Case Study 8 — Web Scraping to Vector Database

IT 721 — Applied Research Topics in Deep Learning · Antonio Gonzalez · August 2026

Summary. A complete retrieval-augmented generation pipeline was built and executed end to end on a three-container Docker stack (ChromaDB, Ollama, OpenWebUI). Three public pages on retrieval-augmented generation were checked against robots.txt, scraped, cleaned, chunked into 54 passages, embedded with SentenceTransformers and stored as 384-dimensional vectors. Retrieval was then evaluated across 20 configurations, compared against a lexical baseline, and wired to a local LLM for grounded generation. The work produced three findings that contradict common assumptions about RAG tuning, and surfaced two real defects — one compliance, one citation integrity — that are documented rather than hidden.

1. URL Selection Rationale

The three sources share one topic and differ in *editorial register*. That choice is the experimental design, not a convenience. If the sources had covered unrelated subjects, any embedding model would separate them on topic alone, every configuration in Section 3 would score identically, and the experiment would measure nothing. Holding the topic constant while varying vocabulary, depth and rhetorical style forces the retriever to discriminate on meaning — which is the capability this case study exists to demonstrate.

Source (domain)	Category	Register	Yield
en.wikipedia.org — Retrieval-augmented generation	Educational / encyclopedic	Neutral, citation-dense	21,024 chars
aws.amazon.com — What is RAG?	Cloud-vendor documentation	Instructional, architectural	10,781 chars
blogs.nvidia.com — What Is RAG	Industry blog	Narrative, analogy-driven	10,698 chars

All three are publicly accessible without login, in English, text-heavy, and served as server-rendered HTML rather than JavaScript-only shells — which is what makes extraction reliable. Each slot carries a documented cross-domain fallback, used only if robots.txt denies the primary URL.

2. Technical Summary

Pipeline: scrape → robots check → clean → chunk → embed → store → search → generate.

Layer	Component	Runtime	Endpoint
Orchestration	Jupyter notebook (venv, Python 3.11)	macOS host, Apple Silicon	VS Code
Vector store	chromadb/chroma:latest	Docker container	localhost:8000
Embedding	all-MiniLM-L6-v2 / all-mpnet-base-v2	host process, MPS accelerated	in-process
Generation	llama3.2:3b via ollama/ollama	Docker container	localhost:11434
Chat UI	ghcr.io/open-webui/open-webui:main	Docker container	localhost:3000

A deliberate architectural choice: embeddings are computed **on the host** and pushed to ChromaDB as pre-computed vectors rather than letting the server embed. This keeps the container free of an embedding runtime, makes the embedding model an explicit and swappable experiment variable, and — most importantly — guarantees that index-time and query-time vectors come from the same model instance. Vectors are L2-normalised, so cosine distance and inner product agree and `similarity = 1 − distance` is well defined.

Engineering controls: configuration is externalised to a .env file; requests carry a single identifying User-Agent with a contact address; a 1.5-second inter-request delay rate-limits the crawler; failures retry with exponential backoff; and the

ChromaDB collection factory attempts both the modern configuration= and legacy metadata={"hnsw:space"} spellings so the notebook runs against either server image.

3. Results Analysis

3.1 What was captured and stored

54 chunks were produced from roughly 26,000 words, distributed 26 / 14 / 14 across Wikipedia, AWS and NVIDIA. That 2:1:1 imbalance is inherited from extraction yield, and it means Wikipedia holds about 48% of the index — a structural prior that reappears in the search results. Storage verified at **54 vectors × 384 dimensions** in cosine space, built in 4.10 s (13.3 chunks/s on MPS).

Three properties are asserted rather than eyeballed: the indexed count matches the corpus, the stored vector length matches the model's declared dimension, and the **L2 norm of a stored vector is 1.0000** — confirming that normalisation actually took effect at encode time. That check earns its place: cosine space assumes unit-length vectors, and had normalisation silently failed, `similarity = 1 − distance` would be wrong while still returning plausible-looking rankings.

The build timing is the most instructive number in this section, because it is *not reproducible*. Four executions indexed the same 54 chunks and reported 11.53 s, 3.84 s, 19.48 s and 4.10 s — a 5× spread driven entirely by whether the embedding model was already warm. The timer wraps a call that lazily loads the model, so on the slowest run 13.1 s of the 19.48 s was `SentenceTransformer(...)`, not indexing. It is reported as measured, with its composition documented, rather than quietly replaced by the fastest run: this is the single most common way an embedding benchmark ends up wrong by a factor of several, and the fix is to warm the model outside the timed region and report encode and upsert separately.

