

Search & verification

Search waterfall

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

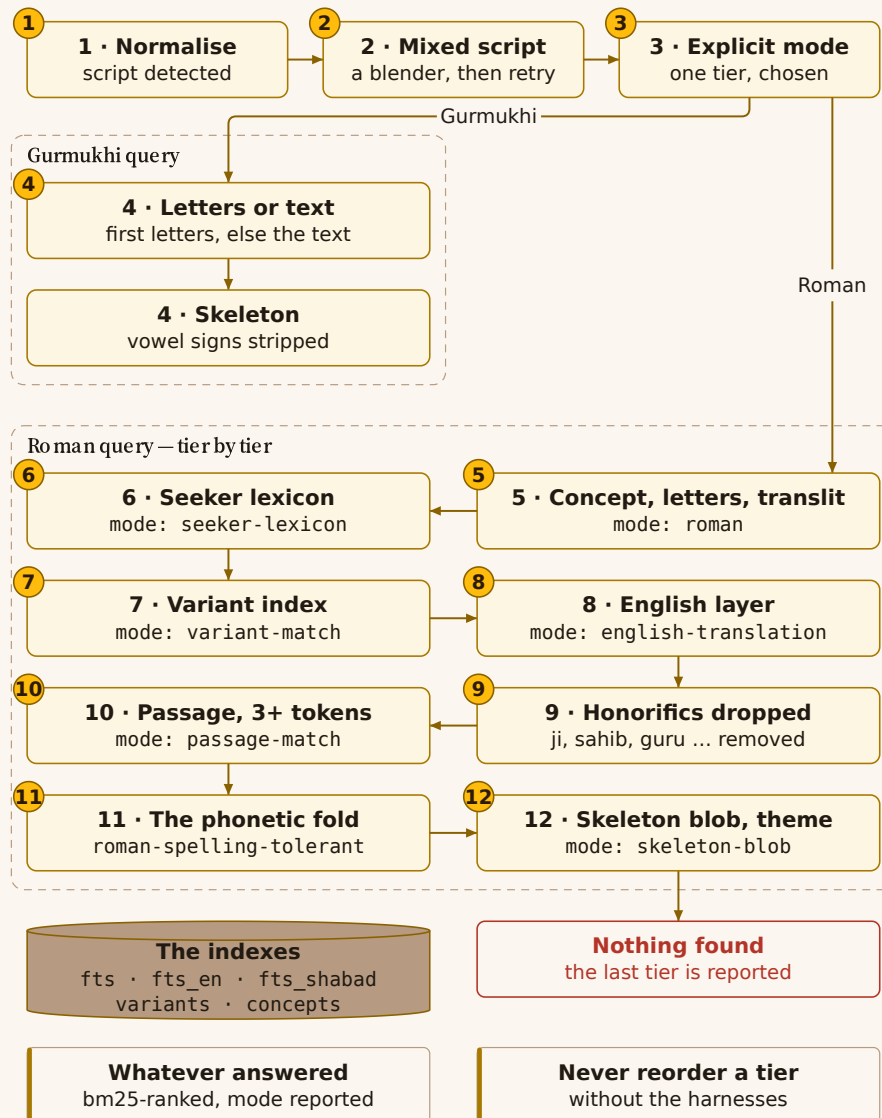
How `/api/search` decides, tier by tier, in the order `do_search` runs them — with a simulator that runs your query against the real API and lights the tier that answered.

<https://docs.gurbanisoul.com/architecture/search-waterfall/>

`/api/search?q=&mode=` runs a deliberate **waterfall**: cheap exact tiers first, progressively fuzzier tiers only when the earlier ones found nothing, results bm25-ranked with weighted columns. The poster shows the cascade as `do_search` codes it; the simulator below runs a real query and lights the tier that answered. The tiers in detail, the fold, the verification engine and the harnesses are in [Search & verification](#).

05 · The search waterfall

What `do_search` does with a query, in the order the code does it — exact tiers first, fuzzier ones only when the earlier found nothing



☐ component in this project ☒ data at rest
☐ nothing found: the last tier is reported, never a guess ☐ explanation
 — flow: data or control moves this way

Source of truth: `webapp/sggs/search.py` · `webapp/romannorm.py` · `webapp/sggs/core.py`

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

Sri Guru Granth Sahib Ji — Knowledge Base · engineering wiki

Poster 05 — The search waterfall: normalise, mixed script, explicit modes, then the auto-mode cascade of twelve tiers and the ranking

1. Normalise and detect the script. A query longer than 300 characters is refused. Dandas and punctuation become separators, whitespace collapses, and the code notes whether the query holds Gurmukhi, Latin letters, or both. In auto mode the API chooses the tier and reports it as mode; the simulator on this page reads that field.

