

Data

Corpus pipeline and gates

🕒 4 min read · Updated 25 September 2026 · Verified against fe1ff7d on 2026-09-25

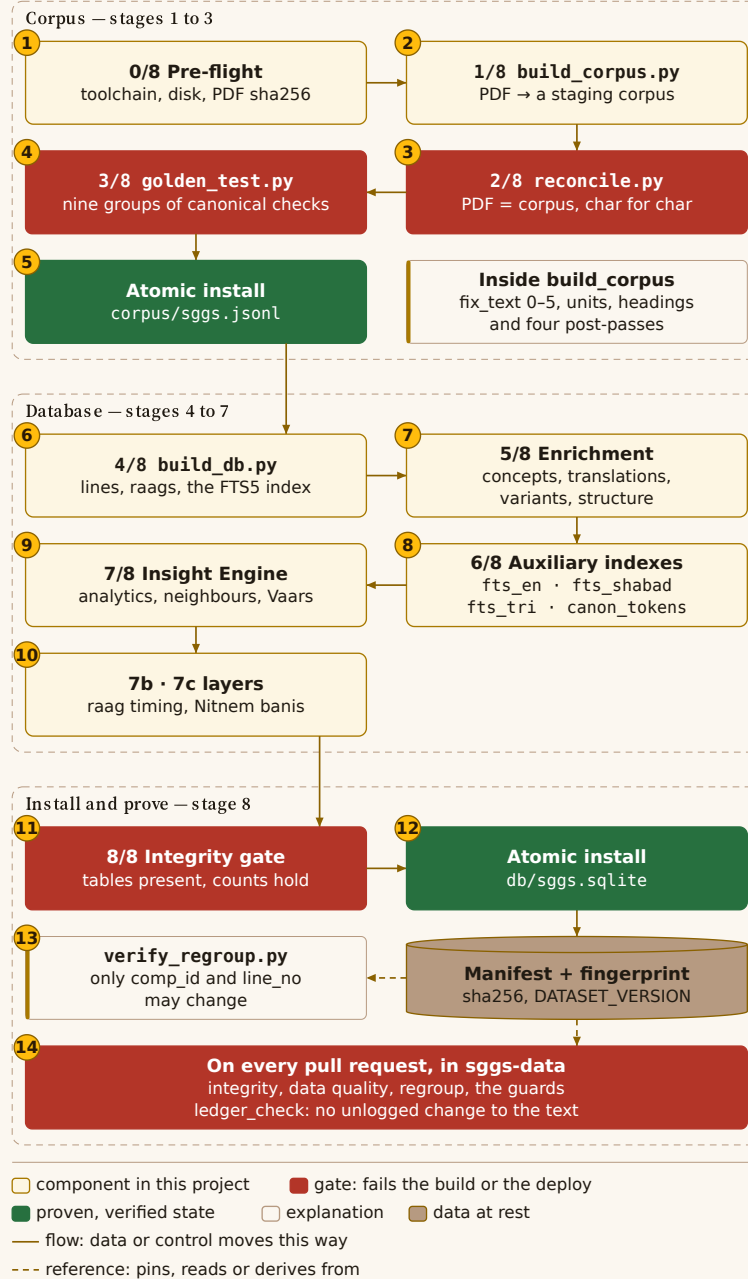
bash pipeline/rebuild_all.sh in sggs-data, stage by stage: extract, reconcile character for character, golden checks, atomic installs, the database layers, the integrity gate; code read at the pin.

<https://docs.gurbanisoul.com/data/pipeline/>

The rebuild lives in the data repository and is one command: `bash pipeline/rebuild_all.sh`. Every stage gates on its verifier and every install is atomic, so the committed corpus and database are replaced only after their checks pass and are never left half-written. The poster walks the stages in the order the script runs them; the excerpts below are read from sggs-data at the commit this wiki pins, never pasted.