Everything measuring *quality* rather than wall-clock was fully deterministic across all four runs — MRR 0.9375, precision@5 0.925, Jaccard 0.23, one citation leak. That reproducibility is what makes the conclusions below trustworthy.

3.2 Configuration matrix — 20 runs

Chunk size × overlap × embedding model, each materialised in its own ChromaDB collection and scored on a fixed set of eight information-need queries.

Model	Best geometry	MRR	precision@5	Chunks	Index
all-MiniLM-L6-v2 (384d)	`1000 / 300`	`0.9062`	`0.775`	`60`	0.21 s
all-mpnet-base-v2 (768d)	`1500 / 300`	`0.9375`	`0.925`	`36`	0.98 s

Finding 1 — hit@5 is useless at this scale, and that is the first result.

hit@5 reached 1.0 in 16 of 18 configurations. On a 54-chunk corpus where all three documents explain the same concept, "at least one relevant chunk in the top 5" is satisfied almost regardless of configuration. MRR and precision@5 are the metrics that carry information here — reporting hit@5 as a success measure would have concealed every real difference.

Finding 2 — absolute similarity tracks chunk length, not relevance.

Mean top-1 similarity *falls* as chunks grow (0.5305 at 500/300 → 0.4059 at 1500/300) while ranking quality *improves*. Longer passages average more concepts into a single vector, lowering peak cosine similarity even as the passage becomes more useful. The practical consequence is concrete: a fixed similarity threshold ("reject anything below 0.45") is not portable across chunk geometries — it silently becomes stricter as chunks grow. Rank, do not threshold.

Finding 3 — high overlap is a storage tax with no retrieval return.

The 500/300 configuration produces 208 chunks — 5.8× the storage of 1500/300 — for a lower MRR (0.900 vs 0.9375). A 300-character overlap on a 500-character chunk is 60% redundancy: the same sentences are embedded three times over.

Finding 4 — the chunking hypothesis was not supported.

The pipeline replaces the template's character-offset slicer with a sentence-aware packer, on the rationale that cutting mid-sentence produces noisy embeddings. The ablation disagreed: at 1000/200, `fixed_char` scored MRR 0.8438 against sentence_aware's 0.8167, with identical precision@5. At a 1000-character budget a chunk already spans many sentences, so one damaged boundary is a small fraction of the vector; and the sentence-aware variant's minimum-length filter drops fragments the baseline keeps, so the two are not scored on identical text. The hypothesis is plausible at 200–400 character chunks and was simply not tested where it would matter. Reported as measured, not as intended.

4. Search Examples

Five queries were issued against the production collection, written as natural information needs. Three deliberately use vocabulary absent from the corpus, so a hit can only come from semantic proximity.

Query 1 — "How does grounding a chatbot in external documents stop it from making things up?"

Design: paraphrase test. The corpus says `"hallucination"`; the query says `"making things up"`.

- #1 sim 0.4035 [en.wikipedia.org] — the RAG definition passage
- #2 sim 0.3733 [aws.amazon.com] — "Presenting false information when it does not have the answer"

Neither chunk contains the query phrase. This is the clearest single demonstration that the index retrieves on meaning rather than string overlap.

Query 2 — "What infrastructure is needed to turn a company's internal documents into a searchable knowledge base?"

Design: architecture intent — should favour the implementation sources over the encyclopedic one.

- #1 sim 0.5780 [aws.amazon.com] — semantic search for organisations adding external knowledge sources
- #2 sim 0.5169 [aws.amazon.com] — knowledge bases for Amazon Bedrock

The highest similarity of the entire set, and it correctly preferred AWS over Wikipedia — the register diversity in the source selection paying off directly.

Query 3 — "Is it cheaper to retrain a model on new data or to look the data up at answer time?"

- #1 sim 0.3942 [aws.amazon.com] — the information-retrieval component

A comparative trade-off phrased as reasoning, with no single anchor term. It was the only query to pull all three domains into the top 5 — the expected behaviour for a question no single source answers alone.

