I Crawled 65,000 Pages of My Own Site Without Parsing a Single Line of HTML

Metehan Yesilyurt

AI Search & Information Retrieval Researcher · metehan.ai

Published: May 04, 2026 • Canonical: https://metehan.ai/blog/http-headers-internal-links/

Last week I spoke at **SEO Week 2026**[1] in New York City, organized by iPullRank[2], on April 27th and 30th. It was, easily, the most concentrated room of practitioners thinking about where SEO, AEO, and GEO are actually going.

Somewhere between talks on day one, I think it was during a hallway conversation about how badly LLM crawlers struggle with rendered DOMs. A question I had been chewing on for months finally crystalized:

> *What is the smallest, fastest, most boring thing a website can do to make itself perfectly understood by every crawler, scraper, and LLM that visits it?*

Robots.txt? Sitemaps? `llms.txt`? Schema.org[3]? They all help. They all also assume the same thing: that a crawler is going to download your HTML, render it, parse it, and politely figure out what you meant.

That's a lot of trust to place in a stranger's parser.

So I flew home and tried something different. I took my entire internal link graph, every page, every connection, encoded it as compact JSON, base64url-ed it, and stuffed it directly into an HTTP response header on every single page of my site. Then I added a second header carrying the page's heading hierarchy.

Not in the body. Not in a sidecar file. **In the headers.**

Then I crawled 65,000 pages in **99 seconds** without parsing one byte of HTML.

This post is the story of that experiment, the numbers from a fresh 100-URL run I did this morning, the prior art (yes I checked), the things that broke in production, and the open-source code so you can replicate it on your own site.

> *All the code: **github.com/metehan777/http-header-link-graph**[4] (Cloudflare Worker + Rust crawler + Python insights pipeline, MIT-licensed).*

The idea in one sentence

> *HTTP responses can carry **8 KB to 32 KB** of headers depending on the server config. That is more than enough room to ship structured site metadata next to every page for free, on every request, before a single*

> *byte of body is sent.*

Once that clicks, a lot of things change.

A WAF blocks the body? The headers still went out. A page returns 403? The headers still went out. A page returns 500? The headers still went out. The page is JavaScript-rendered? The headers went out before the JS even loaded.

Headers travel light, headers travel first, and headers travel even when everything else falls apart. That makes them a beautiful place to publish things you actually want crawlers to see.

What I actually built

A toy. On purpose.

A small Cloudflare Worker that serves a real site (`data.stateglobe.com`, 65k+ pages), but on every response it adds:

HTTP

```
X-Internal-Links: WyIvIiwiL2Jsb2ciLCIvYXJ0aWNsZXMiLCIvYW5hbHl0aWNzIi...
X-Internal-Links-Encoding: json+base64url
X-Internal-Links-Bytes: 1455
X-Internal-Links-Count: 31

X-Headings: W3sibCI6MSwidCI6IkFJIEFkb3B0aW9uIFJhdGUgU3RhdGlzdGljcyBpbiBBZmdoYW5pc3Rh...
X-Headings-Encoding: json+base64url
X-Headings-Bytes: 534
X-Headings-Count: 8
X-Headings-Schema: [{l:1-6,t:string}]

Access-Control-Expose-Headers: X-Internal-Links, X-Internal-Links-Encoding,
X-Internal-Links-Count, X-Internal-Links-Bytes, X-Headings,
X-Headings-Encoding, X-Headings-Count, X-Headings-Bytes, X-Headings-Schema
```

Decode `X-Internal-Links` and you get the full list of internal links from that page:

JSON

```
["/", "/blog", "/articles", "/analytics",
"/category/ai-machine-learning", "/afghanistan",
"/albania/ai-adoption-rate-statistics", ...]
```

Decode `X-Headings` and you get the page's heading skeleton:

JSON

```
[{"l":1,"t":"AI Adoption Rate Statistics in Afghanistan (2026)"},
{"l":2,"t":"Frequently Asked Questions"},
{"l":3,"t":"What are Afghanistan's main sectors adopting AI in 2026?"},
{"l":4,"t":"Methodology"}, ...]
```

No DOM, no Readability, no Playwright, no Cheerio.

Then I asked one question:

> *If I act like a scraper, with no HTML parsing at all, can I rebuild the entire internal link graph **and** the heading hierarchy of every page using only response headers?*

Spoiler: yes. And the speed surprised me.

