legal-right-extractor - TDD - v1.2

Technical Design Document

Legal Rights Extraction System (Term Sheets / SHA)

Version 1.1 • Updated • Author: Corentin Marion

Goal

Build a **local-first, auditable extraction pipeline** that converts legal documents into a structured portfolio dataset of investor rights, each backed by explicit textual evidence and conflict flags.

The system prioritizes **determinism, traceability, and reproducibility** over coverage or abstraction.

Design Principles

- **Accuracy over coverage**
  Prefer `present = false` over unsupported extraction.
- **Auditability**
  Every extracted item must include a direct quote and provenance (file, page, article when available).
- **Deterministic scaffolding around the LLM**
  Ingestion, chunking, retrieval, validation, conflict detection, and aggregation are explicit steps.
- **Local-first by default**
  Documents and models run locally; no mandatory external services.

Audience

- Engineers building applied extraction systems.
- Technically literate investment and legal operations users reviewing outputs.

1. System Overview

The system is a **pipeline controller** orchestrating deterministic stages and invoking an LLM **only for schema-locked extraction**.

The LLM does not decide, infer, or reconcile.
It extracts or flags uncertainty.

Architecture (MVP)

```
Ingestion
  ↓
Chunking (page / article mapping)
  ↓
Retrieval (BM25 lexical search)
  ↓
Extraction (LLM, schema-locked)
  ↓
Validation (deterministic rules)
  ↓
Conflict Detection
  ↓
Aggregation
  ↓
Export / UI (CSV, XLSX, Streamlit)
```

Key Outputs

- Portfolio table (one row per `RightItem`)
- Per-company detail view with evidence excerpts (≤ 25 words)
- Run metadata for reproducibility

2. Open-Source Stack

All components are usable locally without paid APIs.

Layer	Tools	Rationale
LLM runtime	Ollama, llama.cpp, vLLM	Local inference, controllable

Layer	Tools	Rationale
Retrieval	**BM25 (rank_bm25 / Whoosh / Lucene-style)**	Exact lexical matching
PDF extraction	pdfplumber, pypdf	Page-level mapping
OCR (optional)	ocrmypdf + Tesseract	Fallback only
DOCX parsing	python-docx	Paragraph + heading extraction
Orchestration	Python + Pydantic	Schema enforcement
UI	Streamlit	Internal tooling
Export	pandas + openpyxl	Structured XLSX
Evaluation	promptfoo, Label Studio	Regression and gold sets

Embedding models and vector indexes are intentionally excluded in the current design.

---

3. Data Model and Schemas

All extracted rights are normalized into a single canonical object.

Core Object: `RightItem`

```
company_id: str
clause_family: enum[veto, liquidity, exit, anti_dilution]
clause_type: str
present: bool
beneficiary: enum[investor, founders, company, mixed, unclear]
trigger: str
effect: str
thresholds: str | null
evidence:
  quote: str          # ≤ 25 words
  source_file: str
  page: int | null
  article: str | null
confidence: float       # triage only
flags: list[enum[ambiguous, conflicting, missing_evidence]]
```

Normalization Rules

- One `RightItem` per distinct right.
- No paraphrasing in evidence.
- If evidence is missing, set `present = false` and flag `missing_evidence`.

---

4. Pipeline Design

4.1 Ingestion and Text Extraction

- Detect file type (PDF / DOCX / TXT).
- Store file hash for reproducibility.
- Extract per-page text for PDFs.
- Infer section/article structure where possible.
- OCR is a fallback, not default behavior.

Each extracted span retains `(source_file, page, char_start, char_end)`.

4.2 Chunking and Page Mapping

Chunking preserves legal structure.

- First split by detected articles/sections.
- Then enforce a size cap (≈ 800–1200 tokens).
- Maintain overlap (≈ 10–15%) only within sections.
- Store metadata: company_id, source_file, page_start, page_end, section_title.

4.3 Retrieval (BM25)

Retrieval is **purely lexical**.

- Index all chunks using BM25.
- For each clause family, define a fixed query set (synonyms and drafting variants).
- Example (liquidity):
    - “right of first refusal”

    - “transfer restriction”
    - “lock-up”
    - “preemption”

Top-k chunks (e.g. k = 8–15) are passed to extraction **with full provenance**.

No embeddings, no semantic similarity, no vector index.

---

5. Prompting and Extraction

Prompts are treated as versioned code.

The extractor:

- receives only retrieved chunks,
- outputs **valid JSON only**,
- follows the `RightItem` schema exactly.

Hard constraints:

- No inference.
- No conflict resolution.
- If unsure: `present = false` or `ambiguous`.

---

6. Validation and Aggregation

6.1 Deterministic Validation

- Reject `present = true` without evidence.
- Enforce quote length ≤ 25 words.
- Validate enums and required fields.
- Auto-flag low-confidence items.
- Fill missing page numbers deterministically if available upstream.

6.2 Conflict Detection

- Group by `(company_id, clause_family, clause_type)`.
- Compare normalized trigger / effect / thresholds strings.

- If differences exceed a strict lexical threshold, flag `conflicting`.
- Never override silently.

6.3 Aggregation and Export

- Aggregate validated `RightItem`s into a table.
- Export CSV and XLSX.
- Generate per-company drilldown views.

---

7. Security, Privacy, Reproducibility

- Confidential by default.
- Minimal document retention.
- No logging of full text.
- Version prompts and schemas.
- Store run metadata: model, prompt version, timestamp, git commit.
- File hashing used to detect document changes.

---

8. Engineer Playbook

Step 0 — Environment
Install: pdfplumber, pypdf, python-docx, rank-bm25, pydantic, pandas, openpyxl, streamlit.

Step 1 — Ingestion
Extract per-page text and metadata.

Step 2 — Chunking
Apply rule-based legal segmentation.

Step 3 — Retrieval
Index chunks with BM25; query per clause family.

Step 4 — Extraction
Run schema-locked LLM prompts.

Step 5 — Validation + Conflicts
Apply deterministic rules.

Step 6 — Export / UI
Generate XLSX and Streamlit views.

Step 7 — Evaluation
Maintain gold sets and regression tests.

---

9. Design Rationale (Explicit)

BM25 was chosen over embeddings because:

- legal drafting relies on **exact language**,
- lexical transparency matters for audit,
- semantic similarity introduces uncontrolled recall,
- deterministic ranking is easier to reason about and debug.

This is a deliberate trade-off.