Knowledge Hub — Technical Deep Dive

This document explains every moving part of the Knowledge Hub project, with implementation notes and the math behind the search stack (FTS, embeddings, IVFFlat), OCR pipeline, hybrid ranking, and RAG.

Table of Contents

1. Architecture Overview
2. Data Model & Storage
3. Ingestion Pipeline (PDF/Image → Text Chunks)
4. Rendering & Preprocessing
5. Tesseract OCR (multi-pass) + Confidence
6. Handwriting Fallback (TrOCR)
7. Full-Text Search (FTS) in Postgres
8. Tokenization → `tsvector`
9. Queries → `tsquery`
10. Ranking → `ts_rank` / `ts_rank_cd`
11. Snippets → `ts_headline`
12. Semantic Search with Embeddings (pgvector)
13. Embedding Model & Normalization
14. Distance Metrics (`<->`, `<#>`, `<=>`)
15. IVFFlat Index: lists, probes, complexity & tuning
16. Hybrid Ranking (FTS ⊕ Semantic)
17. Score normalization (z-score)
18. Weighted blending
19. Confidence-aware penalties
20. Retrieval-Augmented Generation (RAG) to LLM (Ollama)
21. Prompt structure & citation discipline
22. Context packing & map-reduce for large docs
23. Performance, Scalability & Ops
24. Security & Privacy
25. Future Upgrades & Research Notes

---

1) Architecture Overview

```
Client (Next.js)
   |
   ├── Upload →  POST /api/documents/upload  (202 Accepted)
   |               └─ Background ingestion → chunks
   |
   ├── Search →   POST /api/search | /search/semantic | /search/hybrid
```

```
    │
    ├── Embed →    POST /api/embeddings/reindex
    │
    └── Answer →   POST /api/answer  (Hybrid retrieval → LLM via Ollama)

Services: Flask API (Gunicorn) • Postgres 16 + pgvector • Storage (local/S3)
Indexes: GIN (FTS) • IVFFlat (vectors)
```

Why this split? - Postgres handles both **keyword search** (FTS) and **semantic vectors** (pgvector) so we keep a single operational store. - Background ingestion avoids request timeouts on OCR-heavy PDFs.

---

2) Data Model & Storage

Key tables - `documents(id, user_id, title, source_path, mime_type, pages, bytes, hash_sha256, status, created_at, updated_at)` - `chunks(id, document_id, version, page_no, chunk_index, text, tokens, modality, bbox, extra_json, created_at)` - `embeddings(id, chunk_id, model, dim, vector)` ⟵ `vector` is a pgvector column - `users`, `tags`, `document_tags`

Notes - `chunk` is the retrieval unit. For scans initially 1 chunk/page; later refined to 300–700 tokens with overlap. - `extra_json` stores attributes like `{ "ocr_conf": 72.5 }`. - **Embedding dimension must equal model output** (e.g., 384 for `all-MiniLM-L6-v2`).

Storage - Originals saved to `storage/…` with SHA-256 for dedup/versioning.

---

3) Ingestion Pipeline (PDF/Image → Text Chunks)

3.1 Rendering & Preprocessing

- **Rendering**: PyMuPDF rasterizes each page at scale `s` (typically `s=3`) to improve readability for OCR.
- **Deskew**: estimate rotation angle via minimum-area rectangle on foreground pixels.
- Given grayscale image I , invert to $\bar{I} = 255 - I$. Let $S = \{(x, y) | \bar{I}(x, y) > 0\}$. Fit `minAreaRect(S)` to get angle θ . Rotate by $-\theta$.
- Rotation uses affine transform: $M = \begin{bmatrix} \alpha & \beta & t_x \\ -\beta & \alpha & t_y \end{bmatrix}$, where $\alpha = \cos\theta,\ \beta = \sin\theta$.
- **Denoise/Sharpen**: non-local means denoising + unsharp mask: $I_{sharp} = a\, I - b\, G_\sigma(I)$ (typical $a = 1.5, b = 0.5$).
- **Binarization**:
- **Otsu**: global threshold $t^* = \arg\max_t\ \sigma_B^2(t)$ maximizing between-class variance.
- **Adaptive**: local thresholds $T(x, y) = \mu_{N(x,y)} - C$ over neighborhoods N .
- **Morphology**: small closing (structuring element 2×2) to connect handwriting strokes.

