RAGGAE: A multipurpose local RAG system for Adservio

Olivier Vitrac, PhD., HDR | olivier.vitrac@adservio.fr – 2025-10-24

Summary

This note discusses the design of a **generic RAG/embeddings library** can serve **CVs, reports, and tenders**, which relies on different *document adapters* using a shared *semantic core* (retrieval + re-rank + annotation + scoring). A **hybrid (dense+sparse) + cross-encoder** is proposed. The POC adds **domain-tuning** and **NLI checks**, and is designed from day one for **traceability** (provenance spans, scores, reasons). The whole system is designed to run on minimal infrastructure: fully local MVP – GPU with 8 GB VRAM and possibly running on CPU.

> *Access to all **files**, read this file in **PDF***

1 | Technical Review

1.1 | Embedding options (and when to use which)

A. Dense text embeddings (bi-encoders) — default for RAG

- **General English/Multilingual:** E5-family, GTE-family, bge-family, Jina, Sentence-Transformers (MiniLM, MPNet), Cohere, OpenAI, etc.
- **Pros:** fast, scalable, cheap to store/query; perfect for "retrieve top-k chunks."
- **Cons:** retrieval scores are approximate; for high-precision ranking add a re-ranker.

B. Cross-encoders (re-rankers) — for precision at the top

- BERT/DeBERTa/Modern LLM cross-encoders (e.g., *ms-marco*-tuned) that score (query, passage) jointly.
- **Use:** take the top 50–200 dense hits, re-rank to get very accurate top-10.
- **Trade-off:** slower and costlier per query, but best quality for tenders.

C. Hybrid retrieval (dense + sparse) — when vocabulary matters

- Combine **BM25 / SPLADE** (sparse, exact terms) with dense vectors (semantics).
- **Use:** tenders have jargon, acronyms, legal clauses—hybrid boosts recall on rare terms.

D. Domain-tuned embeddings — when your domain dominates

- Light fine-tuning (or adapters) using your **historic tenders, SoWs, CVs, past responses**.
- **Use:** improves intent matching on "DevOps/MLOps" specifics, vendor boilerplate, compliance phrasing.

E. Multilingual & French

- Choose a **multilingual** model (FR/EN at minimum). If not, keep separate indices per language and route queries.
- Consider **language-aware chunking** and **query translation** as a fallback.

F. Long-document strategies (tenders/CVs/reports)

- **Hierarchical embeddings:** section → paragraph → sentence; route queries to the right level.
- **Layout-aware chunking:** keep tables, bullets, headers/footers; preserve section numbers and annex links.

1.2 | "Semantic analysis" we'll want beyond embeddings

We think of this as *signals* layered on top of retrieval:

- **Document structure parsing:** title, sections, annexes, tables, numbered requirements (MUST/SHALL/DOIT).
- **Keyphrase & requirement mining:** extract capabilities (e.g., "K8s, ArgoCD, MLflow, ISO 27001, on-prem"), constraints (SLA, RPO/RTO, sovereignty).
- **NER & taxonomy mapping:** map entities/skills/standards to an **Adservio capability ontology** (DevOps, MLOps, Security, Cloud, Data).
- **Entailment/NLI checks:** "Does our offer satisfy clause 4.2.3?" (Yes/No/Partial + rationale).
- **De-duplication & canonicalization:** normalize synonyms ("GPU farm" ≈ "on-prem compute with NVIDIA A-series").
- **Risk & eligibility flags:** deadlines, mandatory certifications, exclusion criteria, IP/sovereignty clauses.

These features feed your **scoring/ranking** (fit, risk, attractiveness) and later your **form pre-fill**.

1.3 | Can one library handle CVs, reports, tenders? (Yes—if you design it right)

Design a **document-agnostic semantic layer** with adapters:

- **Core abstractions:**
  - `Document` (metadata + pages + spans + tables)
  - `Chunk` (text, layout anchors, section path)
  - `EmbeddingProvider` (pluggable: dense, sparse, hybrid)
  - `Indexer/Retriever` (vector DB + BM25)
  - `Reranker` (cross-encoder)
  - `Annotator` (NER, keyphrases, taxonomy linker)
  - `Scorer` (tender-fit, confidence, risk)
  - `Extractor` (field mappers for pre-fill)
- **Adapters per doc type:** `TenderAdapter`, `CVAdapter`, `ReportAdapter` implement:
  - **Parsing rules** (e.g., numbered requirements vs. experiences vs. results)
  - **Chunking rules** (keep bullets, tables, job periods)
  - **Field mappers** (e.g., "Lot 2 scope" → `scope.devops`, "Years exp in K8s" → `cv.skills.k8s.years`)

Result: *same* embedding/retrieval engine, *different* adapters and scoring logic.