04 · Corpus pipeline and gates

bash pipeline/rebuild_all.sh — from the attested source PDF to a fingerprinted database, every stage gated, every install atomic



Source of truth: sggs-data/pipeline/rebuild_all.sh ·
 sggs-data/pipeline/build_corpus.py · sggs-data/pipeline/reconcile.py ·
 sggs-data/pipeline/golden_test.py
 v1.3.10 · verified 2026-09-27 · d85c2ad
 Sri Guru Granth Sahib Ji — Knowledge Base · engineering wiki

Poster 04 — Corpus pipeline and gates: pre-flight, extraction, reconcile, golden suite, atomic installs, the database layers, the integrity gate, fingerprint, and the CI gates on every pull request

1. Pre-flight: the exact inputs. The rebuild refuses to start unless there is free disk, the interpreter has PyMuPDF, numpy and scipy, and the source PDF's sha256 equals the attestation recorded with the committed corpus. The translation sources and the ShabadOS database must be byte-identical to their checksum lists. SOURCE_DATE_EPOCH is the commit time, so the same commit, PDF and toolchain give the same bytes — the rebuild is reproducible.

2. **1/8 Extract the corpus.** `build_corpus.py` reads every page: `page_content` strips borders and joins wrapped lines; `fix_text` applies the registered decoding steps 0 to 5 (print timestamps, halant order, font-glyph restorations, sihari order, the four editorial rules, stray characters); `segment_units` cuts the stream on the double danda; `detect_header` reads the metadata from headings; four post-passes fix page-break numerals, fold each heading run into its composition, let an invocation adopt its section and refine Bhatt attribution. Everything is written to a staging file.
3. **2/8 Reconcile: character for character.** `reconcile.py` rebuilds the PDF's text stream through the same `page_content` and `fix_text`, joins the corpus rows in id order, drops the spaces on both sides, and requires the two streams to be identical. Nothing lost, nothing duplicated, nothing reordered — or the rebuild aborts and the committed corpus stays as it was.
4. **3/8 The golden suite.** `golden_test.py` runs nine groups of canonical checks: the unit corrections on raw words, the page-to-Ang map, the Mool Mantar character for character, transliteration and first-letter spot checks, rahao detection, header detection (including verses that must not be headings), Angs 1429–1430, and raag-start pages where the title and the invocation must be separate, correctly placed units.
5. **Install the corpus atomically.** Only now does `atomic_install.py` replace `corpus/sggs.jsonl`: the new file is written beside the old one, fsynced and renamed in one step, so a failed or interrupted rebuild never leaves a truncated corpus behind.
6. **4/8 Build the database.** `build_db.py` creates the lines table (23 columns), its indexes, the raags, sections and authors aggregates, the FTS5 index over the six search columns and the meta table, in a temporary location.
7. **5/8 Enrichment.** Concept tags, the English translations (ShabadOS and Dr. Sant Singh Khalsa, loaded as a separate table), the phonetic variant index and `enrich_v2`'s structural columns (`stanza_index`, `pada_total`, `source_category`) are added. None of these touches a text column.
8. **6/8 Auxiliary indexes.** `fts_en` for English, `fts_shabad` for cross-line passages, `fts_tri` (a trigram index — built, though no API route reads it today) and `canon_tokens`. The trigram tokenizer is a build contract: an SQLite without it must fail the build, never ship a database that silently lacks it.
9. **7/8 The Insight Engine.** Theme network, fingerprints, author and raag analytics, semantic neighbours (exact sparse tf-idf cosine), author resonance and the 22 Vaars, detected by their title headers. Stages 4 to 7 only ever add tables: the scripture written at stage 1 is never altered after it.
10. **7b • 7c The knowledge layers.** The raag-timing knowledge layer and the Nitnem bani registry (from the pinned ShabadOS database) add their own new tables; `guard_banis` checks the registry. Scripture stays untouched.
11. **8/8 The integrity gate.** `db_integrity_gate.py` refuses a database that lacks a declared table or breaks a counted invariant, before anything is installed.
12. **Install, stamp, fingerprint.** `atomic_install.py` replaces `db/sggs.sqlite`. `MANIFEST.json` records the database's sha256 and `DATASET_VERSION`; `sggs_integrity.py` writes the per-table content fingerprint to `audit/dataset-fingerprint.json`. Both are committed together with the database.
13. **Beside it: verify_regroup.** When a rebuild changes how lines are grouped, `verify_regroup.py` compares the old and new databases and allows only `comp_id` and `line_no` to differ: every Gurmukhi byte must be identical.
14. **On every pull request.** `sggs-data`'s CI re-proves the committed data: `sggs_integrity` against the fingerprint, `data_quality` against its reviewed baseline, `verify_regroup`'s invariants, `guard_scripture` (pre-existing tables byte-identical), `guard_banis`, the timing tests, and `ledger_check` — a scripture change or a new `fix_text` rule that has no entry in the editorial ledger fails the build.

