real.

The result

Metric	BM25	Dense, zero shot
NDCG@10	**0.6611 ✓**	**✗ 0.6451**
Precision@1	**0.5433 ✓**	**✗ 0.5033**
MRR	**0.6350 ✓**	**✗ 0.6113**
Recall@10	0.7784	**0.7833 ✓**
Recall@100	0.8852	**0.9250 ✓**

✗ The neural model loses on the headline metric. **Thirty years of age, no training, no GPU, no vector store — and BM25 is ahead by 0.0160 NDCG@10.**

What this result does not license

- It does **not** say dense retrieval is worse in general. Fine-tuned on in-domain pairs, it beats BM25 on this corpus, and that is a well-replicated result
- It does **not** say embeddings are a bad idea. Recall@100 went up, and the mismatch case went from rank 4,999 to 8
- It *does* say that this particular substitution — off-the-shelf encoder for BM25, no labels, technical corpus — is a loss, and that is a decision many teams make without measuring

Stating the scope of a claim is not hedging. A result whose boundary you cannot draw is a result you cannot use.

But look at recall@100

Metric	BM25	Dense	Dense − BM25
NDCG@10	0.6611	0.6451	✗ **−0.0160**
Recall@10	0.7784	0.7833	+0.0049
Recall@100	0.8852	0.9250	**+0.0398** ✓

The dense retriever is *worse at ordering* and **better at finding**. It retrieves 92.5% of relevant abstracts somewhere in its first hundred, against BM25's 88.5%.

Recall from the last lecture what a first stage is judged on: not what the user sees, but what it makes possible. On *that* criterion — the criterion for the job it would actually be given — the dense retriever wins.

Do they find the same documents?

Take every relevant abstract in the test set and ask which method placed it in the top ten:

Found by	Count	Share
Both	227	67.0%
BM25 only	**30 ✓**	8.8%
Dense only	**38 ✓**	11.2%
Neither	**✗ 44**	13.0%

68 of 339 relevant abstracts are found by exactly one of the two. They are not two attempts at the same ranking — they fail on different documents.

Two methods with almost the same score and different errors is the classic signal that combining them will help. Which we will do, after the interesting case.

A mean hides the disagreement

BM25 0.6611, dense 0.6451 — a gap of 0.0160, which sounds like two systems doing almost the same thing.

The overlap table says otherwise: 227 relevant abstracts found by both, 30 by BM25 alone, 38 by the dense model alone. Per claim, they frequently disagree completely.

> **FAILURE CONDITION — A MEAN OVER QUERIES**
>
> **Two systems with the same mean can have entirely different error sets.** The mean answers “which is better on average”; it cannot answer “are these the same system”, and only the second question tells you whether fusing them is worth anything.

The claim BM25 could not find

“In mice, P. chabaudi parasites are able to proliferate faster early in infection when inoculated at lower numbers than when inoculated at high numbers.”

Rank of its one relevant abstract	
BM25	✗ 4,999
Dense, zero shot	8 ✓

From 4,999 to 8. The abstract still shares almost no words with the claim — nothing about the text changed. What changed is that the comparison is no longer between strings.

This is the case the whole lecture was built around, and it is worth being precise about what it does and does not show: it does **not** show the dense retriever is better. The result table at the start of this section showed it is not. It shows the two methods fail *independently*.

Hybrid retrieval: fuse the two rankings

Two methods, complementary errors. The scores are not on a comparable scale — BM25 returns unbounded sums, the dense model returns cosines — so fuse the **ranks** instead:

$$\mathrm{RRF}(d) = \sum_{\text{systems}} \frac{1}{k + \mathrm{rank}(d)}, \qquad k = 60$$

Reciprocal rank fusion. No training, no tuning, no calibration — and it needs nothing from the two systems but their orderings, which is why it is what people actually deploy.

The constant k damps the top ranks so one system cannot dominate on a single confident hit. 60 is conventional, and yes, that is the same kind of convention as $\log_2(i+1)$.

Hybrid, measured

Metric	BM25	Dense	Hybrid
NDCG@10	0.6611	0.6451	**0.6948** ✓
Precision@1	0.5433	0.5033	**0.5633** ✓
Recall@10	0.7784	0.7833	**0.8239** ✓
Recall@100	0.8852	0.9250	**0.9617** ✓

Better than either, on every metric, from adding two reciprocals. The gain over BM25 is **+0.0338** NDCG@10 — larger than anything else in this lecture will buy you.

