Lesson 3 · Hybrid Retrieval & Reranking

PDF: this lesson · **Install (Linux · macOS · Windows):** guide

> **Part of local-ai-lab** - a hands-on course for building local AI.

> **Interactive version (slides):** https://nikolareljin.github.io/local-ai-lab/lesson-3-hybrid-retrieval-reranking.html **Read it locally (no GitHub Pages):** `./run -l 3 lesson` **Course home:** https://nikolareljin.github.io/local-ai-lab/ **Source:** https://github.com/nikolareljin/local-ai-lab **Author:** Nik Reljin **Time:** ~30-45 min · **Prerequisites:** Lesson 1 · full objectives in SYLLABUS.md

> **Lessons:** 1 · RAG → 2 · MCP → **3 · Hybrid retrieval (you are here)** → 4 · RAG safety → 5 · RAG evaluation → 6 · Repo assistant → 7 · LangChain → ... → 15 · Docs from changes

> **Status: working demo.** Runnable in **Python, Node.js, and C# / .NET** - same algorithm, three languages, identical rankings. Runs 100% offline, no embedding model required. See **From demo to production** at the end for what to harden for real use.

What you'll learn

Lesson 1 put **BM25** and **embeddings** side by side as two ways to retrieve. This lesson answers the question that always comes next: **which one should I use?** The honest answer is **both** - and this lesson shows why, and how to combine them.

By the end you'll understand:

- **BM25** - best for exact terms: IDs, error codes, names, keywords
- **Embeddings** - best for semantic similarity and paraphrasing
- **RRF** (Reciprocal Rank Fusion) - combine multiple ranked lists into one
- **Reranking** - reorder the top chunks before generation for precision
- **Query rewriting** - turn a vague question into a better retrieval query
- **Metadata filtering** - narrow by file type, product, date, page, module, system

> **The one idea:** BM25 and embeddings are not enemies. They solve **different** retrieval problems. **"What fixes error E_4096?"** needs exact-term BM25; **"why won't it turn on?"** needs semantic matching. Good enterprise RAG runs both and fuses the results.

The demo

A tiny corpus of three support docs lives in `data/`. We run **two queries** that pull in opposite directions:

- an **exact-keyword** query - `error E_4096`
- a **keyword-free paraphrase** - `broken gadget` (no document contains either word)

...and print the BM25 ranking, the semantic ranking, and the fused (RRF) ranking for each.

Run the comparison (offline, no dependencies)

From the repo root - the `demo` action prints the rankings and exits, with **nothing to install** (pure standard library). Pick any language; all three give the **same** output:

```
./run -l 3 demo                # Python - print the BM25 / semantic / hybrid comparison and exit
./run -l 3 --lang node demo    # Node.js - same output
./run -l 3 --lang csharp demo  # C# / .NET 8 - same output
./run -l 3 test                # the offline Python test
./run -l 3 lesson              # read this lesson in a browser, served locally
./run -l 3 show                # walk through this lesson's steps (code, data, prompts, commands)
```

> `./run -l 3 demo` is the dependency-free **comparison**: it prints the rankings for two queries and exits - no server, no install, byte-identical across Python, Node.js and C#.

Experiment in the GUI (needs Flask)

For a hands-on feel, bare `./run -l 3` opens an interactive **experiment GUI** over the bundled short story - **The Voyage of Caretta the Magnificent** (the magic turtle who became an astronaut), five chapters in `story/`. Type a query and watch the BM25, semantic and hybrid rankings (and the numbers behind them) recompute as you tune the knobs - no code editing.

```
./run -l 3           # opens http://127.0.0.1:<port> - the experiment GUI over the 5-chapter story
```

> The GUI is a small Flask app, so unlike the `demo` it needs one dependency. `./run` installs it into the project venv automatically on first use (`pip install -r requirements.txt` if you prefer manual).