The stages

Stage	Script	What it does	What stops it
0/8 pre-flight	<code>rebuild_all.sh</code>	free disk, PyMuPDF + numpy + scipy, the source PDF's sha256 equals the attestation, translation and ShabadOS inputs match their checksum lists, <code>SOURCE_DATE_EPOCH</code> = commit time	any mismatch; a new PDF edition needs <code>SGGS_ALLOW_NEW_PDF=1</code> and review
1/8 extract	<code>build_corpus.py</code>	page → Ang, <code>fix_text</code> steps 0–5, unit segmentation, header detection, four post-passes, written to a staging file	— (the committed corpus is untouched)
2/8 reconcile	<code>reconcile.py</code>	the PDF's text stream and the corpus stream, spaces dropped, must be identical	one character of difference
3/8 golden suite	<code>golden_test.py</code>	nine groups of canonical checks	any failed check
install corpus	<code>atomic_install.py</code>	temp file beside the target, fsync, rename	sha mismatch
4/8 database	<code>build_db.py</code>	<code>lines</code> (23 columns), indexes, <code>raags/sections/authors</code> , <code>fts</code> (FTS5), <code>meta</code>	—
5/8 enrichment	<code>load_translations.py</code> , <code>build_variants.py</code> , <code>enrich_orchestrator.py</code> , <code>enrich_v2.py</code>	concepts, translations as their own table, phonetic variants, structural columns	—
6/8 auxiliary indexes	inline SQL	<code>fts_en</code> , <code>fts_shabad</code> , <code>fts_tri</code> , <code>canon_tokens</code>	an SQLite without the trigram tokenizer (a search-tier contract)
7/8 Insight Engine	<code>ml_analytics_builder.py</code> , <code>build_semantic_vectors_lite.py</code> , <code>build_resonance.py</code> , <code>build_vaars.py</code>	analytics, semantic neighbours, resonance, the 22 Vaars — additive tables	—
7b·7c layers	<code>pipeline/timing/*</code> , <code>pipeline/banis/*</code>	raag-timing knowledge layer, Nitnem bani registry; <code>guard_banis.py</code>	the bani guard
8/8 gate + install	<code>db_integrity_gate.py</code> , <code>atomic_install.py</code> , <code>sggs_integrity.py</code>	every declared table present and counted; atomic install; <code>MANIFEST.json</code> ; <code>audit/dataset-fingerprint.json</code>	the integrity gate

Extraction, read from the source

`fix_text` is the only place the printed glyph stream is transformed into logical Unicode. Steps 0 to 3 and 5 are decoding rules (timestamps, halant order, glyph restorations, sihari order, stray characters); step 4 is the four registered editorial rules. Every rule is an entry in the editorial ledger, and `ledger_check.py` fails a pull request whose code and ledger disagree.