4.1 Semantic vs. lexical retrieval — a measurement ceiling

A keyword baseline was run over the same corpus. It tied with semantic retrieval on both query sets (literal MRR 0.817 vs 0.844; paraphrase 1.000 vs 1.000). Taken alone that suggests the vector store adds nothing. A set-overlap diagnostic shows otherwise:

Query set	Semantic MRR	Lexical MRR	Jaccard@5	Top-1 agreement
Literal (n=8)	`0.817`	`0.844`	`0.303`	`3 / 8`
Paraphrase (n=4)	`1.000`	`1.000`	`0.083`	`0 / 4`
Overall	—	—	`0.23`	`5 / 12`

The two methods share barely a quarter of their top-5 results, and on paraphrase probes they *never* agree on which passage is best — one query returned completely disjoint sets. Identical scores over near-disjoint retrievals can only mean the scoring function cannot tell the retrievals apart. The cause is the lexical relevance proxy: with 54 chunks all discussing RAG, nearly every chunk contains a gold anchor term, so both methods saturate. Note the structure of the failure — divergence is largest exactly where theory predicts, 0.083 on paraphrase versus 0.303 on literal. This is a measurement ceiling, not a null result.

4.2 Grounded generation and the negative control

Question	top-1 similarity	Outcome
What problem does RAG solve?	`0.6325`	Answered with citations
Main components of a RAG pipeline?	`0.3462`	Answered as a 7-step pipeline
What is the population of Saltillo?	`0.0652`	"The indexed sources do not cover this."

The out-of-corpus control produced a top-1 similarity an order of magnitude below the answered questions, and the model abstained verbatim rather than improvising from parametric memory. Abstention on out-of-domain input is what separates a grounded system from a merely fluent one, and the 0.065-vs-0.633 gap shows the retriever — not the prompt alone — is what makes that abstention reliable. That gap is also the natural place to set an operational retrieval floor.

5. Challenges Faced

5.1 A false robots.txt denial that the scraper then ignored

The first execution logged en.wikipedia.org as disallowed — and scraped it anyway. Log and behaviour contradicted each other.

Root cause: `RobotFileParser.read()` fetches robots.txt with urllib's default `Python-urllib/3.x` User-Agent. Wikimedia answers that UA with HTTP 403, and on 401/403 the parser sets `disallow_all = True` — an entire domain denied by an artefact of the HTTP client, not by any rule Wikipedia publishes.

Fixed by fetching robots.txt with the same identifying User-Agent used for page requests. The verdict inverted and is now trustworthy:

```
wikipedia  ALLOW  parsed OK (HTTP 200, 712 lines); no crawl-delay declared
```

The second correction matters as much: the verdict is now **enforced**. A source still disallowed after its fallback is flagged and never fetched, and an assertion halts the run rather than proceeding on two sources. A compliance check that is logged but not acted on is worse than no check — it produces an audit trail certifying the wrong thing.

5.2 Citation leakage in generated answers

One of the five URLs the model emitted was not an indexed source — a 20% leak rate, reproduced across both runs:

```
LEAK -> https://ars-technica.com/2024/06/06/can-a-technology-called-rag-keep-ai-...
```

The model did not fabricate it. That URL lives inside the scraped Wikipedia text as one of the article's own references; the model lifted it from the chunk body and presented it as provenance. The prompt said "cite the source URL", and from the model's position a URL in the context is a source URL.

This breaks the one property that makes a RAG answer auditable — and it does so while looking well-sourced. The other two answers cited only legitimate indexed URLs, so the failure is intermittent, which is worse than systematic: spot-checking would likely miss it.

Fix, as design rather than prompt engineering: provenance is a property of the retrieval step, so it must never be delegated to the model. Cite from `metadatas[i]["url"]` of the retrieved chunks; strip bare URLs and reference markers from chunk bodies at ingest; and post-validate every emitted URL against the allow-list. The defect is deliberately left visible in the notebook output so the validation check has something to demonstrate.

5.3 Others

- **ChromaDB API drift.** The distance metric moved from metadata={"hnsw:space"} to configuration={"hnsw": {...}}. The collection factory attempts both spellings so the notebook runs against either server image.
- **A retriever comparison that saturated.** Diagnosed with set-overlap analysis rather than explained away in prose (Section 4.1).
- **Kernel working directory.** VS Code resolves the notebook root differently from Jupyter Lab, which can leave .env unloaded. Paths are anchored on the project folder rather than on the process CWD.
- **Ollama on Apple Silicon in Docker.** No Metal passthrough, so generation is CPU-bound (4.8–13.7 s per answer). llama3.2:3b was chosen to stay comfortable within that constraint.