Run 1: 65,000 pages, 14 minutes (cold cache)

I wrote a Rust crawler with `tokio` and `reqwest` (because once you taste 1,000 RPS you cannot go back), pointed it at the live site with 800 concurrent connections, and watched it grind away.

```
TEXT
elapsed_human = 14m 14s
rps = 76
unique_pages = 65,292
```

76 requests per second. For a static-feeling site. Ouch.

The bottleneck was obvious once I looked: every request was hitting my Worker, executing logic, and serializing the link payload from scratch. Cloudflare's edge cache was sitting there, completely empty, watching me re-render the same HTML for the millionth time.

So I taught the Worker to use `caches.default.put` and added a sane `Cache-Control: public, s-maxage=7200, max-age=300`. Then I purged the edge cache once to flush stale stuff.

Run 2: 65,000 pages, 99 seconds (warm cache)

Same crawler, same site, same 65,000 pages:

```
TEXT
elapsed_human = 1m 39s
rps = 660 (peaks at 969)
unique_pages = 65,292
```

8.6× speedup. Once Cloudflare's edge had a copy of every response with my custom headers attached, the Worker barely had to lift a finger.

The crawler isn't even downloading the full body. It is streaming the response headers, decoding a base64 string, and skipping the rest. That's why 1 KB of header outperforms 50 KB of HTML the crawler never asked for the 50 KB.

Run 3: 100 URLs, both headers, no cache, this morning

After the talk in NYC, a few people asked, *"yeah but can you actually fit headings in there too?"*

So I deployed an updated Worker that also emits `X-Headings`, then ran a fresh, **cache-busted** 100-URL probe so we could measure the worst case (every request hits the Worker, no edge cache, both headers populated). Code is

in `scripts/probe_100.py`.

Here is the actual output:

```TEXT
URLs probed: 100
Status mix: 200=100
5xx pages: 0 (0.0%)
Network errors: 0
Latency (ms): min=173 | p50=274 | p95=364 | max=428

Pages exposing X-Internal-Links (decoded ok): 100 (100.0%)
Pages exposing X-Headings (decoded ok): 100 (100.0%)
Pages exposing both: 100 (100.0%)

X-Internal-Links bytes: min=1375 | p50=1544 | p95=1710 | max=1967
X-Headings bytes: min=338 | p50=551 | p95=615 | max=662
Combined bytes: min=1848 | p50=2085 | p95=2321 | max=2629
Pages exceeding 8 KB combined: 0 (0.0%)

Mean links per page: 31.0
Mean headings per page: 7.9
Heading-level distribution: H1=100, H2=100, H3=392, H4=200
```

Translating from machine to human:

- **100 pages, 1.88 seconds, all 200s.** Both headers present and decoded successfully on every single page.
- **p95 latency 364 ms** while hitting the Worker on every request, no edge cache.
- **p95 combined header size 2.3 KB.** Comfortably inside the 8 KB budget I aim for.
- The **heading distribution is very revealing**: every page has exactly 1×H1, 1×H2, plus a long tail of H3/H4. That tells me my templates are consistent, which is exactly the kind of structural signal AEO/GEO systems reward and I never had to render the page to find that out.

This is what I mean by “headers as a publishing surface.” We are not waiting for a parser to figure out the hierarchy. We are *handing it over*.

Then I built SEO insights from headers alone

After the 65k crawl finished, I had 65,000 JSON payloads sitting in a file. Each one was just `{ url, links: [...] }`. No HTML.

Out of curiosity, I wrote a small Python script (`scripts/seo_insights.py`) that did one thing: build the full directed graph from those headers and compute SEO metrics on it.

What came back floored me:

- **27,372 pages (41.9%) had zero inbound internal links.** Pure orphans, only reachable through the sitemap. On a site that was supposed to be tightly cross-linked.
- **27,659 pages were unreachable by walking from `/`.** Same root cause.
- **Click-depth distribution:** 54% of pages were 4+ clicks deep from the homepage. 53% sat at depth 6.

- **Gini coefficient on inbound links: 0.918.** That's near-monopolistic. The top 10 pages were hoovering up 17% of all internal links.
- **Cluster inequality.** The country sub-folders had identical structure -301 pages each- but median inbound count was **199 for some countries** and **1 for others**. Same template, same code, wildly different link distribution.

I have used Screaming Frog, Sitebulb, Ahrefs, OnCrawl, JetOctopus. They are all great. But this hit different. Because it took 99 seconds, cost approximately nothing, and surfaced a structural problem that no amount of "let's audit the homepage" would have caught.