sggs-data/pipeline/sggs_pipeline.py · fix_text · lines 23–95

```

def fix_text(s):
    """Apply the proven encoding corrections. Returns (fixed, anomalies)."""
    anomalies = []
    # 0. strip PDF print-job timestamps (HH:MM:SS, ASCII digits – never Gurbani)
    for m in re.finditer(r'\d{1,3}:\d{2}:\d{2}', s):
        anomalies.append(('timestamp_stripped', m.group()))
    s = re.sub(r'\s*\d{1,3}:\d{2}:\d{2}\s*', ' ', s)
    # 1. swap (consonant, halant) -> (halant, consonant) e.g. ਪ੍ਰਭ -> ਪ੍ਰਭ
    out = []
    i = 0
    while i < len(s):
        c = s[i]
        if c in CONS and i + 1 < len(s) and s[i+1] == HALANT:
            out.append(HALANT); out.append(c); i += 2
        else:
            out.append(c); i += 1
    s = ''.join(out)
    # 2. font-glyph restorations (proven by word evidence; see
    docs/reports/archive/Validation-Report.md)
    for a, b in [(('Â', 'ਾ'), # kanna+bindi ਸਾੰਤਿ -> ਸਾਂਤਿ
                  ('.', 'ੁਹ'), # pairin-haha ਭਿਤਨੁ -> ਭਿਤਨੁ
                  ('±', 'ੁਹ'), ('ੌ', 'ੁ'), ('¼', 'ੁਹ'), ('¾', 'ੁਹ'), # ਕੇੜੜੌ ਪੜੜ ਕਾਨੜ ਚੜੜੌ
                  ('ੌ', 'ੁਵ'), # pairin-vava ਸਾੰਮੀ -> ਸ਼ਾਮੀ
                  ('0', 'ਾ'), ('□', 'ਾ'), # bindi ਨੀਠੁ ਨੀਠੁਦ -> ਨੀਂ...
                  ('□', 'ਾ'), # hora ਗੁਰਿਬੰਦ -> ਗੁਰਿਬੰਦ
                  ('□', 'ਾ')]: # kanna (rare Sahaskriti ligature; flagged)
        if a in s:
            if a == '□': anomalies.append(('f02d_kanna', s[:60]))
            s = s.replace(a, b)
    s = s.replace('ੴ', IK_ONKAR) # Ik Onkar restoration
    # 2b. re-attach vowel signs split off by a stray space (archaic ਉ spellings:
    # ਸਫਲਿਉ -> ਸਫਲਿਉ, ਗਾਉ -> ਗਾਉ – 12 cases corpus-wide, all logged)
    for m in re.finditer(r'(\S) ([ੌ])(?=\S)', s):
        anomalies.append(('vowel_reattached', m.group(1) + m.group(2)))
    s = re.sub(r'(\S) ([ੌ])(?=\S)', r'\1\2', s)
    # 3. move sihari after its consonant cluster (ਕਿਪਾ -> ਕ੍ਰਿਪਾ)
    out = []
    i = 0
    n = len(s)
    while i < n:
        c = s[i]
        if c == 'ਿ':
            j = i + 1
            if j < n and s[j] in CONS:
                j += 1
                while j + 1 < n and s[j] == HALANT and s[j+1] in CONS:
                    j += 2
                out.append(s[i+1:j]); out.append('ਿ')
                i = j
            else:
                anomalies.append(('orphan_sihari', s[max(0,i-6):i+6]))
                out.append(c); i += 1
        else:
            out.append(c); i += 1
    s = ''.join(out)
    # 4. documented editorial corrections (impossible Unicode produced by the
    # font's glyph order; restored to the canonical reading -- see
    docs/reports/archive/Validation-Report.md)

```

```

    for a, b in [('\u0a4d\u0a30\u0a3f\u0a3f\u0a4d\u0a38\u0a1f',
'\u0a4d\u0a30\u0a3f\u0a38\u0a1f\u0a3f'),
                ('\u0a4d\u0a38\u0a28\u0a47\u0a39', '\u0a38\u0a4d\u0a28\u0a47\u0a39'),
                ('\u0a3f\u25cc \u0a2e\u0a4d\u0a2f\u0a4d\u0a2f\u0a3e\u0a28\u0a47',
'\u0a2e\u0a4d\u0a2f\u0a4d\u0a2f\u0a3f\u0a3e\u0a28\u0a47')]:
        if a in s:
            anomalies.append(('editorial_fix', a + ' -> ' + b))
            s = s.replace(a, b)
        for m in re.finditer('\u0a3f\u0a4d([\u0a38\u0a2a])\u0a24', s):
            anomalies.append(('editorial_fix', 'sahaskriti ' + m.group(0) + ' -> ' + m.group(1)
+ '\u0a4d\u0a24\u0a3f'))
            s = re.sub('\u0a3f\u0a4d([\u0a38\u0a2a])\u0a24', '\\1\u0a4d\u0a24\u0a3f', s)
        # 5. residual private-use / stray latin/ascii -> flag + strip
        stray = re.compile('[!~\u00a0-\u00ff\u25cc\ue000-\uf8ff]')
        for m in stray.finditer(s):
            anomalies.append(('stray_char', hex(ord(m.group())) + ' @ ' +
s[max(0,m.start()-8):m.start()+8]))
            s = stray.sub('', s)
            s = unicodedata.normalize('NFC', s)
            s = re.sub('[ \t]+', ' ', s).strip()
        return s, anomalies

```

Read from the source at build time — [view these lines on GitHub at c36e317](#) .

