Step 1 — Ingestion & Normalization (Technical Summary)

Objective

Convert heterogeneous legal documents (PDF, DOCX, TXT) into a **single, deterministic, normalized internal representation**, suitable for downstream structural parsing, chunking, and retrieval.

This step establishes the **contract between raw documents and the rest of the pipeline**.

What this step produces

For each input document:

- A **normalized** `Document` **object**
- A list of **ordered pages**, each with:
  - `page_index` (0-based, internal)
  - `text` (raw extracted text)
- A **deterministic** `doc_id`, stable across runs

Output is serialized as **JSONL** for:

- auditability
- replayability
- decoupling from later steps

Core design choices

1. Early normalization, not early intelligence

Step 1 is intentionally *dumb*:

- no semantic understanding

- no section detection
- no chunking
- no cleaning beyond what is strictly necessary

Its job is **not to be clever**, but to be **reliable and predictable**.

---

2. Single internal data model

All formats are normalized into the same structure:

- PDF → many pages
- DOCX → single page
- TXT → single page

This removes format-specific logic from downstream steps.

> From Step 2 onward, the system never cares whether a document came from PDF or DOCX.

---

3. Deterministic document identity

Each document gets a stable `doc_id` derived from:

- its source path
- a deterministic hash

This enables:

- reproducible pipelines
- chunk ID stability
- diffing across runs
- safe re-ingestion

---

4. Minimal but explicit failure handling

- Unsupported file types fail fast
- Empty documents still produce a `Document` object
- No silent skipping

The pipeline prefers **explicit failure** over silent data loss.

What I implemented

- A **loader layer** with one function per format:
  - `load_pdf`
  - `load_docx`
  - `load_txt`
- A single dispatch function: `load_any(path)`
- A normalized data model (`Document`, `Page`)
- A CLI entrypoint that:
  - walks the raw directory
  - ingests all supported files
  - writes `ingest.jsonl` as the canonical output of Step 1

Tricky difficulties encountered

1. PDF text extraction is lossy

- Headers, footers, page numbers, and TOC entries are mixed into text
- Visual structure is lost

Mitigation:

Step 1 does *not* try to fix this. It preserves text as-is and defers interpretation to later steps.

---

2. Page indexing consistency

- PDFs are naturally paginated
- DOCX/TXT are not

Mitigation:

Use a unified concept of “pages”:

- real pages for PDFs
- synthetic page `0` for DOCX/TXT

This keeps provenance logic simple later.

---

3. Avoiding premature cleaning

It was tempting to:

- remove headers
- strip TOC
- normalize typography

Mitigation:

All semantic or structural decisions were deferred to Step 2, where context exists.

Key lessons learned

- **Normalization is a contract.**

  If Step 1 is unstable, everything downstream becomes fragile.

- **Do not interpret too early.**

  Early heuristics almost always fail on edge cases.

- **Traceability matters more than cleanliness.**

  It’s better to carry noisy text forward than to drop potentially important content.

Current limits (by design)

- No semantic understanding
- No section awareness
- No de-duplication across documents
- No OCR correction or layout reconstruction

These are *not* flaws — they are **explicitly out of scope** for Step 1.