Try these and watch the two retrievers diverge:

Query	Type	Expect
`Nuevo Edén`	exact name	BM25 nails the discovery chapter (`04-nuevo-eden.md`)
`Alpha Centauri`	exact name	the chapters that name the star system rank top
`who found the new planet`	paraphrase	the semantic arm surfaces the discovery chapter with no shared keyword
`how did they keep the turtle alive in space`	paraphrase	the spacesuit chapter (`02-spacesuit.md`) rises

Exact names are BM25's strength; the paraphrases are where the semantic arm earns its keep - and the hybrid (RRF) column gets both right. Prose like this shows the trade-off far better than the toy support-doc corpus.

Output:

```
Query: "error E_4096"
 BM25 (lexical):   ['error_codes.md']
 Semantic (stand): ['error_codes.md']
 Hybrid (RRF):     ['error_codes.md']

Query: "broken gadget"
 BM25 (lexical):   []
 Semantic (stand): ['power_issues.md']
 Hybrid (RRF):     ['power_issues.md']
```

Notice the second query: **BM25 returns nothing** - no document literally contains **broken** or **gadget** - yet the semantic arm recovers `power_issues.md`, and the hybrid keeps that hit. That is the divergence fusion exists for.

> **Why no embedding model?** To keep the demo offline and dependency-free, the "semantic" arm is a small **synonym-expanded overlap** stand-in - enough to behave like embeddings (mapping **broken** → **fail/dead**, so it recovers the **power/startup** doc that BM25 misses entirely). In production you swap in real sentence embeddings; the **fusion** code doesn't change. That swap is the first item in **From demo to production**.

Concept 1 · BM25 - lexical retrieval

BM25 ranks documents by **exact term overlap**, weighting rare terms more (IDF) and damping repeated terms and long documents. The compact implementation in the demo:

```
idf = math.log(1 + (n - df[t] + 0.5) / (df[t] + 0.5))      # rare terms count more
tf  = d["tokens"].count(t)
score += idf * (tf * (K1 + 1)) / (tf + K1 * (1 - B + B * dl / avgdl))   # tf, length-normalized
```

- **Wins** the exact query: `E_4096` appears in exactly one document, so its IDF is high and `error_codes.md` shoots to the top.
- **Loses** on a keyword-free paraphrase: ask **"broken gadget"** and BM25 returns **nothing** - no document shares a single token with the query.

> **Teaching point:** reach for BM25 when the user knows the exact string - an error code, a product SKU, a function name, a person's name. It is fast, needs no model, and is unbeatable at literal matching.

Concept 2 · Embeddings - semantic retrieval

Embeddings map text into a vector space where **meaning** is distance, so paraphrases land near each other even with no shared words. In the demo a synonym-expanded overlap **stands in** for this:

```
SYNONYMS = {"broken": ["fail", "fails", "dead"], "turn": ["power", "start", "startup", "boot"], ...}
# "broken gadget" now overlaps the power/startup document (broken -> fail/dead)
```

- **Wins** the keyword-free paraphrase: `power_issues.md` surfaces even though **"broken gadget"** shares no word with any document.
- **Loses** on exact IDs: a rare code like `E_4096` can get diluted among "semantically similar" text.

> **Teaching point:** reach for embeddings when the user describes a problem in their own words. They cost more (you embed every chunk and the query) but they recover the recall BM25 misses.

Concept 3 · RRF - fusing the two rankings

Each retriever produces a different **ranked list**. **Reciprocal Rank Fusion** combines them without needing comparable scores - each list contributes `1 / (k + rank)` to a document, then you re-sort:

```
fused[name] += 1.0 / (RRF_K + position + 1)     # RRF_K = 60
```

Because it works on **ranks, not raw scores**, BM25 and embeddings get an equal vote despite living on totally different scales. In the demo, the fused ranking gets **both** queries right where each single retriever gets only one right.

> **Teaching point:** RRF is the cheapest, most robust way to combine retrievers - a few lines, no tuning of score scales. It's the default first step of any hybrid system.