The crawl wasn't the product. The crawl was just the *delivery vehicle* for a much cleaner SEO insight pipeline.

And now, with `X-Headings` shipping alongside, I can layer on:

- Heading uniqueness across the site (duplicate H1s, missing H2s, broken hierarchy)
- Topical drift between cluster pages
- AEO-friendly question detection (any H3 phrased as a question)
- Snippet candidates per page

…all without rendering a single page.

Why this matters for SEO, AEO, and GEO

This is the part that got me genuinely excited at SEO Week, because it generalizes.

I started with internal links and headings. But the header is just an envelope. You can put almost anything structured inside it.

For traditional SEO

- **Internal link graph** (what I tested): orphans, dead-ends, click-depth, hub identification, Gini concentration.
- **Heading skeleton** (what I added this week): hierarchy validation, topical clustering, duplicate detection.
- **Canonical hints, hreflang, content hash**: ride along with every response, no extra request.
- **Last-edited timestamp**: helps crawlers decide whether to revisit. Cheaper than `Last-Modified` games.
- **Page tier signal**: tell crawlers "this is a tier-1 hub" or "this is a tier-3 long-tail page" so they can budget accordingly.

For AEO/GEO (Answer Engine Optimization)

This is where it gets fun. AI engines like Perplexity, ChatGPT, Gemini, and the new wave of search agents are extremely sensitive to *clarity of structure*. They reward sites where the topic, the headings, and the link relationships are obvious.

A header set like:

```HTTP
X-Headings: <base64url-json (flat list of H1-H6 with anchor IDs)>
X-Page-Topic: "ecommerce conversion benchmarks Bangladesh 2026"
X-Page-Entities: ["Bangladesh", "ecommerce", "conversion rate", "2026"]
X-Page-Summary: <base64url(280-char abstract)>
```

…lets any answer engine that fetches your URL get a perfect, parser-free abstract before the body even arrives. You are not begging the crawler to "understand" your page. You are *handing* it the understanding.

You're publishing a machine-readable "if you quote this page, here is the canonical snippet to use, and here is the license." That is way more useful for an LLM than scraping a paragraph and hoping it picked the right sentence.

For technical SEO operations

- **Crawl prioritization at the edge.** Inject `X-Crawl-Priority: high` on hub pages.
- **Crawl change detection.** A hash of the page's link graph in the header lets you detect navigation drift between deploys without comparing HTML.
- **Independence from rendering.** Edge headers are added by your platform, not your CMS. So even if marketing forgets to add canonical tags, the platform still publishes the truth.

Has anyone done this before? (I checked, yes and no.)

After SEO Week I spent an hour digging through prior art, because if this idea were obvious someone would have shipped it by 2015. Here is what exists, and how it differs from what I am proposing.

1. RFC 8288 - `Link` HTTP response header (2017). The canonical standard for putting links in HTTP headers. Used in production for `rel="canonical"`, `rel="next"/"prev"`, `rel="preload"`, `rel="api-catalog"`. It is *not* used to ship a page's full internal link graph wrong shape (one rel per link), and the syntax balloons fast.

2. Cloudflare's Agent Readiness framework (Q1 2026). Cloudflare's agent-readiness[5] post lists `Link` headers as one of three official "discoverability" standards for agents. Alongside `robots.txt` and `sitemap.xml`. They explicitly say: *"agents can discover important resources directly from HTTP response headers… without having to parse any markup."* But they're still scoped to the standard rels. Nobody at Cloudflare (or anyone else I could find) is pushing a "full link graph in a header" pattern.

3. `llms.txt` (Sept 2024, Jeremy Howard / Answer.AI[6]). A markdown file at `/llms.txt` that describes your site for LLMs. Closest to my idea in *intent*. But it is a separate file (extra round-trip), curated and static, not per-page metadata. My approach: zero extra requests, per-page granularity, machine-precise structured payload instead of a hand-curated reading list.

4. JSON-LD via `Accept: application/ld+json` content negotiation. A documented pattern to serve a JSON-LD version of a page when the client asks for it. Still a separate request, still the body, still requires the agent to opt in. My approach arrives unconditionally on every response, in the headers, regardless of what the client asked for.

5. Open Graph / structured data via headers (proposals). A few Stack Overflow[7] and Webmasters SE[8] threads where developers asked, *"why can't we put OG/JSON-LD in HTTP headers?"* The community answer is

always: “search engines don’t support it,” so it stayed theoretical.

