AustLii— Technical Handover

Table of Contents

[AustLii— Handover Document 1](#_Toc207034395)

[1. System Overview 1](#_Toc207034396)

[2. Database Schema 2](#_Toc207034397)

[**Tables** 2](#_Toc207034398)

[**Extensions** 2](#_Toc207034399)

[**Triggers, Functions & Indexes** 2](#_Toc207034400)

[3. Embedding Model 3](#_Toc207034401)

[4. Chunking Strategy 3](#_Toc207034402)

[5. Pipeline for Loading Embeddings into Database 3](#_Toc207034403)

[6. Search, Query, and Best Practices 4](#_Toc207034404)

[7. Schema Initialization (deploying to new database) 4](#_Toc207034405)

[8. Useful Code Locations 4](#_Toc207034406)

[9. Example: Typical Ingest & Embedding Usage 4](#_Toc207034407)

# AustLii— Handover Document

This document explains the architecture, database schema, embedding and chunking strategies, and ingestion/loading pipeline for the Austlii RAG system.

## 1. System Overview

AustLiii a research platform for legal documents, designed with semantic search, hybrid/BM25 and FTS capabilities, leveraging large transformer-based vector models. It provides a robust and reproducible vector and metadata management, optimized for legal research use cases.

## 2. Database Schema

All schema and migrations are managed in db/store.py. The schema is auto-initialized on first run and during migrations, and includes all required columns, PostgreSQL extensions, triggers, and indexes. Key elements:

### **Tables**

* **users**: User credentials, with password hash and optional Google login.
* **documents**: Main store for raw documents.
  + id: PK
  + source: String identifier (filepath/url/resource)
  + content: Raw full text
  + format: “case”, “journal”, “legislation”, etc.
  + document\_fts: tsvector column for full-text search (auto-maintained via trigger)
* **embeddings**: Stores one row per chunk, with embedding vector and metadata.
  + id: PK
  + doc\_id: FK to documents
  + chunk\_index: Int, sequence within document
  + vector: vector(768) for pgvector
  + chunk\_metadata: JSONB (includes all extracted metadata, including URLs/dates/citation info)
* **embedding\_sessions**, **embedding\_session\_files**: Track batch ingestion/processing.
* **chat\_sessions**: For conversational/Ai workflows.
* **conversion\_files**: Tracks conversion/import job steps.

### **Extensions**

Automatically enabled (if missing): - vector (pgvector): Required for ANN search - pg\_trgm: Fuzzy/trigram search - "uuid-ossp": UUIDs in tables - fuzzystrmatch: Optional advanced search

### **Triggers, Functions & Indexes**

* document\_fts maintained by trigger on INSERT/UPDATE: uses to\_tsvector('english', content)
* GIN index on document\_fts
* GIN index on content with trigram ops
* IVFFLAT index on embeddings.vector for efficient vector search
* All DDL is idempotently issued on app launch

## 3. Embedding Model

* Default: [nomic-ai/nomic-embed-text-v1.5](https://huggingface.co/nomic-ai/nomic-embed-text-v1.5) (AUSLEGALSEARCH\_EMBED\_MODEL env var overridable)
* Vectors: 768 dimensions
* Context window: Up to ~2048 tokens per chunk
* HuggingFace sentence-transformers loader
* Easily swappable via environment variable or code
* Extensible to other models (MiniLM, BAAI/bge, etc)

Example:

from embedding.embedder import Embedder  
embedder = Embedder() # or Embedder("all-MiniLM-L6-v2")  
vecs = embedder.embed(["example text"])

## 4. Chunking Strategy

Defined in ingest/loader.py:

* **Supported types:** .txt, .html
* **Chunking rules:**
  + Max 1500 characters per chunk
  + Prefer paragraph boundaries, fallback to split by sentence if chunk >1500 chars remains
  + Special handling for:
    - Legislation: Section-wise, then by para
    - Journals: Headings/sections, fallback para
    - Cases/generic: Para, else raw splits
  + Each chunk stores its own metadata block, including title, date, URL, section, etc
* **Metadata block:** Parsed at file head, YAML-like, inserted into chunk\_metadata

## 5. Pipeline for Loading Embeddings into Database

1. **Discovery:** walk\_legal\_files() traverses root directories for .txt/.html files.
2. **Parsing:** Each file parsed (parse\_txt/parse\_html) for body text and optional metadata.
3. **Chunking:** Text chunked as per type and size (1500 chars max/chunk).
4. **Embedding:** Each chunk transformed to a 768-dim vector using Embedder (model as above).
5. **Storing:** Each chunk (text, source, metadata, embedding) is inserted into the embeddings table. Parent document is inserted to documents if not already present (with content, format, FTS field auto-populated).
6. **Full Text Search:** document\_fts is auto-updated by triggers and indexed for fast text queries.
7. **Search:** Hybrid/BM25/FTS queries handled with deduplication at the case/url level, returning unified results for legal research tasks.

## 6. Search, Query, and Best Practices

* **Vectors:** Searched with pgvector (approximate or exact depending on index availability)
* **BM25/Fuzzy:** Content indexed for BM25 and trigram fuzzy search
* **FTS:** tsvector and JSONB full-text logic (with phrase/stemming logic, dedup by URL/case)
* **Smart deduplication:** All search endpoints deduplicate results by case/URL when possible, so the user sees one card per real-world legal entry.

## 7. Schema Initialization (deploying to new database)

All schema setup is **automatic**: - On running the backend or Gradio/Streamlit app, create\_all\_tables() runs, enabling extensions and setting up all indexes, triggers, and functions. - Any missing columns/tables/functions will be created without data loss.

***If you are deploying to a completely new database, simply run the app; all tables, extensions, FTS triggers, and indexes required for smart, fast search will be created.***

## 8. Useful Code Locations

* db/store.py: Full table/ORM definitions, DDL, FTS, and search logic
* embedding/embedder.py: Embedding model selection
* ingest/loader.py: Chunking methods and file pipeline
* fastapi\_app.py, gradio\_app.py: API/frontend, for connecting to backend

## 9. Example: Typical Ingest & Embedding Usage

# Walk files and ingest  
for f in walk\_legal\_files(["/path/to/legal/docs"]):  
 doc = parse\_txt(f) # or parse\_html  
 chunks = chunk\_document(doc)  
 for idx, chunk in enumerate(chunks):  
 vec = Embedder().embed([chunk["text"]])[0] # 768 dim  
 add\_embedding(doc\_id, idx, vec, chunk.get("chunk\_metadata"))