3.2 OCR with Tesseract (multi-pass) + Confidence

- Run several PSM configs (`--psm 6, 11, 4`) and keep the best by **average word confidence**.
- `image_to_data` returns token confidences $c_i \in [0, 100]$; we compute

$$\mathrm{avg\_conf} = \frac{1}{n} \sum_{i=1}^{n} c_i.$$

- Store text + `ocr_conf` in `chunks.extra_json`.

3.3 Handwriting Fallback (TrOCR)

- Optionally run **TrOCR** (VisionEncoderDecoder) when Tesseract confidence is low.
- Encoder (ViT) maps image to latent sequences; decoder (Transformer LM) generates text via conditional language modeling minimizing cross-entropy:

$$\mathcal{L} = -\sum_{t} \log p(y_t \mid y_{<t}, \mathrm{image}).$$

- Use as a fallback to improve cursive/handwritten pages.

---

4) Full-Text Search (FTS) in Postgres

4.1 Tokenization → `tsvector`

- Postgres parses text, normalizes (lowercase, stemming), removes stopwords, producing a multiset of **lexemes** with positional info.
- Example: `to_tsvector('english', text)` ⇒ `a:1 b:2,7 c:4`.

4.2 Queries → `tsquery`

- `plainto_tsquery('english', q)` for robust plain queries; `websearch_to_tsquery` supports Google-like syntax (`"phrase"`, `-exclude`, `OR`).

4.3 Ranking → `ts_rank`, `ts_rank_cd`

- `ts_rank` ranks by term frequency with optional **weights** per lexeme class (A/B/C/D).
- `ts_rank_cd` (cover density) favors **tight spans** covering more query terms with fewer gaps. Conceptually it scores compact coverage windows; exact formula considers positional distances and normalizes by document length.
- We index with **GIN** on `to_tsvector('english', coalesce(text,''))` for sub-second search over large corpora.

4.4 Snippets → `ts_headline`

- Generates fragments with query terms emphasized; parameters control fragments, min/max words, and tags (e.g., `<b>…</b>`).

---

5) Semantic Search with Embeddings (pgvector)

5.1 Embedding Model & Normalization

- We use `sentence-transformers/all-MiniLM-L6-v2` to encode chunks and queries into dense vectors $x \in \mathbb{R}^d$ ($d = 384$ for this model).
- We **L2-normalize** vectors so $\|x\|_2 = 1$. This makes cosine similarity equal to the inner product: $\cos(\theta) = x \cdot y$.

5.2 Distance Metrics in pgvector

- **L2** distance `<->`: $\|x - y\|_2$.
- **Inner product** `<#>`: $-x \cdot y$ `distance` (smaller is better when normalized).
- **Cosine** `<=>`: $1 - \cos(\theta)$. With normalized vectors, `<=>` equals $1 - x \cdot y$.

Operator class chosen at index time must match your metric: - `vector_l2_ops` for `<->` - `vector_ip_ops` for `<#>` - `vector_cosine_ops` for `<=>`

5.3 IVFFlat Index (Approximate Nearest Neighbor)

- **Idea**: Coarse quantization partitions the space into `lists` buckets using k-means centroids $\{\mu_j\}_{j=1}^{L}$. Each vector is assigned to its nearest centroid: $\arg\min_j d(x, \mu_j)$.
- **Build**: choose `lists = L` (e.g., 100–1000). pgvector trains centroids; each list stores **full vectors** ("Flat").
- **Search**: probe `probes = P` nearest centroids to the query; scan only those lists. Complexity ~ $O(P \cdot N/L)$ vs $O(N)$ exact.
- **Recall/Latency trade-off**: higher `lists` and `probes` → better recall, more CPU.
- **Tuning**: start with `lists ≈ 4·√N` (rule of thumb) and `probes ≈ 5–10% of lists`, then adjust by latency.
- **Planner stats**: run `ANALYZE embeddings;` after bulk inserts; optionally set `ivfflat.probes = P` per session.