My angle is different. I am not asking Google to parse it. I am publishing it for **my own tooling**, my own SEO/AEO pipelines, my own monitoring, my own dev team and any forward-thinking agent that decides to read it. That inversion is the new bit. The prior art is all about *what crawlers ask for*. My framing is about *what site owners can publish*.

If anyone has shipped exactly this before, I genuinely want to hear about it. Reach out and I’ll happily update this post.

A real-world warning: I broke my own site doing this

Now the embarrassing part.

About an hour before I sat down to write this post, I deployed the second header (`X-Headings`) and ran a probe. The 100 sub-page URLs returned 100/100 success.

Then I tried `curl https://data.stateglobe.com/` .

```
TEXT
HTTP/2 500
content-type: text/plain; charset=UTF-8
content-length: 16
```

My homepage was 500-ing. So was `/blog` . So was `/articles` . So was `/analytics` every single hub page.

The reason? My homepage had **230 links** and a long heading list. `X-Internal-Links` alone was around **10.8 KB**. Add `X-Headings` at ~6.9 KB on top, and the *combined response headers* on those hub pages crossed **~17 KB**. That breached the response-header size limit on the path between origin and edge, and the platform refused to deliver the response at all. Real users were getting 500s.

The sub-pages, where the link list was shorter (31 links, 1.5 KB) and the heading list was small (8 items, 540 bytes), worked perfectly. The hubs, which are by definition the most important pages on the site, were broken.

This is exactly the kind of thing you don’t want to learn in production.

The fix: a defensive header-budget module

After watching my own homepage 500 in production, I extracted a tiny, dependency-free TypeScript module that **enforces a combined header-size budget and gracefully truncates the payload before it can blow up your origin**. It’s in the repo at `src/headers.ts` and works in any modern JS runtime. Cloudflare Workers, Next.js middleware, Deno, Bun, Node 18+.

The interface is minimal:

```ts
import { attachStructuralHeaders } from "./headers";

return attachStructuralHeaders(
new Response(html, { status: 200 }),
{
url: req.url,
links: getInternalLinks(page), // can be huge, will be capped
headings: getHeadings(page), // can be huge, will be capped
}
// defaults: 6 KB per header, 12 KB combined
);
```

Internally it does three things:

1. **Per-header cap (default 6 KB).** Each list is shrunk in 10% chunks until its base64url-encoded size fits under the per-header budget.
2. **Combined hard cap (default 12 KB).** If both fit individually but their sum exceeds 12 KB, the heading list is truncated first, then if needed the link list, until the combined size fits.
3. **Truncation telemetry.** When clipping happens, the response gains `X-Internal-Links-Truncated: 1` and `X-Internal-Links-Original: 230` (and the equivalents for headings), so your monitoring can alert you when budgets are being hit.

I added a stress test in `scripts/test-headers-budget.mjs` that throws **5,000 links + 5,000 headings** at the function. Result: combined output is 11.4 KB, safely under the 12 KB cap, and the response still ships valid (truncated) payloads. No 500. Ever.

What's running on data.stateglobe.com[9] right now

After deploying that module and a defensive `combined > 12 KB → truncate` rule, the live site is back to 200s on every page, including the homepage and hubs.

Important note about the live demo: to make the experiment safe to run on a real site, I'm intentionally truncating the payload on `data.stateglobe.com` for testing. Hub pages with 200+ links would otherwise need a chunked-header approach (`X-Internal-Links-1`, `X-Internal-Links-2`, …) to ship the full graph. For the public demo, I'd rather you see a clean, 200-OK response with a representative subset of links + headings than a "complete" payload that risks 500ing hubs. Treat the live numbers as a lower bound on what's possible.

If you want to see the full, untruncated technique, run the local Worker in the repo[4] small demo site, no truncation needed.

Read this list before you ship anything

So please, before you ship this on anything that matters:

1. **HTTP response header size limits are real and origin-dependent.** Cloudflare's default response-header limit is around 16 KB. Many origins enforce 8 KB or stricter. If your combined headers don't fit, your origin returns 5xx, to crawlers *and* to humans.
2. **Hub pages are the danger zone.** A homepage with 200+ links and a 50-item heading map can easily blow past the limit. Test every hub before rollout.