2. **Mixed script.** Gurmukhi and Latin in one query (auto mode) go to `mixed_search` (mode `mixed-script`), a blender in which Gurmukhi tokens match the text column and Latin tokens resolve through variants, the lexicon and the fold in one AND expression. If that finds nothing, the Latin tokens alone are retried through the whole waterfall.
3. **Explicit modes.** `mode=gurmukhi, roman, first, theme` or `english` picks one tier up front — the text column, the transliteration (with a fold fallback), first letters, the theme index or the English layer — and skips the cascade.
4. **Auto, Gurmukhi: letters, text, skeleton.** A Gurmukhi query of single letters is read as first letters of each word (`fl_g, phrase match; mode first-letters`). Otherwise the verbatim text column is searched; if nothing matches, the vowel signs are stripped and the skeleton column is matched by `LIKE` (mode `gurmukhi-skeleton`).
5. **Auto, Roman: concept, first letters, transliteration.** A query that is exactly a concept name answers from the theme index. Two or more tokens of at most three letters are first letters (`fl_r`). Otherwise the transliteration column is searched (mode `roman`). Every later tier runs only if this one found nothing.
6. **The seeker lexicon.** A curated table of words seekers actually type — names, common spellings — maps to a transliteration term or a theme. The mode names the term that matched: `seeker-lexicon` (`vaahiguroo`).
7. **The variant index.** The precomputed romanisation-variant index resolves each token to up to three spellings, with a trailing-vowel retry, so one stubborn token cannot kill the AND (mode `variant-match`).
8. **The English layer.** Before the fold is tried, the English translations are searched (`fts_en`): "mercy" must reach the translations rather than collide with a fold of a Gurmukhi word (mode `english-translation`).
9. **Honorifics dropped.** `ji, sahib, maharaj, guru, dev, baba` and the like are removed and the shorter query is retried through the waterfall; the mode gains the suffix (`honorifics dropped`). After any tier, a thin result set is also topped up the same way (mode `... + honorific-dropped`).
10. **Passages across lines.** Three or more tokens are folded and matched at composition level in `fts_shabad`, so a couplet that spans two corpus lines is found and the composition's lines are returned in order (mode `passage-match`).
11. **The phonetic fold.** `roman_norm` folds the query and the index the same way (`translit_norm`), so `waheguru` and `vaahiguroo` meet at `vhgr`. Only tokens of two or more folded characters count, and heading lines are excluded because the fold collapses distinct words (mode `roman-spelling-tolerant`).
12. **The last resorts, and the ranking.** Up to 14 tokens collapse to a fold-skeleton blob that is prefiltered and fuzzy-ranked (mode `skeleton-blob`); a single token is tried as a theme. Whatever tier answered, results are `bm25`-ranked with the column weights 10, 5, 4, 3, 3, 1, translations are attached, related themes are added for Gurmukhi queries, and the mode is reported. Never reorder a tier or change a weight without the harnesses and `make contract`.

Try it

Interactive on the web: a search, traced through each tier.

The tiers, in code order

Every mode string below is a literal in `webapp/sggs/search.py`; `tools/docs_check.py` fails the build if the code gains a tier this page does not name.