Concept 4 · Reranking, query rewriting & metadata filtering

Three more levers that production systems add on top of fusion:

- **Reranking** - take the fused top-k and reorder them with a stronger model (a cross-encoder that scores each `(query, chunk)` pair). Retrieval optimizes **recall**; reranking optimizes **precision** at the very top, where the LLM actually reads.
- **Query rewriting** - rewrite a vague question into a better retrieval query **before** searching (**"it keeps crashing" → "device repeatedly powers off overheating E_4096"**).
- **Metadata filtering** - restrict candidates by structured fields (file type, product, version, date, page, module) before or after ranking, to cut noise in large mixed corpora.

> **Teaching point:** reranking is usually the single biggest quality win - and the main added cost and latency. Add it once you've proven fusion isn't enough (and measure it - see Lesson 5 · RAG evaluation).

When each technique wins

Technique	Best for	Weak at	Cost	Needs a model?
BM25	exact terms: IDs, error codes, names	paraphrase, synonyms	very low	no
Embeddings	semantic similarity, paraphrasing	exact tokens, rare IDs	medium	yes
RRF (fusion)	combining BM25 + embeddings	nothing - it's combinatorial	negligible	no
Reranking	precision at the top-k	adds latency; needs top-k first	higher	yes
Query rewriting	vague / underspecified questions	can over-expand	one LLM call	yes
Metadata filtering	scoping large mixed corpora	needs good metadata	negligible	no

On the demo corpus BM25 nails the exact code but returns **nothing** for the keyword-free paraphrase, while the semantic arm recovers the right document - and RRF keeps whichever arm found the answer. That's the whole argument for fusion: you never lose one retriever's strength.

Polyglot by design

The retrieval algorithm is language-agnostic, so this lesson ships in **Python, Node.js, and C# / .NET** - each dependency-free, each reading the same `data/`, each producing byte-identical rankings. Compare the three implementations side by side: the **concepts** (BM25, synonym expansion, RRF) are the same; only the syntax changes.

Port	Entry point	Run
Python	`python/hybrid_demo.py`	`./run -l 3 demo` · `./run -l 3 test`
Node.js	`node/hybrid_demo.mjs`	`./run -l 3 --lang node demo`
.NET 8	`dotnet/Program.cs`	`./run -l 3 --lang csharp demo`

> All three commands are declared once in `lesson.json` - the single source of truth the `./run` engine reads. `./run -l 3 show` renders this lesson's elements (notes, code, sample data, prompts, commands) in order.

Exercises

- **Real embeddings:** replace the semantic stand-in with a local sentence-embedding model and rerun both queries - does fusion still help?
- **Tune RRF:** change `RRF_K` and the per-arm weighting; observe the effect on the fused order.
- **Add a third retriever:** fold in a title/metadata match as a third ranked list - RRF takes any number of lists.
- **Break it:** add a document that mentions **E_4096** in passing and watch how BM25 vs hybrid handle the near-duplicate.

From demo to production

- **Swap in real embeddings** for the semantic arm - a local sentence-embedding model, or the repo's optional embedding path. The demo's synonym stand-in exists only to stay 100% offline; a real embedder is what lets the semantic arm recover the paraphrases BM25 misses.
- **Add a cross-encoder reranker** over the fused top-k. This is usually the single biggest quality win: a reranker scores each (query, passage) pair jointly instead of independently, at some extra latency and compute. Rerank only the top-k and cache the scores to keep it affordable.
- **Rewrite and expand the query** with an LLM step before retrieval - split multi-part questions, add synonyms, and resolve pronouns - so vague or conversational queries still reach the right chunks.
- **Index richer metadata** so you can filter before you rank: product, version, date, language, and section. A lot of "wrong answer" bugs are really "retrieved the wrong document" bugs that a metadata filter would have prevented.
- **Tune the knobs per corpus** - `RRF_K`, the per-arm top-k, and the per-arm weights all shift the fusion. There is no universal setting; the right values depend on how lexical vs. semantic your corpus is.
- **Prove it with numbers** - pair this with **Lesson 5 - RAG evaluation** and show hybrid actually beats either arm alone on your golden set before you ship it. "Felt better" is not a reason to add a reranker.

Next lesson

Lesson 4 · RAG safety & prompt injection - retrieved documents are untrusted input; never follow instructions found inside them.