6. Ethical Considerations

- robots.txt parsed and **enforced** per URL before any request, with the verdict written to an auditable evidence file.
- One honest, identifying User-Agent carrying a contact address — no browser spoofing.
- 1.5 s between requests, a single fetch per page, no recursive crawling.
- Every chunk stores its source URL, so attribution survives into retrieval and generation.
- Content used solely for this coursework; no redistribution of source text.

6.1 Limitations and next steps

Build relevance judgements by hand for 30–50 query/chunk pairs so retrieval quality is measured rather than proxied. Widen the corpus to a dozen topically distinct sources so anchor terms stop being ubiquitous and the semantic-vs-lexical comparison can actually separate. Normalise chunk counts across sources to remove the 48% Wikipedia tilt. And test chunk sizes at 200–400 characters, where the sentence-boundary hypothesis should finally have room to show an effect.

Appendix A — Evidence

A.1 Container stack

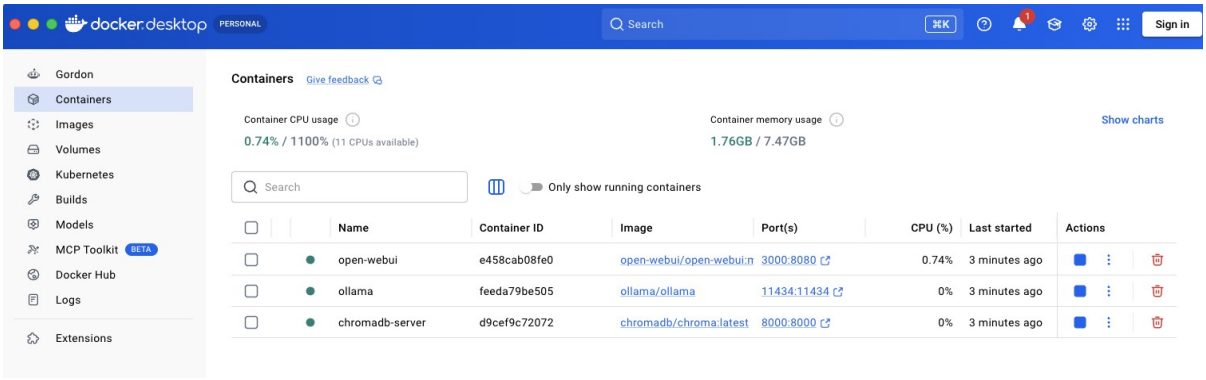

Figure 1 — Docker Desktop: chromadb-server, ollama and open-webui all running (green).

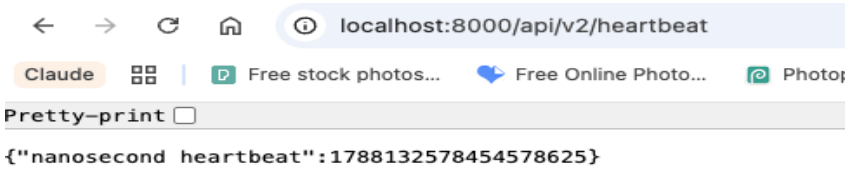

Figure 2 — ChromaDB v2 heartbeat responding on localhost:8000.