---

6) Hybrid Ranking (FTS ⊕ Semantic)

We combine both signals to get precision **and** recall.

1. Retrieve top-K_sem via cosine distance; convert to similarity $s_v = 1 - d_{cos}$.
2. Retrieve top-K_fts via FTS rank $s_f = \text{ts\_rank\_cd}$.
3. **Normalize** each stream using z-scores:

$$z_v = \frac{s_v - \mu_v}{\sigma_v}, \quad z_f = \frac{s_f - \mu_f}{\sigma_f}.$$

4. **Blend** with weights α, β (e.g., $\alpha = 0.6,\ \beta = 0.4$):

$$\text{score} = \alpha\, z_v + \beta\, z_f.$$

5. **Confidence penalty** (optional): if a chunk has $\mathrm{ocr\_conf} < 50$, subtract a small λ from the score.

This yields stable, interpretable ranking and allows tuning α, β, λ from click data.

---

7) RAG (Retrieval-Augmented Generation) with Ollama LLM

Goal: compose an answer **only** from retrieved chunks, with citations.

7.1 Prompt Structure

- **System**: “Answer using only CONTEXT. If insufficient, say so. Add `[CIT-#]` after claims.”
- **Context**: top chunks, deduped by (doc,page), trimmed to ~500–800 chars, labeled `[CIT-1]…`.
- **User**: original question.

7.2 Context Packing

- Sort by hybrid score; keep early context highest quality (models attend more to earlier tokens).
- Cap total context tokens (e.g., 3k–4k). If overflow, switch to **map-reduce**: ask for short per-chunk notes (map), then summarize those notes (reduce), preserving citations.

7.3 Citation Enforcement

- Post-process to extract `[CIT-#]` tags and map back to (document_id, page_no, title).
- If no citations in output, optionally re-prompt with stricter instruction.

---

8) Performance, Scalability & Ops

- **Ingestion**: process in background; commit every N pages to avoid large transactions; record per-page errors and continue.
- **FTS**: GIN index on `to_tsvector(..)` keeps queries sub-second; vacuum/analyze periodically.
- **Vectors**: IVFFlat with cosine ops; tune `lists` and `probes` as corpus grows; batch embeddings (64–256 rows) to limit RAM.
- **Timeouts**: avoid long work in request handlers; background tasks via threads initially, queue later.
- **Monitoring**: log durations for render, OCR, chunk, embed; track retrieval latency, LLM latency, total.

---

9) Security & Privacy

- Local-first by default; originals never leave the machine unless configured.
- Hash files (SHA-256) for dedup/versioning; optional encryption at rest.
- AuthZ per `user_id` for multi-user mode; sanitize any HTML rendering (only allow `<b>` in snippets).

10) Future Upgrades & Research Notes

- **Text-first extraction**: prefer embedded text over OCR; OCR only when needed.
- **Smarter chunking**: 300–700 tokens, overlap 50 tokens, `heading_path` to preserve outline.
- **Tables & Figures**: detect and store `table_json` + auto summaries; caption figures for discoverability.
- **Math/Code**: preserve tokens like `O(n log n)` and code identifiers for precise FTS.
- **Hybrid tuning**: learn α, β from labeled data; consider BM25 for lexical score.
- **ANN variants**: HNSW or PQ-IVF if ultra-large scale; evaluate recall/latency curves.
- **Queue & retries**: switch background threads to Redis/RQ or Celery with a `/tasks/:id` status API.

Quick Reference (Cheat Sheet)

- Cosine similarity: $\cos(\theta) = \frac{x \cdot y}{\|x\| \, \|y\|}$; cosine distance `<=>` = $1 - \cos(\theta)$.
- Z-score: $z = (x - \mu)/\sigma$.
- Otsu threshold: maximize between-class variance σ_B^2 .
- IVFFlat: `lists` =coarse clusters; `probes` =centroids searched; complexity ~ $O(P \cdot N/L)$.
- FTS cover density: prefers compact spans covering many query terms.
- Embedding dim must match model output (e.g., 384 for MiniLM-L6-v2).