And it is not mysterious: the earlier table said 68 relevant abstracts were found by exactly one system. Fusion is how you keep both.

The other neural model: cross-encoders

$$\text{score}(q, d) = \text{Transformer}\big(\,[\text{CLS}]\ q\ [\text{SEP}]\ d\,\big)$$

Concatenate the query and the document and put **both through the model together**. Every token of the query can attend to every token of the document, which is exactly what a bi-encoder forbids by construction.

	Bi-encoder	Cross-encoder
Sees	each text alone	the pair, jointly
Document vectors	precomputed	impossible — depends on the query
Cost per query	one encoding + N dot products	N transformer passes
Accuracy	lower	higher

Why the accurate model cannot go first

A full cross-encoder scan of this corpus, for these claims, is

$$5,183 \times 300 = 1,554,900 \text{ transformer passes}$$

On the *toy* corpus of this course. A web index is 10^{10} documents and the user is waiting 200 ms, so the arithmetic is not close — it is off by ten orders of magnitude.

HENCE THE TWO-STAGE PIPELINE OF LECTURE 19

A **cheap first stage** reduces millions to a hundred; an **expensive second stage** reorders that hundred. The architecture of every production search system is a direct consequence of the fact that the good model does not scale.

What the second stage costs, per query

Stage	Model passes per query
BM25	0
Bi-encoder	1 (the query)
Cross-encoder over top 10	10
Cross-encoder over top 100	100
Cross-encoder over the corpus	✗ 5,183

The last row is why the pipeline exists, and the middle rows are why its depth is the parameter you actually argue about in a design review. Both are the same arithmetic.

An accurate model that cannot be indexed

FAILURE CONDITION

The cross-encoder reads query and document together, which is exactly why it is stronger and why it cannot be precomputed. Put it first and the latency is the corpus size; the quality number says nothing about whether the system can be run.

Re-ranking, measured

Cross-encoder over BM25's top k, everything below k left where it was:

System	NDCG@10	P@1	Pairs scored
BM25 alone	0.6611	0.5433	0
+ re-rank top 10	0.6656	0.5600	3,000
+ re-rank top 50	0.6773	0.5667	15,000
+ re-rank top 100	**0.6781**	**0.5667**	**30,000**

Going from 10 to 50 buys +0.0117; going from 50 to 100 buys +0.0008 for twice the compute. **The depth is a dial with diminishing returns, and it is the main thing you tune in a deployed system.**

Reading the re-ranking curve

Depth	NDCG@10	Gain	Passes
0	0.6611	—	0
10	0.6656	+0.0046	10
50	0.6773	+0.0117	50
100	0.6781	+0.0008	100

The first ten candidates give most of the gain; the next ninety give a tenth of it, for ten times the compute. That is the shape of every re-ranking curve you will meet, and the reason production depths are usually in the tens rather than the thousands.

Note also that recall@10 barely moves: re-ranking the top 10 cannot change *which* documents are in the top 10 at all. It changes their order, and P@1 with it.

The ceiling the re-ranker cannot pass

Re-ranking reorders; it cannot retrieve. So the recall of the first stage at depth k is a hard bound on the second stage at depth k:

Depth k	10	50	100	1000
BM25 recall@k	0.7784	0.8654	0.8852	0.9650

Re-rank BM25's top 100 and a *perfect* cross-encoder still cannot exceed recall 0.8852 at any cutoff — 11.5% of relevant abstracts are simply not in the candidate set.

✗ Notice that the hybrid first stage reaches recall@100 of 0.9617. Improving the first stage raises the ceiling for everything above it; improving the second stage never does. That is the single most useful sentence in this lecture for building a system.

The whole table, in one place

System	NDCG@10	R@100	Cost per query
Corpus unranked	✗ **0.0012**	0.0117	nothing
BM25	0.6611	0.8852	postings walk
Dense, zero shot	0.6451	0.9250	1 encode + full scan
BM25 + cross-encoder@100	0.6781	0.8852	+ 100 transformer passes
Hybrid (BM25 + dense, RRF)	**0.6948**	**0.9617**	**1 encode + both**

The best NDCG@10 in this lecture comes from the row with **no cross-encoder in it** — two first-stage retrievers, used as downloaded, and a sum of reciprocals that learns nothing.

