Checking Competitors' Indexed Pages: Scraping Google SERPs

When you need to know which URLs a rival has live in Google's index, **Checking Competitors' Indexed Pages: Scraping Google SERPs** is the fastest reconnaissance method short of owning their Search Console account. The logic is simple: run targeted site:competitor.com queries and extract every resulting URL. That list tells you what Google considers worth keeping in its main index—not just crawled, but actually indexed. A common scenario we see: a marketing team preparing a content gap analysis and suddenly realizing 30% of the competitor's supposedly "ranking" product pages never show up for any query. That missing chunk of the index is a direct signal about thin content, duplicate filters, or technical debt.

But this is not a one-click report. Scraping search results at any useful scale breaks fast if you ignore how Google's anti-automation layers behave. From a standard datacenter IP, you might get 5–10 requests before hitting a 429 or a reCAPTCHA wall. Residential proxy networks push that to 40–60 lookups, but the quality of retrieved results degrades because location-based personalization injects noise. And the moment you cross 100 queries per hour, you're essentially playing cat-and-mouse with a system engineered to filter out exactly that pattern.

What a SERP Scrape Actually Tells You

A lot of operators confuse "appears in search" with "indexed." They're different things. When you query site:example.com, the returned pages are a subset—usually the most "valuable" ones the ranking engine chooses to surface for that generic intrasite search. It does not guarantee that an omitted URL is completely absent from the index; it may simply not be surfaced. Google itself notes that the site: operator is approximate and not exhaustive. So your scraped dataset is a minimal index snapshot, biased toward pages with some internal link juice or canonical signal.

Still, for competitive baselining, that snapshot is gold. If a competitor has 4,000 URLs indexed according to their sitemap but your scrape only finds 2,800, the gap almost always maps to index bloat—pagination dupes, faceted navigation without proper canonical tags, or parameter-ridden assets. In practice, I have used a site:competitor.com/directory/ slice to confirm that a rival's UGC-heavy subfolder was 70% non-indexed because every user-generated page had the same thin content template. That fact alone justified reallocating our

link-building budget away from that area.

Picking a Spot on the Tooling Spectrum

There is no single “right” method. The spectrum runs from manual to fully automated, and the right choice depends on volume, technical tolerance, and how much you value your IP’s health.

> **Rule of thumb:** For fewer than 100 URLs, a browser-based manual scrape with careful copy-pasting is more reliable than an automated script that trips a block and burns your IP.

Manual: open an incognito window, submit site:competitor.com, note the “About X results” count, then step through page by page (&start=10, &start=20…) copying the green link lines. It’s tedious but leaves zero trace if you don’t use automation. For a one-off audit of 200 pages, an hour of clicking beats debugging a puppeteer error.

Semi-automated: a headful browser extension like Link Klipper (Chrome) can extract all links from a SERP tab with one click, but you still need to handle pagination manually. This is the sweet spot when you want a CSV of URLs for 500–800 indexed pages and you’re willing to spend 30 minutes. The risk of CAPTCHA stays low because the interaction pattern—user-paced clicking—looks human.

Automated: a headless browser script (Playwright or Puppeteer) that rotates user-agents, adds human-like delays, and possibly routes through a pool of residential proxies. This gets you scale—tens of thousands of lookups—but demands constant maintenance. Google updates their DOM structure often; class names like .yuRUbf or .TbwUpd can shift overnight. And the moment you exhibit a regular interval (*exactly* 3.7 seconds between requests), the behavioral anti-bot models catch you.

The Core Pipeline: From Query to Parsed Index List

If you decide to script it, the pipeline has three unavoidable stops: target construction, result extraction, and rate-limit handling. Here’s a minimal flow:

```mermaid flowchart LR A[Form site: query] --> B{Send request} B -- OK --> C[Parse HTML block] C --> D{Status 200?} D -- No, 429/403 --> E[Rotate IP / backoff] D -- Yes --> F{Contains CAPTCHA?} F -- Yes --> E F -- No --> G[Extract href links] G --> H{Pagination?} H -- Yes, next URL --> B H -- No --> I[Stop, save list] ```
```

Below is a bare-bones Python snippet that illustrates the extraction logic using requests and BeautifulSoup. It will fail quickly in the real world—Google serves a CAPTCHA almost immediately—but it shows the plumbing. The # NOTE: comments flag the fragile bits.

```python import requests, time, re from bs4 import BeautifulSoup HEADERS = { "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36" } QUERY = 'site:competitor.com' collected_urls = [] start = 0 while start The real-world problem: after 2–3 requests, you'll get a 200 but the body contains a CAPTCHA challenge instead of search links. Your script keeps running, collects nothing, and you waste time. That's why any serious pipeline adds a detection shim that looks for known CAPTCHA markers (e.g., id="captcha") and triggers an IP rotation queue.

More robust—and what a practitioner actually uses—is a **Playwright**-based approach that takes over a real Chromium instance and reuses a session with realistic mouse movements. The snippet below fetches the first page of results and prints link texts; you'd expand it to paginate and parse the <a> tags' href attributes. Notice the --disable-blink-features=AutomationControlled flag—leaving the automation flag visible gets you flagged.

