

Search & verification

The Roman fold

🕒 3 min read · Updated 25 September 2026 · Verified against 7a343c6 on 2026-09-25

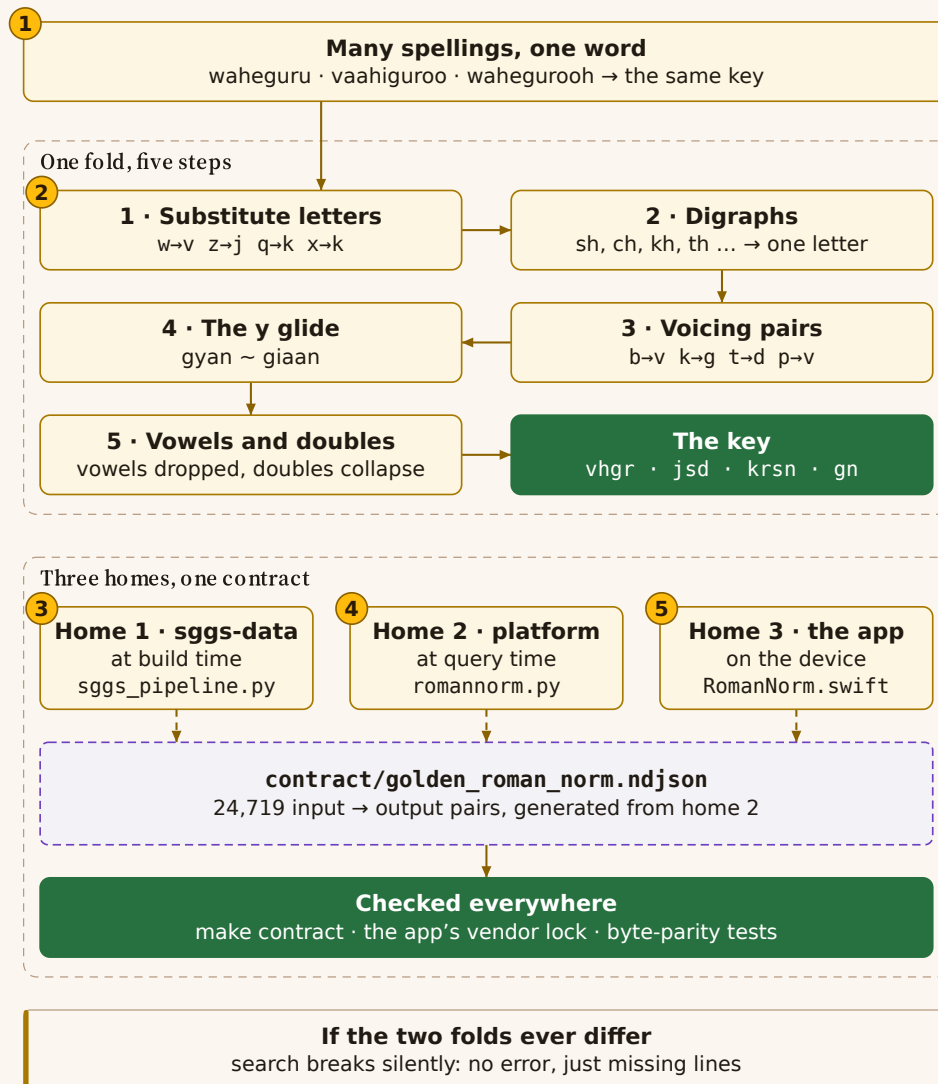
roman_norm: the five-step phonetic fold that lets waheguru and vaahiguroo meet at one key, why it must be byte-identical in three repositories, and the 24,719 golden vectors that hold it there.

<https://docs.gurbanisoul.com/search/the-roman-fold/>

Seekers spell a Gurmukhi word in Roman letters the way they hear it. The fold makes every spelling of a word land on one key — and it must be exactly the same fold wherever it runs, or the key the query produces will not match the key in the index and lines go missing without any error.

06 · The fold in three places

roman_norm turns many spellings into one key — and it lives in three repositories that 24,719 golden vectors hold byte-identical



□ component in this project ■ proven, verified state

□ pin: a reviewed lock file □ explanation

— flow: data or control moves this way

--- reference: pins, reads or derives from

Source of truth: webapp/romannorm.py · contract/golden_roman_norm.ndjson
· tools/gen_golden_vectors.py · contract/_meta.json

v1.3.10 · verified 2026-09-27 · d85c2ad

Sri Guru Granth Sahib Ji — Knowledge Base · engineering wiki

Poster 06 — The fold in three places: the five steps of roman_norm, its three homes, and the golden vectors that bind them

1. The problem. Seekers type a Gurmukhi word in Roman letters the way they hear it: waheguru, vaahiguroo, wahegurooh. A search that demanded one spelling would miss most of them. The fold makes every spelling of a word land on one key — and it must be exactly the same fold wherever it runs, or the query key will not match the index key and lines silently go missing.

2. **Five steps to one key.** Lower-case each word. Substitute $w \rightarrow v$, $z \rightarrow j$, $q \rightarrow k$, $x \rightarrow k$. Reduce the digraphs (sh, chh, ch, kh, gh, jh, th, dh, bh, ph, rh) to their first letter and f to p. Merge the voicing pairs $b \rightarrow v$, $k \rightarrow g$, $t \rightarrow d$, $p \rightarrow v$. Turn a leading y into j and drop a medial y. Keep a leading vowel, drop every other vowel, collapse doubled letters. waheguru becomes vhgr; yashoda and jasodaa become jsd; gyan and giaan become gn.
3. **Home 1: the data repository, at build time.** sggs-data's pipeline applies roman_norm to every line's transliteration and stores the result in the translit_norm column of the lines table. That column is what the fold tier and the verification engine search.
4. **Home 2: the platform, at query time.** webapp/romannorm.py is the one copy in this repository: serve.py re-exports it and verify.py imports it (verify cannot import serve, which imports verify). Search tier 11 folds the query with it; the verification engine folds a Roman claim with it before scoring.
5. **Home 3, and the contract that binds all three.** The iOS app carries a Swift port, RomanNorm.swift. tools/gen_golden_vectors.py drives the platform's function and records 24,719 input and output pairs in contract/golden_roman_norm.ndjson; make contract fails here if the file drifts, the app pins the file by sha256 in vendor.lock.json and runs byte-parity tests, and the data repository's copy is held to the same vectors. Three copies, one truth.