How to choose, in practice

Situation	What to do
No labels, technical vocabulary	BM25. Measure before you consider anything else
No labels, everyday language, paraphrase matters	hybrid — BM25 plus an off-the-shelf encoder, fused
Thousands of labelled pairs in your domain	fine-tune a bi-encoder; now it can beat BM25
Latency budget allows a second stage	add a cross-encoder over the top 50–100

✗ Every row starts with BM25 in the table. Not because it wins — on your corpus it may not — but because a comparison without it is not a measurement.

The notebook for this lecture

Open Lecture 20 in Colab

- **Runs on free CPU.** Encoding the corpus once is most of the work; nothing is trained
- The same BM25 code as Lecture 19, so the baseline in the comparison is the baseline you already built
- The zero-shot bi-encoder scored exactly, the overlap count, RRF, and cross-encoder re-ranking at three depths — no vector database

> **WHAT TO CHANGE**
>
> Re-rank the *hybrid* top 100 rather than BM25's. The ceiling slide predicts what should happen to recall; check whether NDCG follows.

Design exercise · in the manner of an ML system-design interview

The task: policy search inside a compliance platform

> **THE BRIEF**
>
> A compliance-software company sells a platform to banks and insurers, which load their policies into it. The new feature is a search box: an employee types “can I accept a gift from a supplier?” and gets a **ranked list of clauses** from their employer’s policies, the answer at the top. The policy says “hospitality may be declined”; the employee never uses those words. The business measures renewals, tickets removed, and the cost of acting against a policy nobody found.

- **Given:** each customer’s policies in numbered clauses, and the keyword search’s question log, a sample judged by pilot customers’ compliance officers
- **Scale:** tens of thousands of clauses per customer, hundreds of customers
- **Consumers of the output:** the search box, the compliance officers, the audit trail

✓ Nine steps follow, each with the pitfall that catches a first attempt and what to do about it. The model at the centre is today’s: a bi-encoder, fused with a lexical index and re-ranked.

Step 1 — Frame it: a ranking, two metrics, and the baseline in every table

- The output is a ranking of clauses, so the metrics are the ranking metrics: **NDCG at ten** for what the employee sees, **recall at a hundred** for the first stage, because whatever it never returns no later stage can recover
- Name both before the first run; the cut-off is the ten results the product shows
- The baseline is the keyword search the customers already have — idf, saturation, length normalisation over an inverted index, no training — on the same clauses and judgements

> **PITFALL**
>
> Naming the metric after seeing the scores. An off-the-shelf encoder that loses to the keyword search on NDCG at ten and wins on recall at a hundred invites a sales deck that quotes the second number; the customer who renews on it and sees worse results churns. State the metric first, report both, and never leave the keyword baseline out of the table.

Step 2 — Chunking: one vector per text, so decide what a text is

- The encoder produces one vector per text, computed before any question is asked. A whole policy is about everything — gifts, travel, data — so its one vector is somewhat about each and exactly about none
- Cut at the document's own joints: one passage per clause, carrying the policy's title and the section's heading, so that the text stands on its own
- The passage size is a dial: vary it with everything else fixed and read recall at a hundred on the judged questions

PITFALL — WHAT A CHUNK LOSES

A clause that says "the limit above does not apply to gifts from regulators" is unanswerable alone, and one cut mid-sentence retrieves for neither half; an employee who reads the limit without its exception acts on it, and the customer answers to its regulator. Keep the cross-reference's target in the chunk, split only at the structure, and store the clause identifier.

Step 3 — The judgements, and the split of the questions

- A judgement is a person's decision on a (question, clause) pair: the pilot customers' officers mark, per sampled question, the clauses that answer it. Binary, sparse; anything unjudged is scored as irrelevant
- Split the judged *questions*: a tuning set on which every choice is made — passage size, fusion, re-ranking depth — and a test set read once

PITFALL — UNJUDGED IS NOT IRRELEVANT

If the officers judge only what the old keyword search returned, every clause the dense model finds and the old search missed is scored as a miss: the feature is penalised for the clauses it was built to find, and cancelled on a number that measured the old system. Pool both systems' candidates before anyone judges.

Step 4 — The bi-encoder and its index, one per customer

- One encoder, applied separately to question and clause; mean pooling over the real tokens, padding masked; vectors normalised, so that a long clause is not quietly penalised
- The index is the matrix of clause vectors, encoded once, one per customer, so no search crosses a tenant. On tens of thousands of clauses the exact matrix product *is* the search: nothing bought with recall
- Start zero shot with an off-the-shelf encoder; fine-tune once thousands of judged pairs exist

> **PITFALL**
>
> The off-the-shelf encoder was trained on web sentence pairs and never saw a policy: on a technical corpus it can lose to the keyword search on the headline metric while winning on recall at depth: measure it before a date is promised. When fine-tuning, do not mine hard negatives from the unjudged top of the list: some answer the question, and the gradient pushes the model away from them.