A.2 Experiment results

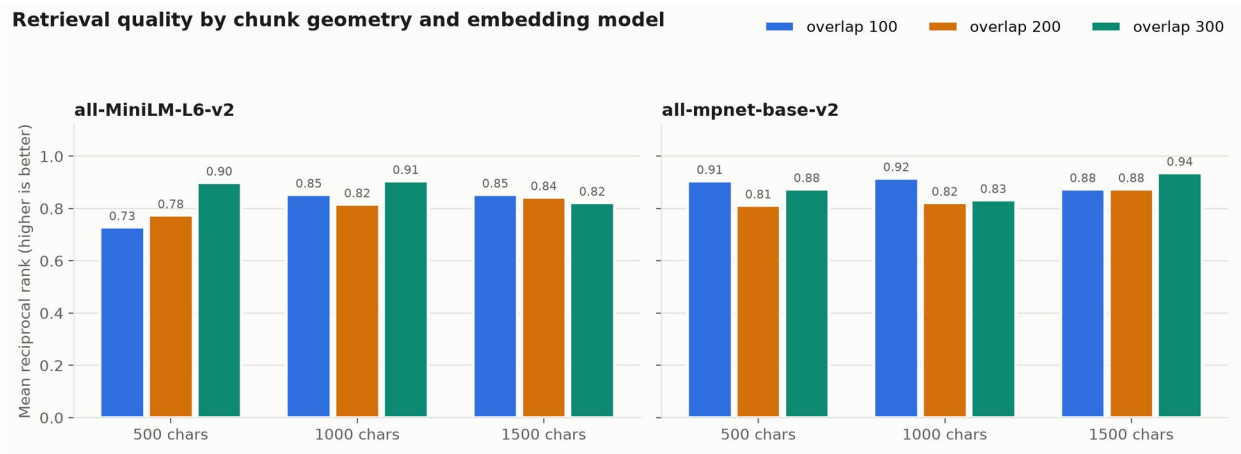

Figure 3 — Retrieval quality (MRR) by chunk geometry, faceted by embedding model.

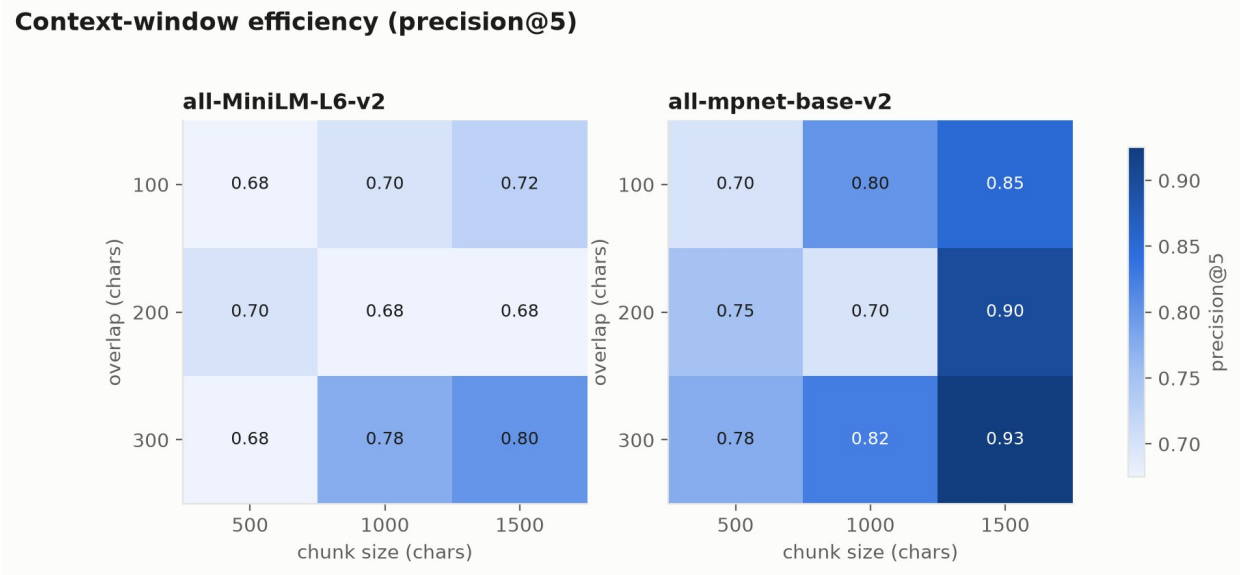

Figure 4 — Context-window efficiency (precision@5). MPNet rises monotonically with chunk size; MiniLM does not.