Source: `pipeline/siggs_pipeline.py · fix_text`² in siggs-data.

segment_units cuts the page stream on the double danda, keeps the numeral markers with the unit they close and sets the rahao flag; detect_header then reads the metadata from a heading, and refuses a verse that merely contains a raag or form word:

sggs-data/pipeline/sggs_pipeline.py · segment_units · lines 214-242

```

def segment_units(stream):
    """Split a ||-delimited stream into units.
    -> list of dicts {text, markers:[..], rahao:bool}, plus trailing fragment."""
    parts = stream.split(DDANDA)
    units = []
    i = 0
    pending_lead = False
    while i < len(parts) - 1:          # parts[-1] = text after last || (fragment)
        text = parts[i].strip()
        i += 1
        markers, rahao = [], False
        while i < len(parts) - 1:
            nxt = parts[i].strip()
            if nxt and all(ch in GDIGITS for ch in nxt):
                markers.append(nxt); i += 1
            elif nxt and RAHAO_RE.fullmatch(nxt):
                rahao = True
                markers.append(nxt); i += 1
            else:
                break
        if text or markers or rahao:
            u = {'text': text, 'markers': markers, 'rahao':rahao}
            if pending_lead:          # title enclosed in dandas, e.g. || ਜਪੁ ||
                u['lead_danda'] = True; pending_lead = False
            units.append(u)
        else:
            pending_lead = True      # empty segment = two adjacent ||
    fragment = parts[-1].strip()
    return units, fragment

```

Read from the source at build time — [view these lines on GitHub at c36e317](#) .

Source: [pipeline/sggs_pipeline.py · segment_units²](#) in sggs-data.

The proof: reconcile

The whole of reconcile.py fits on a screen. It rebuilds the PDF's text through the same page_content and fix_text, joins the corpus rows in id order, drops spaces on both sides and demands equality — nothing lost, nothing duplicated, nothing reordered:

sggs-data/pipeline/reconcile.py · lines 1-49

```
# -*- coding: utf-8 -*-
"""Audit W1: prove the corpus contains EXACTLY the PDF's scripture text –
no loss, no duplication, no reordering – by char-stream equality.

PDF side : pages 54..1483 -> page_content -> fix_text -> join
Corpus side: rows in id order -> gurmukhi display form -> join
Both sides then drop spaces; streams must be identical.
"""
import sys, json, fitz
sys.path.insert(0, __file__.rsplit('/', 1)[0])
from sggs_pipeline import fix_text, page_content

PDF, JSONL = sys.argv[1], sys.argv[2]
doc = fitz.open(PDF)

pdf_parts = []
for p in range(53, 1483):
    _, raw = page_content(doc, p)
    fixed, _ = fix_text(raw)
    pdf_parts.append(fixed)
pdf_stream = ''.join(pdf_parts).replace(' ', '')

rows = [json.loads(l) for l in open(JSONL, encoding='utf-8')]
corpus_stream = ''.join(r['gurmukhi'] for r in rows).replace(' ', '')

print(f'pdf chars : {len(pdf_stream):,}')
print(f'corpus chars: {len(corpus_stream):,}')
if pdf_stream == corpus_stream:
    print('RECONCILED: corpus == source, character for character.')
    sys.exit(0)

# locate first divergence
lo, hi = 0, min(len(pdf_stream), len(corpus_stream))
while lo < hi:
    mid = (lo + hi) // 2
    if pdf_stream[:mid] == corpus_stream[:mid]: lo = mid + 1
    else: hi = mid
i = max(0, lo - 1)
print(f'FIRST DIVERGENCE at char {i:,}')
print('pdf :', repr(pdf_stream[max(0,i-30):i+40]))
print('corpus:', repr(corpus_stream[max(0,i-30):i+40]))
# locate the ang: walk corpus rows accumulating lengths
acc = 0
for r in rows:
    acc += len(r['gurmukhi'].replace(' ', ''))
    if acc >= i:
        print(f'near corpus id={r["id"]} ang={r["ang"]}: {r["gurmukhi"][:70]}')
        break
sys.exit(1)
```

Read from the source at build time — [view these lines on GitHub at c36e317](#) .

Source: [pipeline/reconcile.py](#)⁵ in sggs-data.

The PDF never enters CI. validation/reconcile-attestation.json records that the corpus in a commit reconciled character-exact locally, with the PDF's sha256; the pre-flight refuses any other file.