Step 5 — Where dense fails, and the hybrid that keeps both

- Many questions name a thing exactly — a policy code, a regulation's article, a product name. Exact matching is not guaranteed in the vector space, and the keyword search gets these right
- The two systems fail on different questions, which is the signal that fusing helps: count, per judged question, which system found the answer, rather than compare two means
- Fuse the *ranks* by reciprocal rank fusion: a lexical sum and a cosine are not on a comparable scale, and ranks need no calibration

> **PITFALL**
>
> Adding the two scores, or weighting them by a constant tuned by eye. The lexical score is an unbounded sum and the dense score is bounded in a narrow band, so one system dominates whenever it is confident, which on exact identifiers is always; an officer who searches a policy by its code and misses files a ticket. Fuse ranks, and keep the overlap table in the release review.

Step 6 — Re-rank the top with the model that cannot go first

- A cross-encoder reads the question and the clause together, every token attending to every token: stronger, and nothing of it can be precomputed, because nothing exists until the question arrives
- So it runs second, over the fused top of the list; the depth is the dial, with the first tens of candidates buying most of the gain and the next hundred a tenth of it for several times the compute
- Choose the depth on the tuning questions against the latency the service agreement allows

> **PITFALL**
>
> Putting the accurate model first. A cross-encoder pass over every clause for every question is tens of thousands of transformer passes per question; the quality number says nothing about whether the system can be run within the agreement, or at what cost. And re-ranking reorders what it is given: the order of the top ten, never which clauses are candidates.

Step 7 — Evaluate on the test questions, and read the ceiling

- Same clauses, same judgements, same metric code for every system: keyword, dense, hybrid, hybrid re-ranked — one table, NDCG at ten and recall at a hundred side by side, read once
- The recall of the first stage at depth k is a hard bound on everything above it: a perfect re-ranker over a hundred candidates cannot find a clause that is not among them
- Look at the questions the pipeline still misses, per customer: a mean hides which ones

> **PITFALL**
>
> Spending the next quarter on a bigger re-ranker while the first stage misses one judged answer in ten. Improving the second stage never raises the ceiling; improving the first stage — a hybrid in place of one retriever, a better chunking, a fine-tuned encoder — raises it for everything above. Read recall at depth before the roadmap is written.

Step 8 — In production: policies change, and so does the model

- Customers edit policies daily: a new clause is one encoder pass and an appended vector, a withdrawn one a vector removed, the inverted index updated in the same step
- A new encoder re-encodes every customer's corpus: build the new index beside the old, re-run the judged questions as the regression test, release to a cohort of customers first
- Monitor per customer the questions that end without a click; the on-call engineer owns the index

PITFALL — MIXED VERSIONS

Clauses encoded by two versions of the model in one customer's index. Nothing errors; the older clauses' scores are simply meaningless, the policy amended last quarter is never found again, and an employee acts on the withdrawn rule. Store the model version beside each index, refuse to search a mismatched one, and switch once, never in place.

Step 9 — Ship it: the pipeline, and the pitfalls in one table

Each customer's policies → passages cut at clauses → judged questions pooled from both systems, split → keyword index and clause vectors per customer, one model version → ranks fused → cross-encoder over the top → NDCG at ten and recall at a hundred, read once → cohort release, rebuild on change.

Pitfall	Address it by
Metric named after the scores	both metrics named first; the keyword baseline in the table
One vector per policy	passages at the clause boundaries, size measured
Judging only the old search's results	pooling both systems' candidates before judging
An encoder that never saw a policy	measuring zero shot; no unjudged negatives when fine-tuning
Policy codes lost in vector space	hybrid retrieval, fusing ranks not scores
The accurate model first	a cross-encoder over the top, depth tuned
A bigger re-ranker under a low ceiling	reading recall at depth; improving the first stage
Two model versions in one index	version beside the index; rebuild beside, switch once