#	Runs when	What it queries	mode reported
1	always	length ≤ 300; dandas and punctuation → spaces; script detected	—
2	auto · Gurmukhi and Latin present	mixed_search (one AND over text, variants, lexicon, fold); else the Latin tokens alone, re-run	mixed-script, mixed-script (latin part: ...)
3	an explicit mode	gurmukhi → text FTS then skeleton; roman → translit then the fold; first → fl_g/fl_r phrase; theme → concept index; english → fts_en	gurmukhi, gurmukhi-skeleton, roman, roman-spelling-tolerant, first-letters, theme, english
4	auto · Gurmukhi	all single-letter tokens → fl_g phrase; else text FTS; else vowel signs stripped → skeleton LIKE	first-letters, gurmukhi, gurmukhi-skeleton
5	auto · Roman	exactly a concept name → theme ; ≥ 2 tokens of ≤ 3 letters → fl_r phrase (then translit); else translit FTS	theme, first-letters, roman
6	nothing yet	the seeker lexicon: curated words → a transliteration term or a theme	seeker-lexicon (term)
7	nothing yet	the precomputed romanisation-variant index, hybrid per-token resolution	variant-match
8	nothing yet	the English layer; fts_en — before the fold, so “mercy” reaches translations	english-translation
9	nothing yet, honorifics present	drop ji, sahib, maharaj, guru, dev, baba, ... and re-run the waterfall	... (honorifics dropped)
10	nothing yet, ≥ 3 tokens	folded tokens AND at composition level in fts_shabad ; the composition’s lines in order	passage-match (quote spans lines)
11	nothing yet	the phonetic fold: roman_norm(query) against translit_norm , tokens ≥ 2 chars, headings excluded	roman-spelling-tolerant
12	nothing yet, ≤ 14 tokens	blob_search : the query collapsed to a fold-skeleton blob, prefiltered and fuzzy-ranked; a single token is tried as a theme	skeleton-blob, theme
—	after any tier, auto, < 3 results	the honorific-dropped retry tops up the list	... + honorific-dropped

Then, whatever tier answered: bm25 ranking with column weights **10 · 5 · 4 · 3 · 3 · 1** over **text, translit, translit_norm, fl_g, fl_r, skeleton**; translations attached to every result; related themes for a Gurmukhi query. Nothing is guessed: when every tier is exhausted the list is empty and the last tier reached is the mode.

webapp/sggs/search.py · do_search · lines 240–351

```

def do_search(q, mode, limit, offset):
    if len(q) > 300: raise ValueError('query too long (max 300 chars)')
    q = PUNCT_RE.sub(' ', q).strip()          # dandas & punctuation are separators
    q = re.sub(r'\s+', ' ', q)
    if not q: return {'mode': mode, 'results': []}
    if core.HAVE_FTS is None: core.HAVE_FTS = have_fts()
    toks = q.split()
    is_gurmukhi = bool(GURMUKHI.search(q))

    def run(col, phrase=False):
        if core.HAVE_FTS: return search_fts(col, q, phrase, limit, offset)
        return search_like('text' if col == 'text' else col, q, limit, offset)

    latin_present = bool(re.search('[a-zA-Z]', q))
    if mode == 'auto' and is_gurmukhi and latin_present:
        ms = mixed_search(q, limit, offset)
        if ms: return {'mode': 'mixed-script', 'results': ms}
        q_lat = ' '.join(t for t in toks if not GURMUKHI.search(t))
        if q_lat:
            sub = do_search(q_lat, 'auto', limit, offset)  # drop Gurmukhi tokens, retry
            if sub.get('results'):
                sub['mode'] = 'mixed-script (latin part: ' + sub['mode'] + ')'
                return sub
    # explicit modes
    if mode == 'gurmukhi':
        res = run('text'); used = 'gurmukhi'
        if not res and core.HAVE_FTS:
            res = search_like('skeleton', re.sub('[ᑭᑲᑯᑲ]', '', q), limit, offset); used = 'gurmukhi-skeleton'
        elif mode == 'roman':
            res = run('translit'); used = 'roman'
            if not res and core.HAVE_FTS:
                strong = ' '.join(t for t in roman_norm(q).split() if len(t) >= 2)
                if strong:
                    res = search_fts('translit_norm', strong, False, limit, offset,
no_headers=True)
                    used = 'roman-spelling-tolerant'
        elif mode == 'first':
            col = 'fl_g' if is_gurmukhi else 'fl_r'
            res = run(col, phrase=True); used = 'first-letters'
        elif mode == 'theme':
            return theme_search(q, limit, offset)
        elif mode == 'english':
            res = search_en(q, limit, offset); used = 'english'
        else: # auto
            if is_gurmukhi:
                single_letters = all(len(t) == 1 for t in toks) and len(toks) >= 2
                if single_letters:
                    res = run('fl_g', phrase=True); used = 'first-letters'
                else:
                    res = run('text'); used = 'gurmukhi'
            if not res:
                res = search_like('skeleton', re.sub('[ᑭᑲᑯᑲ]', '', q), limit, offset);
used = 'gurmukhi-skeleton'
        else:
            ql = q.lower()
            cpt = db().execute('SELECT concept FROM concepts WHERE concept = ?',
(ql,)).fetchone()