3. **Always cap the payload defensively.** Use `attachStructuralHeaders` or roll your own equivalent. Set a hard byte limit (e.g. 6 KB per header, 12 KB combined) and gracefully truncate when over budget. Better to ship 50 of 200 links than to 500 the page.
4. **Cache it at the edge.** Workers don't cache by default. You have to explicitly `caches.default.put` with a real `Cache-Control`. Otherwise every crawl hits compute, and you'll see 76 RPS instead of 660.
5. **Edge caches are sticky.** After deploying a new header shape, purge once, otherwise old responses keep getting served.
6. **HEAD requests are not your friend.** I tried switching the crawler to HEAD to skip body bytes. Cloudflare and many origins respond differently to HEAD, and I lost the headers. Stick with GET; the body is cheap to drop on the client side.
7. **Treat missing headers as crawl-incomplete, not as data.** When my first insights pass tagged 256 pages as "dead ends," they were actually pages where the header was missing on that fetch. Re-crawl those before drawing conclusions.
8. **Scope this to your own domains.** This is a publishing technique for site owners, not a bypass tool. Don't go ramming custom headers through someone else's WAF and call it a day.

Important: do not roll this out alone, especially in enterprise

I want to be very direct about this, because it will save someone's job.

This experiment is fun on a personal site. **It is not a thing you ship to an enterprise site without your dev team in the room.** Specifically your platform/SRE team, your security team, and whoever owns your CDN config.

Why?

- This change touches your **edge layer, your origin response budget, and your bot/security ruleset** all at once. None of those are "the SEO team's domain."
- A misconfigured rule can **5xx your homepage to real users** (literally what happened to me an hour ago, in a controlled experiment).
- Many enterprise stacks (Akamai, Fastly, NetScaler, F5, custom WAFs) have their own response-header rewriting, header-size limits, and stripping behaviors that aren't documented anywhere obvious. You will only find out by testing.
- You almost certainly need to coordinate with **logging and observability**. Custom headers can show up in access logs, in CDN logs, in SIEM rules, in compliance audits. Surprise nobody.
- Custom headers can also get **stripped by intermediate proxies** (some corporate/enterprise WANs aggressively prune unknown headers). Test from the actual customer egress paths if your audience includes corporate networks.

The right rollout is:

1. Build it on a staging hostname.
2. Get sign-off from platform/SRE on header-size limits, caching strategy, and observability impact.
3. Get sign-off from security on the new bot/agent surface.

4. Roll out to a single product directory first (e.g. `/blog/*` only), monitor 5xx for a week.
5. Then expand to the rest.

If you are an SEO doing this solo on a serious site, you are going to have a bad week. Bring your team in early. They will probably also help you avoid the homepage-500 trap I just walked into.

The architecture, in three boring sentences

1. Generate a small JSON payload per page describing its links / headings / topic. Cache it.
2. On every response, attach the payload as a base64url-encoded HTTP header. **Cap the size** so combined headers stay safely under your origin's limit.
3. Cache the response at the edge so the second crawl is essentially free.

That's it. There is no machine learning, no fancy infra, no protocol change. It is the kind of thing a single dev can ship in an afternoon and the kind of thing a careless dev can use to break production in five minutes. Be the first kind.

The bigger picture: headers as a publishing surface

Most of SEO has trained us to think of "the content" as the thing on the page. The body. The DOM. The words.

But every HTTP response is actually two things stacked on top of each other:

```text
Headers ← machine-readable, fast, structured, almost free
Body ← human-readable, slow, unstructured, expensive
```

For 30 years, we shoved everything into the body and asked machines to figure it out. Crawlers got stronger, parsers got smarter, and we kept paying the cost of that translation, every single request, forever.

Meanwhile, the headers above the body the most efficient part of every HTTP response on the planet were sitting there carrying `Content-Type: text/html` and not much else.

I think that is going to change. As LLMs and answer engines become the dominant consumers of the web, they will reward sites that publish *structured intent*. And there is no faster, cheaper, more universal place to publish structured intent than HTTP headers.

You don't need a new protocol. You don't need a new standard. You don't need WebSub or ActivityPub or some W3C committee. You just need to put the JSON in the header and serve it. Carefully.

The crawler does the rest in milliseconds.