1.4 | Minimal technical blueprint

```
class EmbeddingProvider:
    def embed_texts(self, texts: list[str]) ->
list[list[float]]: ...
    def embed_query(self, text: str) -> list[float]: ...

class DenseBiEncoder(EmbeddingProvider): ...
class SparseBM25: ...
class HybridRetriever:
    def __init__(self, dense: EmbeddingProvider, sparse:
SparseBM25, alpha=0.6): ...
    def search(self, query: str, k=100) -> list["Hit"]: ...

class CrossEncoderReranker:
    def rerank(self, query: str, hits: list["Hit"], top_k=20) -
> list["Hit"]: ...

class DocumentAdapter:
    def parse(self, raw_bytes) -> "Document": ...
    def chunk(self, doc: "Document") -> list["Chunk"]: ...
    def annotate(self, chunks) -> list["Chunk"]: ...
    def score(self, query, chunks) -> list["ScoredChunk"]: ...

# Pipeline
adapter = TenderAdapter(lang="fr")
doc = adapter.parse(pdf_bytes)
chunks = adapter.chunk(doc)
vectors = dense.embed_texts([c.text for c in chunks])
index.upsert(chunks, vectors, metadata=adapter.annotations)

hits = hybrid.search(query, k=150)
hits = reranker.rerank(query, hits, top_k=25)
```

1.5 | Choosing an embedding setup (quick decision guide)

- **Early phase / fast demo:** Multilingual dense bi-encoder + BM25 hybrid; add a small cross-encoder re-ranker.
- **Production quality for tenders:** Same as above **plus** (a) domain-tuning on historical tenders & responses, (b) taxonomy-aware scoring, (c) NLI compliance checks.
- **High privacy / on-prem:** Prefer **open models** (no external API), self-host vector DB (FAISS, Qdrant, Milvus).
- **Strict FR/EN mix:** Multilingual embeddings *or* per-language indices with automatic routing.
- **Lots of tables/forms:** Ensure **layout-aware parsing** (tables become key-value triples; keep cell coordinates).

Absolutely feasible locally: E5-small + BM25 + optional cross-encoder, FAISS index, Ollama (7–8B Q4) for NLI/extraction.

One generic library with adapters lets you handle tenders, CVs, and reports with the same semantic core.1.6 | Ranking & classification for tenders

- **Relevance ranking:** Hybrid retrieve → cross-encode re-rank.

- **Fit scoring:** weighted signals (must-haves met, certifications present, tech match, budget window, delivery window, jurisdiction).
- **Classification buckets:** DevOps/MLOps/Lot-based labels via:
    - **Zero-shot** (NLI prompt + label descriptions) for cold start.
    - **Few-shot supervised** (logistic regression or small classifier on embeddings) once you have labeled data.
    - **Topic modeling** (BERTopic/Top2Vec on embeddings) for discovery of recurring themes.

1.7 | Toward pre-filling response forms (step 2)

- **Field schema registry:** define each target field with a canonical name, regex/ontology, and examples.
- **Extractor chain:** retrieval → NER/regex → NLI validation → *LLM with constrained generation* to map spans to fields.
- **Traceability:** keep source spans + page numbers (for audit and human review).
- **Safety gates:** mandatory fields coverage, confidence thresholds, red-flag clauses (IP/sovereignty/insurance).

1.8 | Evaluation of prototype from day 1

- **Retrieval:** Recall@k, nDCG on a seed set of queries (FR/EN).
- **Re-ranking:** MRR@10, precision@5.
- **Classification:** F1 per class, macro-F1; calibration curve.
- **Extraction (pre-fill):** exact-match / relaxed-match and **provenance coverage** (% fields with verified source span).
- **Human-in-the-loop:** review time saved per tender.