```javascript const { chromium } = require('playwright'); (async () => { const browser = await chromium.launch({ headless: false, // visible to appear human args: ['--disable-blink-features=AutomationControlled'] }); const context = await browser.newContext({ userAgent: 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36' }); const page = await context.newPage(); await page.goto('https://www.google.com/search?q=site%3Acompetitor.com'); await page.waitForSelector('a h3', { timeout: 15000 }); const links = await page.$$eval('a h3', els => els.map(el => el.textContent)); console.log(links); await browser.close(); })(); ```

Edge case: even with a residential proxy and human-like delays, Google sometimes returns "About 0 results" when a real index count exists. That's often because the :443 / :80 port is inferred from the proxy exit node's ASN and the country mismatch triggers a different datacenter that hasn't synced the full index shard.

## What Breaks When You Scale

The core bottleneck is not parsing HTML; it's maintaining a session that Google considers low-risk. Spending days on a scraper only to realize your scraped

dataset contains 40% false negatives (pages that *are* indexed but hidden from that particular IP/location) is a common horror story.

- **Myth 1: a residential proxy makes you invisible.** Reality: residential IPs are scored by abuse history. A proxy pool with frequently recycled IPs often has entries already blacklisted for scraping; you'll hit a block even faster than a clean datacenter IP.
- **Myth 2: you can scrape after midnight to avoid detection.** Reality: Google's traffic models are time-zone agnostic; unusual request volume relative to a user profile is the trigger, not absolute time.
- **Myth 3: parser-only changes break scraping.** Reality: Google's JavaScript-driven lazy loading and cookie consent overlays often hide the result list until a scroll event fires, which headless requests skip entirely.

A less-technical but real-world obstacle: geographic data skew. Searching site:competitor.com from a German IP often shows fewer local-market pages than a US IP. You cannot easily acquire a representative global index snapshot without setting up multiple geolocation-aware scrapers, each with a resident proxy in that country, which balloons cost.

The "are you trying to outsmart Google" conversation usually ends with "use the official Indexing API" for your own URLs. But for competitors, there's no official endpoint. That's why commercial index-check solutions exist: they maintain their own proxy infrastructure and return a clean yes/no per URL. SpeedyIndex's bulk checker, for example, accepts a spreadsheet of competitor URLs and returns index status via API without you ever touching a scraper—the integration doc shows how. The trade-off is cost (per-URL pricing) versus the hours you'd spend nursing a scraper.



## A Real Recon Example with Numbers

In a recent drill-down on an e-commerce competitor, we targeted their /products/ subfolder. Their sitemap listed 9,430 product pages. A semi-automated scrape using a residential proxy-rotating browser over two evenings returned 7,612 unique URLs from the site: search. That's a 19.3% gap. After cleaning out 404 redirects and canonical mismatches (the scraper sometimes picked the parameter URL variant), we verified that 1,818 pages were *not* represented in any SERP snippet—consistent with "Crawled – currently not indexed" signals.

Later, we pulled the list through a bulk index checker API as a sanity check. The API agreed on 1,791 of those pages, a mismatch of just 27. Scraping gave us the initial signal cheaply; the API gave us the confident confirmation. Doing it the other way—buying 9,430 API checks upfront—would have cost significantly more without the pre-filter.

## FAQ for the Hesitant Operator

**Is scraping Google SERPs legal?** Depends on jurisdiction and method. In the U.S., the Computer Fraud and Abuse Act (CFAA) has been interpreted broadly, but scraping publicly accessible data without bypassing authentication is generally in a gray zone. Google's own robots.txt disallows /search for most crawlers; violating robots.txt may be used as evidence of unauthorized access. Practically, many SEOs scrape at low volume and never face legal action, but a cease-and-desist from Google is possible if you cause noticeable load or sell the data. Larger enterprises use official APIs or third-party services explicitly designed for competitive index analysis.

**How many URLs can I realistically scrape before getting blocked?** From a single residential IP with no special configuration, 15–50 requests per hour normally triggers a temporary block. With a well-tuned residential proxy fleet (rotating IPs, session mimicry), hundreds per hour are possible, but the cost exceeds most alternative tools.

**Does site: ever show all indexed pages?** No. Google intentionally limits the display for large sites. You'll often see a note like "In order to show you the most relevant results, we've omitted some entries…" The scrape gives you a partial, prioritized subset.

**Can I compare scraped data with Search Console to check my own site?** Yes, and that's straightforward—you own the property. But for competitor checks, you don't have Search Console access, so the site: scrape remains the primary free method.

**Are there non-scraping alternatives?** Paid third-party databases (Ahrefs, Semrush) maintain their own indices and can show estimated indexed pages, but they're crawl-based and not live. The SpeedyIndex index checker API and similar services are probably the closest to live status without scraping—see their API documentation.

## If You'd Rather Not Scrape at All

Scraping is the brittle, low-cost shortcut that keeps SEO departments moving

when official doors are closed. It works until it doesn’t. The moment it burns your IP or wastes half a day chasing false negatives, the economics invert. For a production-grade competitor index monitor, a purpose-built index-check API that already solves proxy rotation, CAPTCHA handling, and result deduplication makes more sense—even if it costs a few hundred dollars a month. That’s cheaper than a senior technician’s time spent debugging Playwright selectors on a Friday night. The scraper serves one-off reconnaissance and proof-of-concept; the API serves systematic data pipelines. Knowing when to stop scraping is the real skill.

---

## Sources


1. Schema.org. "Getting Started with Schema." schema.org
2. Google Search Central. "Search Essentials." developers.google.com
3. Search Engine Journal. "SEO Guide." searchenginejournal.com