The function, read from the source

webapp/romannorm.py · roman_norm · lines 11–28

```
def roman_norm(s):
    """Phonetic-fold Roman normal form, applied to BOTH the index column and the
    query: 'waheguru'/'vaahiguroo' -> 'vhgr'; 'yashoda'/'jasodaa' -> 'jsd';
    'krishna'/'krisan' -> 'krsn'; 'gyan'/'giaan' -> 'gn'.
    KEEP IN SYNC with sggs-data pipeline/sggs_pipeline.py:roman_norm()."""
    out = []
    for w in s.lower().split():
        w = w.replace('w', 'v').replace('z', 'j').replace('q', 'k').replace('x', 'k')
        for dg in ('sh', 'chh', 'ch', 'kh', 'gh', 'jh', 'th', 'dh', 'bh', 'ph', 'rh', 'f'):
            w = w.replace(dg, dg[0] if dg != 'f' else 'p')
        w = w.replace('b', 'v').replace('k', 'g').replace('t', 'd').replace('p', 'v') # ष/
        # + Sanskrit→Punjabi voicing (bhakti~bhagatee)
        if w.startswith('y'): w = 'j' + w[1:]
        w = w.replace('y', '') # medial glide: gyan ~ giaan
        head = w[0] if w and w[0] in 'aeiou' else ''
        body = re.sub('[aeiou]', '', w)
        body = re.sub(r'(\.)\1+', r'\1', body) # collapse doubles
        out.append((head + body) if (head + body) else w)
    return ' '.join(o for o in out if o)
```

Read from the source at build time — [view these lines on GitHub at c3d9360](#).

Source: [webapp/romannorm.py](#)², the platform's one copy of the fold.

In words, for each lower-cased word:

1. **Substitute letters** — $w \rightarrow v$, $z \rightarrow j$, $q \rightarrow k$, $x \rightarrow k$.
2. **Reduce digraphs** — sh, chh, ch, kh, gh, jh, th, dh, bh, ph, rh become their first letter; f becomes p.
3. **Merge voicing pairs** — $b \rightarrow v$, $k \rightarrow g$, $t \rightarrow d$, $p \rightarrow v$ (

ष
/

₹

, and Sanskrit ↔ Punjabi voicing: *bhakti* ~ *bhagatee*).4. **The y glide** — a leading y becomes j; a medial y is dropped (*gyan* ~ *giaan*).5. **Vowels and doubles** — a leading vowel is kept, every other vowel is dropped, doubled letters collapse.

So *waheguru*, *vaahiguroo* → *vhgr*; *yashoda*, *jasodaa* → *jsd*; *krishna*, *krisan* → *krsn*; *gyan*, *giaan* → *gn*. The fold is lossy on purpose: it trades precision for recall, which is why the waterfall tries it only after the exact tiers, and why the verification engine uses it to *find candidates*, never to decide a verdict on its own.

Three homes

Where	File	When it runs
sggs-data	pipeline/sggs_pipeline.py:roman_norm	at build time, writing <code>lines.translit_norm</code>
sggs-platform	webapp/romannorm.py (re-exported by <code>serve.py</code> , imported by <code>verify.py</code>)	at query time: search tier 11, the verification engine
gurbani-soul-ios	GurbaniSearchKit/Sources/GurbaniSearchKit/RomanNorm.swift	on the device: search and verification

`verify.py` cannot import `serve.py` (`serve` imports `verify`), which is why the fold lives in its own module; there is exactly one copy in this repository.

One contract

`tools/gen_golden_vectors.py` drives the platform's function over a corpus-wide vocabulary and records **24,719 input → output pairs** in `contract/golden_roman_norm.ndjson`. `make contract` regenerates the vectors and fails if the committed file drifts; the app pins the file by sha256 in its `vendor.lock.json` and runs byte-parity tests against the Swift port; the data repository's copy is held to the same vectors. Change one character of the fold and three test suites fail before any index goes stale. How the contract is generated and replayed is on [Harnesses and golden vectors](#).

What the fold is not

- **Not scripture, not a transliteration.** `translit` is the reading aid; `translit_norm` is an index key that is never shown to a reader.
- **Not applied to Gurmukhi.** A Gurmukhi query goes to the text and skeleton columns; the fold exists for Roman input only.
- **Not a place for clever fixes.** A “small improvement” to the fold changes the key of every line in the corpus: rebuild the database, regenerate the contract, re-vendor the app.

Links

1. <https://github.com/Algorythmos-AI/siggs-platform/blob/c3d936031f0c283283c5d1102705affe1f8cf404/webapp/romannorm.py#L11-L28>
2. <https://github.com/Algorythmos-AI/siggs-platform/blob/integration/webapp/romannorm.py>

Sri Guru Granth Sahib Ji — KnowledgeBase wiki. Generated from commit c3d9360. Updated 25 September 2026. Verified against 7a343c6 on 2026-09-25. This copy may be out of date; the page is at <https://docs.gurbanisoul.com/search/the-roman-fold/>