1.9 | Practical shortlist (safe bets to prototype)

- **Dense bi-encoder:** a strong multilingual Sentence-Transformers-style model (or equivalent GTE/bge multilingual).
- **Sparse:** BM25; consider SPLADE later if needed.
- **Re-ranker:** MS-MARCO-style cross-encoder or a modern cross-encoder fine-tuned on your domain pairs.
- **Vector DB:** FAISS (embedded) → Qdrant/Milvus (server) when scaling.
- **Parsers:** pdfminer/pymupdf + table extraction (camelot/tabula) + a layout-retaining schema.

Bottom line

- **Yes**, one **generic RAG/embeddings library** can serve **CVs, reports, and tenders** if you separate *document adapters* from a shared *semantic core* (retrieval + re-rank + annotation + scoring).
- Start **hybrid (dense+sparse) + cross-encoder**, add **domain-tuning** and **NLI checks**, and design from day one for **traceability** (provenance spans, scores, reasons).
- This sets you up cleanly for step-2 **form pre-fill** with auditable mappings..

---

2 | Local MVP stack (FR/EN tenders, CVs, reports)

2.1 | Retrieval (dense)

- **Multilingual small (fits easily):**
  - `intfloat/multilingual-e5-small` (~33M) or `sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2` (~118M).
- **English-optimized (if you need max quality in EN):**
  - `thenlper/gte-small` or `Alibaba-NLP/gte-base-en-v1.5` (base is fine on CPU/GPU).
- Tip: start with **`multilingual-e5-small`** for FR/EN, upgrade to `multilingual-e5-base` when you want a tiny quality boost.

2.2 | Re-ranking (cross-encoder)

- **Light & accurate:** `cross-encoder/ms-marco-MiniLM-L-6-v2` (EN).
- **Multilingual option:** `jinaai/jina-reranker-v1-base-multilingual` (base size, still comfy on 8 GB). Use it only on top-100 dense hits → top-20 final.

2.3 | Sparse retrieval (for jargon & exact clauses)

- **BM25:** `rank_bm25` (pure Python) to start.
- Later: Elastic (OpenSearch) or SPLADE if recall needs help.

2.4 | Vector store

- **FAISS** for embedded mode (simple and fast).
- Optional server mode later: **Qdrant** (Docker) when you need multi-user + filters.

2.5 | Parsers & chunking

- **PyMuPDF (`fitz`)** + metadata/page anchors.
- **Camelot**/`tabula` for tables → convert to key-value triples with cell coordinates.
- Chunk by sections/bullets; keep **(doc_id, section_path, page, bbox)** in metadata for traceability.

1.6 | Local NLI/extraction (for "does this clause match?" and pre-fill)**

- With **Ollama**: `mistral:7b-instruct` or `llama3:8b-instruct` in **Q4_K_M** quant runs on 8 GB.
- Use for: entitlement checks, short rationales, and field extraction with **constrained prompts**.

---

3 | Minimal pipeline (drop-in code)

```
# pip install sentence-transformers faiss-cpu rank-bm25 pypdf
pymupdf tqdm
from sentence_transformers import SentenceTransformer
import faiss, numpy as np
from rank_bm25 import BM25Okapi
import fitz  # PyMuPDF

# 1) Parse & chunk
def parse_pdf(path):
    doc = fitz.open(path)
    chunks = []
```

```
    for pno in range(len(doc)):
        page = doc[pno]
        text = page.get_text("blocks")  # retains block order
        for i, (_, _, _, _, t, _, _) in enumerate(text):
            t = (t or "").strip()
            if len(t) > 40:
                chunks.append({"text": t, "page": pno+1,
"block": i})
    return chunks

chunks = parse_pdf("tender.pdf")
texts = [c["text"] for c in chunks]

# 2) Dense embeddings
model = SentenceTransformer("intfloat/multilingual-e5-small")
# e5 expects "query: ..." vs "passage: ..." prefixes for best
results
passages = [f"passage: {t}" for t in texts]
E = np.vstack(model.encode(passages,
normalize_embeddings=True))

# 3) FAISS index
index = faiss.IndexFlatIP(E.shape[1])
index.add(E.astype("float32"))

# 4) BM25
bm25 = BM25Okapi([t.split() for t in texts])

# 5) Hybrid search
def hybrid_search(q, k_dense=100, k=20, alpha=0.6):
    q_dense = model.encode([f"query: {q}"],
normalize_embeddings=True)
    D, I = index.search(q_dense.astype("float32"), k_dense)
    dense_scores = {i: float(s) for i, s in zip(I[0], D[0])}
    # BM25 scores
    bm = bm25.get_scores(q.split())
    # Normalize BM25
    bm = (bm - bm.min()) / (bm.ptp() + 1e-9)
    # Fuse
    fused = []
    for i, ds in dense_scores.items():
        fs = alpha*ds + (1-alpha)*float(bm[i])
        fused.append((i, fs))
    fused.sort(key=lambda x: x[1], reverse=True)
    return [chunks[i] | {"score": s} for i, s in fused[:k]]

hits = hybrid_search("ISO 27001, MLOps platform avec MLflow et
K8s")
for h in hits[:5]:
    print(h["score"], h["page"], h["text"][:120], "…")
```