```

```

if cpt and ql not in LEXICON_OVER_SLUG:
    return theme_search(ql, limit, offset)
if all(len(t) <= 3 for t in toks) and len(toks) >= 2:
    res = run('fl_r', phrase=True); used = 'first-letters'
    if not res:
        res = run('translit'); used = 'roman'
else:
    res = run('translit'); used = 'roman'
if not res:
    lx = lexicon_search(q, limit, offset)
    if lx: return lx
if not res:
    vr = variant_search(q, limit, offset)
    if vr: return {'mode': 'variant-match', 'results': vr}
if not res:
    res = search_en(q, limit, offset)
    used = 'english-translation'
if not res:
    kept = [t for t in toks if t.lower() not in
{'ji','jee','jeo','jio','sahib','maharaj','maharaaj','shri','shree','sri','baba','guru','dev','waale','wale'}]
    if 2 <= len(kept) < len(toks):
        sub = do_search(' '.join(kept), 'auto', limit, offset)
        if sub.get('results'):
            sub['mode'] = sub['mode'] + ' (honorifics dropped)'
            return sub
    if not res and len(toks) >= 3:
        ps = passage_search(q, limit, offset)
        if ps: return {'mode': 'passage-match (quote spans lines)', 'results': ps}
# precise than per-line fold
if not res and core.HAVE_FTS:
    strong = ' '.join(t for t in roman_norm(q).split() if len(t) >= 2)
    if strong:
        res = search_fts('translit_norm', strong, False, limit, offset,
no_headers=True)
        used = 'roman-spelling-tolerant'
if not res and len(toks) <= 14:
    bl = blob_search(q, limit, offset)
    if bl: return {'mode': 'skeleton-blob', 'results': attach_translations(bl)}
# blob line – skip just those
if not res and len(toks) == 1:
    t = theme_search(q, limit, offset)
    if t['results']: return t
HONORIFICS = {'ji', 'jee', 'jeo', 'sahib', 'maharaj', 'maharaaj', 'shri', 'shree',
'sri', 'baba', 'guru', 'dev', 'waale', 'wale', 'jio'}
if len(res) < 3 and mode == 'auto':
    kept = [t for t in toks if t.lower() not in HONORIFICS]
    if len(kept) >= 2 and len(kept) < len(toks):
        sub = do_search(' '.join(kept), 'auto', limit, offset)
        if sub.get('results'):
            seen = {r['id'] for r in res}
            merged = [r for r in sub['results'] if r['id'] not in seen]
            if not res:
                sub['mode'] = sub['mode'] + ' (honorifics dropped)'
                return sub
            res = (res + merged)[:limit]

```

```

        used = used + ' + honorific-dropped'
    out = {'mode': used, 'results': res}
    if is_gurmukhi:
        rel = term_concepts(toks)
        if rel: out['related_themes'] = rel[:3]
    return out

```

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

Source: [webapp/sggs/search.py · do_search](#)³ — the excerpt is read from the file at build time.

The Roman fold — the highest-risk invariant

roman_norm folds spelling variants (waheguru, vaahiguroo → vhgr). It exists in **three** places that must stay byte-identical: webapp/romannorm.py here (query time; re-exported by serve.py, imported by verify.py), pipeline/sggs_pipeline.py: roman_norm in sggs-data (built the translit_norm column) and RomanNorm.swift in the iOS app. contract/golden_roman_norm.ndjson (24,719 vectors) pins all three. The whole story is [The Roman fold](#).

Verifying a search change

```

python3 tools/roundtrip_harness.py
python3 tools/casual_quote_harness.py
python3 tools/chaos_harness.py "qa/chaos/attacks_*.jsonl" qa/results/chaos.jsonl # needs a
running server
make contract # golden vectors must not drift unexpectedly

```

Do not reorder tiers or change weights without re-running these: small changes shift ranking corpus-wide. What each harness measures is on [Harnesses and golden vectors](#).

Links

1. <https://gurbanisoul.com/>
2. <https://github.com/Algorythmos-AI/sggs-platform/blob/c3d936031f0c283283c5d1102705affe1f8cf404/webapp/sggs/search.py#L240-L351>
3. <https://github.com/Algorythmos-AI/sggs-platform/blob/integration/webapp/sggs/search.py>

Sri Guru Granth Sahib Ji — Knowledge Base 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/architecture/search-waterfall/>