Where I'm taking this next

A few directions I'm exploring this week:

1. ~~A defensive header budget library.~~ ✅ **Shipped.** `src/headers.ts` Pure, dependency-free, drop into any Worker or middleware. Stress-tested with 5,000 links + 5,000 headings. Will not let your origin 500.
2. **Topic embeddings as a header.** A 256-dim quantized vector base64'd. LLMs can compare pages without reading them.
3. **A crawl-budget protocol.** A header that says "I have 65,000 pages, 240 hubs, last full re-index was 14 hours ago." Let crawlers use that to decide how aggressively to revisit.
4. **A public hosted "header crawler" API.** Take any site you own, pass the base URL, and watch a 99-second full audit happen. The Rust crawler in the repo already does this; I want to host it as a free SaaS for the SEO community.
5. **AEO-friendly header pack.** `X-Headings` + `X-Page-Topic` + `X-Cite-Snippet` + `X-Page-Summary` A reference set with sensible defaults and hard size caps.
6. **Chunked-header support.** `X-Internal-Links-1`, `X-Internal-Links-2`, … so very large hubs can ship the full graph without truncation.

If any of this excites you, the code is on **GitHub**[4]. Open an issue, send a PR, or just ping me. This is the kind of weekend rabbit hole that is genuinely more fun with collaborators.

You can also check outputs; I'm building my own custom reporting schema.

- **Homepage in nodes (HTML):** https://metehan.ai/graph-input/link-graph[10]
- **JSON report:** https://metehan.ai/graph-input/seo-graph-report.json[11]
- **CSV report:** https://metehan.ai/graph-input/seo-header-report.csv[12]
- **Header summary:** https://metehan.ai/graph-input/seo-header-summary.json[13]

Closing thought

I went into this experiment thinking I would solve a crawling problem.

I came out of SEO Week 2026 thinking I had stumbled onto a different way of looking at the entire SEO/AEO/GEO stack: **the header is the most under-used publishing surface on the web, and it is going to matter more, not less, as machines do more of the reading.**

65,000 pages. 99 seconds. No HTML parsed. A 41% orphan rate I would have never caught with traditional tools. A 100-URL fresh probe with both headers, both decoded, in 1.88 seconds. One spectacularly broken homepage as a free lesson. And about a hundred new ideas for what to put in a header next.

If you have a site you own, an afternoon to spare, and your dev team in Slack, try it. The first thing you will notice is how *small* the change is. The second thing you will notice is how much it changes how you think about your site.

Thanks again to **Michael King and the iPullRank team** for putting on SEO Week 2026. This idea wouldn't exist without that room.

> *Code, sample reports, and the full Worker:* ***github.com/metehan777/http-header-link-graph***[4]

Related deep-dives: where to put schema code on your website[14], answer engine optimization tools with near-instant insights[15], and how many internal links a blog post should have[16].

HTTP headers and internal links are the quiet backbone of AEO. AI citations in ChatGPT, Perplexity and Gemini follow authority signals that live in headers and link equity long before content quality matters. Answer engine optimization starts with infrastructure.

REFERENCES & SOURCES

[1] **SEO Week 2026** — https://seoweek.org/

[2] **iPullRank** — https://ipullrank.com/

[3] **Schema.org** — http://schema.org/

[4] **github.com/metehan777/http-header-link-graph** — https://github.com/metehan777/http-header-link-graph

[5] **agent-readiness** — https://blog.cloudflare.com/agent-readiness/

[6] **Answer.AI** — http://answer.ai/

[7] **Stack Overflow** — https://stackoverflow.com/questions/77369003/open-graph-protocol-via-http-response-headers

[8] **Webmasters SE** — https://webmasters.stackexchange.com/questions/83372/why-dont-search-engines-support-json-ld-schema-org-as-seperated-endpoints-via

[9] **data.stateglobe.com** — http://data.stateglobe.com/

[10] **https://metehan.ai/graph-input/link-graph** — https://metehan.ai/graph-input/link-graph

[11] **https://metehan.ai/graph-input/seo-graph-report.json** — https://metehan.ai/graph-input/seo-graph-report.json

[12] **https://metehan.ai/graph-input/seo-header-report.csv** — https://metehan.ai/graph-input/seo-header-report.csv

[13] **https://metehan.ai/graph-input/seo-header-summary.json** — https://metehan.ai/graph-input/seo-header-summary.json

[14] **where to put schema code on your website** — https://metehan.ai/articles/where-to-put-schema-code-in-website-best-practices-for-modern-seo-and-ai-discovery/

[15] **answer engine optimization tools with near-instant insights** — https://metehan.ai/articles/answer-engine-optimization-tools-near-instant-insights/

[16] **how many internal links a blog post should have** — https://metehan.ai/articles/how-many-internal-links-in-a-blog-post/