> *Swap in a cross-encoder re-ranker later (e.g., `jinaai/jina-reranker-v1-base-multilingual`) on the `hits[:100]` to boost precision@5.*

---

4 | Using Ollama locally (NLI/extraction)

```
# examples: mistral & llama3 in 4-bit quant
ollama pull mistral:latest
ollama pull llama3:8b

# Python: pip install ollama
import ollama

PROMPT = """You are a compliance checker.
Clause: "{clause}"
Requirement: "Provider must be ISO 27001 certified"
Answer with JSON: {{"label":"Yes/No/Partial","rationale":
"..."}}
"""

def nli_check(clause):
    r = ollama.chat(model="mistral", messages=
[{"role":"user","content":PROMPT.format(clause=clause)}])
    return r["message"]["content"]
```

---

5 | What fits in 8 GB VRAM (comfortably)

- **Embeddings:** "small/base" sentence-transformers (CPU or GPU).
- **Re-rankers:** MiniLM-class and multilingual base rerankers (GPU helps; CPU is fine).
- **LLM for reasoning/extraction: 7B–8B** quantized via Ollama (Q4_) — good for short answers and NLI.
- You don't need bigger models for step-1 retrieval/ranking.

---

6 | Can the *same* lib read CVs, reports, tenders? Yes — via adapters

Keep a shared semantic core and add thin adapters:

- `TenderAdapter`: numbered requirements (MUST/SHALL), lots/eligibility, deadlines.
- `CVAdapter`: roles, durations, skills, certs; normalize to a **capability ontology** (e.g., `devops.k8s`, `mlops.mlflow`, `security.iso27001`).
- `ReportAdapter`: sections, methods, results, conclusions, annexes/tables.

All three reuse **the same**: parser → chunker → embeddings → FAISS/BM25 → (optional) reranker → scorers.

---

7 | Folder scaffold (ready to `uv`/`pip`)

```
adservio-tender-ai/
  core/
    embeddings.py          # providers (E5, GTE, bge…)
    retriever.py           # hybrid retrieve
    reranker.py            # optional cross-encoder
    index_faiss.py         # vector index
    scoring.py             # signals + weighted fit score
    nli_ollama.py          # local NLI/extractor
```

```
io/
  pdf.py                # PyMuPDF parsing
  tables.py             # camelot/tabula wrappers
adapters/
  tenders.py            # parse/chunk/fields
  cv.py                 # parse/chunk/fields
  reports.py
cli/
  index_doc.py          # index PDFs
  search.py             # query + show provenance
  quickscore.py         # tender fit score
data/
  ontology.yaml         # skills, certs, standards
  labels/               # few-shot seeds for classifiers
```

---

8 | Early-phase eval (so you can show value next week)

- **Retrieval:** Recall@50 on 10–20 real tender questions (FR/EN).
- **Top-k quality:** nDCG@10 with cross-encoder on/off (demo the delta).
- **Classification:** Zero-shot labels (DevOps/MLOps/Lot) → quick F1 from a tiny hand-labeled set.
- **Traceability:** Every hit printed with `(doc, page, block, score)` — reviewers love this.

TL;DR

- Absolutely feasible locally: **E5-small + BM25 + optional cross-encoder**, FAISS index, **Ollama (7–8B Q4)** for NLI/extraction.
- One generic library with **adapters** lets you handle **tenders, CVs, and reports** with the same semantic core.
- Start with the code above; one can add the cross-encoder and a simple **fit score** next (must-haves met, tech match, risk flags).