Atomic installs

Both the corpus and the database are installed by `atomic_install.py`: the new file is written to a temporary name in the destination directory, fsynced, optionally checked against an expected sha256, then renamed over the old one in a single step.

sggs-data/pipeline/atomic_install.py · install · lines 41–68

```
def install(src, dest, expect_sha256=None):
    """Atomically replace dest with a verified copy of src; return its sha256."""
    src_sha = sha256_file(src)
    if expect_sha256 and src_sha != expect_sha256.lower():
        raise ValueError(f'{src}: sha256 {src_sha} != expected {expect_sha256}')
    dest_dir = os.path.dirname(os.path.abspath(dest))
    need = os.path.getsize(src)
    free = shutil.disk_usage(dest_dir).free
    if free < need * 2: # temp copy + headroom; refuse rather than risk ENOSPC mid-write
        raise OSError(f'not enough free space in {dest_dir}: {free} bytes free, need {need * 2}')
    fd, tmp = tempfile.mkstemp(prefix='.' + os.path.basename(dest) + '.', suffix='.tmp',
    dir=dest_dir)
    try:
        with os.fdopen(fd, 'wb') as out, open(src, 'rb') as inp:
            shutil.copyfileobj(inp, out, CHUNK)
            out.flush()
            os.fsync(out.fileno())
        tmp_sha = sha256_file(tmp)
        if tmp_sha != src_sha:
            raise IOError(f'copy verification failed: {tmp_sha} != {src_sha}')
        if os.path.exists(dest):
            shutil.copypode(dest, tmp)
        os.replace(tmp, dest)
        _fsync_dir(dest_dir)
    except BaseException:
        if os.path.exists(tmp):
            os.unlink(tmp)
        raise
    return src_sha
```

Read from the source at build time — [view these lines on GitHub at c36e317](#) .

Source: [pipeline/atomic_install.py · install](#)⁷ in sggs-data.

Beside the rebuild

- **verify_regroup.py** compares an old and a new database and allows only `comp_id` and `line_no` to differ — every *Gurmukhi* byte must be identical. Any pull request in sggs-data that touches the corpus or the database runs it; --invariants re-checks the grouping invariants of the committed database alone.
- **On every pull request** the data repository's `data-ci.yml` re-proves the committed data: `sggs_integrity.py` against `audit/dataset-fingerprint.json`, `data_quality.py` against its reviewed baseline, `verify_regroup.py` --invariants, `guard_scripture.py` (pre-existing tables byte-identical), `guard_banis.py`, the timing-layer tests and `ledger_check.py`.

- **Reproducibility.** Every date the build writes derives from `SOURCE_DATE_EPOCH` (the commit time), so the same commit, PDF and pinned toolchain produce the same bytes; `line_neighbors` is the one table whose *row order* varies, which is why the fingerprint compares content, not bytes.

The operator's view — interpreter trap, re-baselining the timing layer, what to commit — is the canonical **rebuild runbook**. This platform never runs the rebuild; it receives the result as a **pin**.

Links

1. https://github.com/Algorythmos-AI/siggs-data/blob/c36e3172bdf38a61070849bb604b38da89ca5d8b/pipeline/siggs_pipeline.py#L23-L95
2. https://github.com/Algorythmos-AI/siggs-data/blob/main/pipeline/siggs_pipeline.py
3. https://github.com/Algorythmos-AI/siggs-data/blob/c36e3172bdf38a61070849bb604b38da89ca5d8b/pipeline/siggs_pipeline.py#L214-L242
4. <https://github.com/Algorythmos-AI/siggs-data/blob/c36e3172bdf38a61070849bb604b38da89ca5d8b/pipeline/reconcile.py#L1-L49>
5. <https://github.com/Algorythmos-AI/siggs-data/blob/main/pipeline/reconcile.py>
6. https://github.com/Algorythmos-AI/siggs-data/blob/c36e3172bdf38a61070849bb604b38da89ca5d8b/pipeline/atomic_install.py#L41-L68
7. https://github.com/Algorythmos-AI/siggs-data/blob/main/pipeline/atomic_install.py

Sri Guru Granth Sahib Ji — Knowledge Base wiki. Generated from commit c3d9360. Updated 25 September 2026. Verified against felff7d on 2026-09-25. This copy may be out of date; the page is at <https://docs.gurbanisoul.com/data/pipeline/>