Summary

1. A lexical index matches strings; relevance is about meaning, and no patch on the index closes that gap
2. A bi-encoder can be indexed because $E(d)$ does not depend on the query — and that same independence is why it is the weaker model
3. Zero shot on this corpus, it **loses** to BM25 on NDCG@10 (0.6451 against 0.6611) and **wins** on recall@100 (0.9250 against 0.8852)
4. They fail on different documents: 68 of 339 relevant abstracts were found by exactly one, so fusing them beats both (0.6948)
5. The accurate model goes second because it does not scale; and it cannot exceed the first stage's recall at that depth
6. Designed a compliance-software company's search over its customers' policies in nine steps — both metrics named first with the keyword baseline in the table, passages cut at the clauses, judgements pooled from both systems, a normalised dense index per customer measured zero shot, ranks fused for the policy codes dense loses, a cross-encoder over the top within the service agreement, recall at depth read as the ceiling, and one model version per index — each step named for the pitfall it avoids

NOT PRESENTED — FOR AFTERWARDS

Exercises Lecture 20

five, in the style of the written paper

Exercises — Lecture 20

1 Explain why a bi-encoder can be indexed and a cross-encoder cannot, and say what the two differences have in common as a cause. [5]

2 A first stage has recall@100 of 0.885. State the highest recall a perfect re-ranker over those 100 candidates can reach, and what follows for where to spend effort. [4]

3 BM25 scores 0.6611 NDCG@10 on SciFact and a zero-shot bi-encoder 0.6451, yet fusing their rankings by reciprocal rank fusion scores 0.6948. State the measurement that predicts the gain, and why the fusion uses ranks rather than scores. [5]

Solutions on Lecture 21’s deck.

Exercises — Lecture 20 (continued)

4 Hard negatives are mined by calling the unjudged documents in a first stage's top results negative. On SciFact, BM25's top 10 holds on average 0.86 judged-relevant abstracts and 9.14 others. State the trap, and what it does to training. [5]

5 Zero shot on SciFact, a bi-encoder loses to BM25 on NDCG@10 (0.6451 against 0.6611) and wins on recall@100 (0.9250 against 0.8852). Say which job each metric judges, and which system you would put in the first stage. [4]

Solutions on Lecture 21's deck.

THE FIVE SET AT THE END OF LECTURE 19

Solutions Lecture 19

not part of this lecture

Solutions — Lecture 19

1. A ranking has relevant documents at ranks 1, 3 and 10, with three relevant documents in total. Compute AP and…

✓ AP $= \frac{1}{3}\left(1 + \frac{2}{3} + \frac{3}{10}\right) = 0.6556$**; NDCG@10** $= \frac{1+0.5+0.2891}{1+0.6309+0.5} = 0.8396$**.**

- AP averages the precision at each hit and divides by $|R|$, not by the number of hits found
- DCG discounts by $1/\log_2(i+1)$, and the ideal puts all three at ranks 1–3

2. State why $\log_2(i+1)$ is used rather than $\log_2(i)$ in DCG, and whether the choice of discount is derived…

✓ $\log_2(1) = 0$ **would divide by zero at rank 1. The discount is conventional.**

- The $+1$ is a necessity, not a preference
- The shape encodes a belief about attention, and reporting NDCG adopts it

Solutions — Lecture 19 (2)

3. BM25 has three parts. Name the failure of raw term counting each one answers, then say which part the…

✓ idf answers common words dominating; saturation answers repetition; length normalisation answers long documents. Of those removed singly idf costs most; saturation is never removed on its own.

- Removing idf alone costs 0.1261 NDCG@10 and removing length normalisation alone costs 0.0120 — those are the two the ablation isolates
- The remaining row drops saturation and length together for 0.1809, and a measurement that removes two things at once cannot attribute the loss to either of them

4. 98% of claims match no document under Boolean conjunction. State the property of the conjunction that causes…

✓ One missing term excludes the document entirely. Score rather than filter.

- The conjunction is brittle, and an empty result looks like 'no such document exists'
- Scoring makes a missing term cost something rather than everything

Solutions — Lecture 19 (3)

5. Large benchmarks choose which pairs to judge by pooling. State what happens to a document no pooled system…

✓ It is unjudged and scored as irrelevant, so a new method is penalised for finding it.

- Everything outside the pool is treated as not relevant by convention
- Gains on a mature benchmark are partly gains at matching the pool

LECTURE 21 · LECTURE NOTES

Recommender systems: from ratings to factors

No query: a user, and what they did — a ratings matrix that is mostly missing, and its long tail
The derivation: matrix factorisation, and why the SVD of Lecture 6 cannot simply be taken
Item neighbourhoods, and implicit feedback, which has no negatives

Part V · Lecture notes · examinable