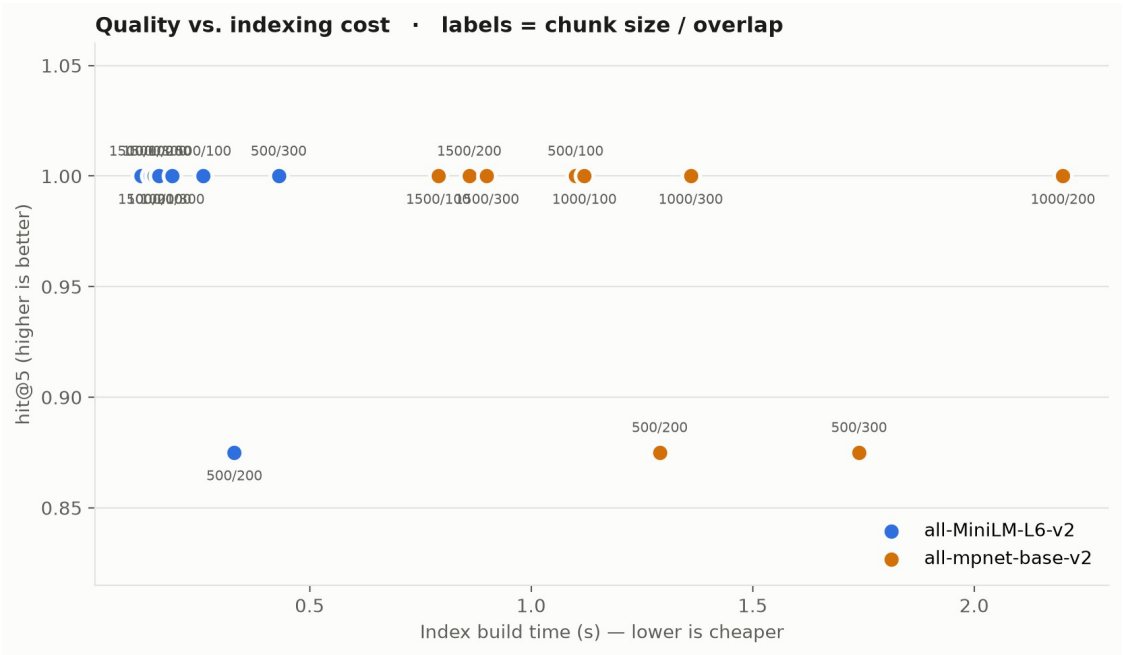

Figure 5 — Retrieval quality against index build cost; labels are chunk size / overlap.

A.3 Chat interface

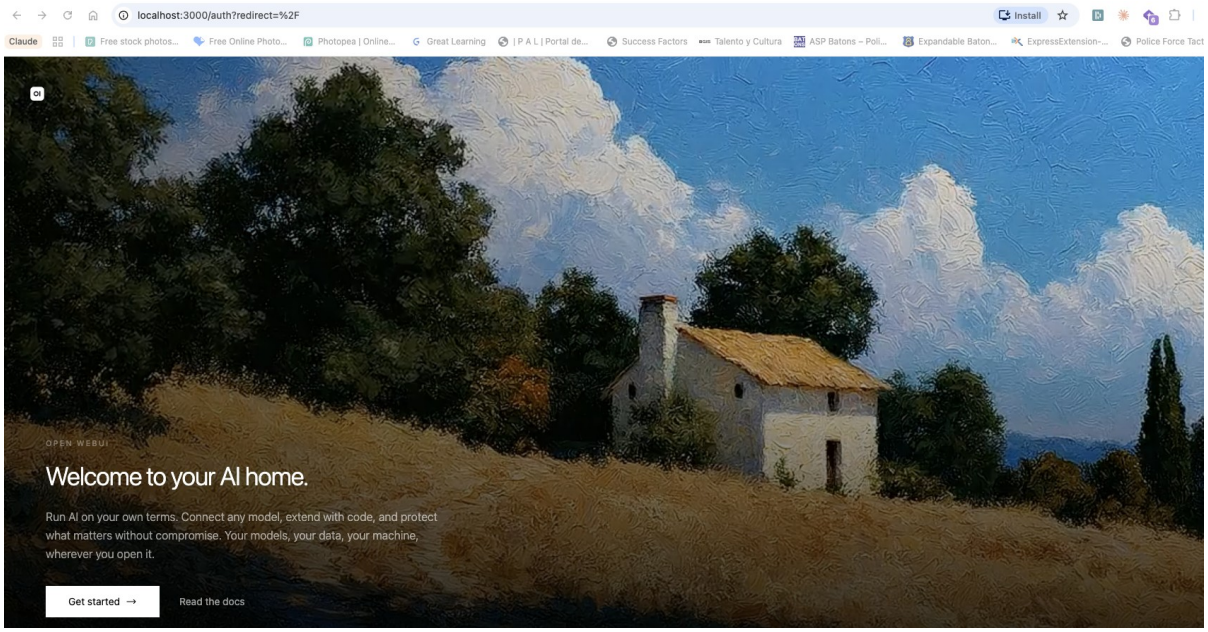

Figure 6 — OpenWebUI served from the open-webui container on localhost:3000.

A.4 Machine-readable evidence bundle

All figures above are reproduced by re-running the notebook. The evidence/ folder additionally contains: environment probes and container state (00), robots.txt verdicts (01), scrape summary (02), corpus statistics (03), ChromaDB storage verification (04), the full 20-run experiment matrix as CSV and JSON (05), the semantic search transcript (06), the retriever comparison (07), generated answers (08), the run manifest and checklist (09), the retriever disagreement analysis (10), and the citation integrity report (11).