Claude Code Web Pipeline

A custom web research pipeline that replaces Claude Code's built-in `WebFetch` with clean markdown extraction, project-level caching, and AI-powered triage — keeping full-resolution web content available while consuming minimal context window.

The Problem

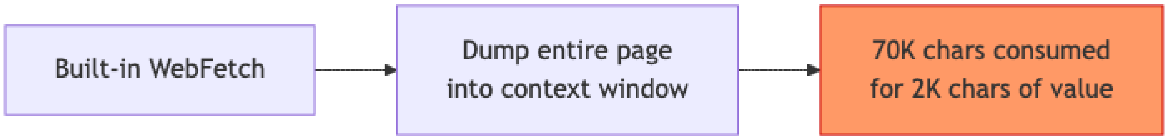
Built-in WebFetch
Dump entire page into context window
70K chars consumed for 2K chars of value

Figure 1: The problem: dumping entire pages into context

Claude Code's built-in web tools either dump entire pages into the conversation (wasteful — a typical page is 20-70K chars) or return lossy summaries (insufficient for detailed work). Neither approach scales when you need to research across multiple pages.

The Solution

Step-by-Step

1. Discover — `WebSearch`

Claude Code's built-in web search returns concise summaries and source URLs. Good for discovery, bad for extraction.

2. Fetch + Cache — `batch_read_urls`

A single MCP tool call fetches all discovered URLs in parallel through Jina Reader (`r.jina.ai`). Each page is:

- Rendered through Jina's headless browser
- Converted to clean markdown
- Stripped of junk (nav, cookies, ads, sidebars, modals — 30+ CSS selectors)
- Images removed (markdown-only)
- Cached permanently to `{project}/.web_cache/{hash}.md`

The tool returns **metadata only** — never the content:

```
[
  {"path": "/project/.web_cache/a1b2c3.md", "title": "Page Title", "lines": 340, "chars": 18200, "cached":
  {"path": "/project/.web_cache/d4e5f6.md", "title": "Another Page", "lines": 520, "chars": 31000, "cached
]
```

3. Triage — Parallel Haiku Agents

Cheap, fast AI agents (one per cached page) scan the full content and return structured JSON identifying exactly which line ranges are relevant to the query:

```
{"relevant": true, "ranges": [[10, 45], [80, 92]]}
```

```
{"relevant": false}
```

Agents run in parallel — triage takes ~3-5s regardless of page count.

4. Inject — `Read` with offset/limit

Only the relevant line ranges enter the main conversation. Everything else stays on disk, available for later queries.

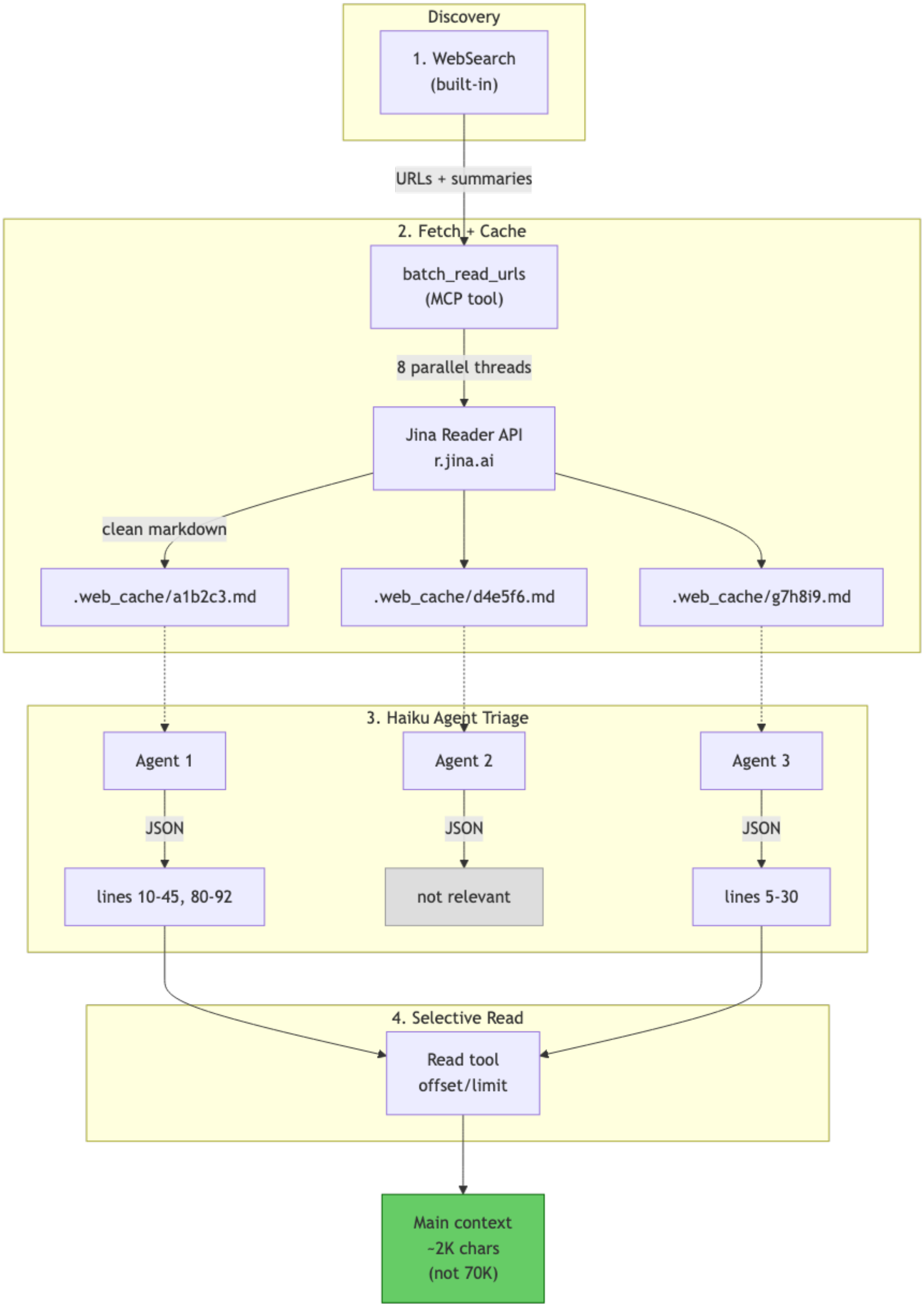

Figure 2: End-to-end pipeline: WebSearch → batch fetch → cache → haiku triage → selective read

What Gets Stripped

30+ CSS selectors strip common junk patterns before caching:

Category	Selectors
Navigation	`nav`, `header`, `footer`, `[role='banner']`, `[role='navigation']`
Consent	`[class*='cookie']`, `[class*='consent']`, `[id*='cookie']`
Popups	`[class*='popup']`, `[class*='modal']`, `[class*='overlay']`
Marketing	`[class*='newsletter']`, `[class*='subscribe']`, `[class*='signup']`
Ads/Promos	`[class*='advert']`, `[class*='sponsor']`, `[class*='promo']`
Social	`[class*='share']`, `[class*='social']`, `[class*='comment']`
Layout junk	`[class*='sidebar']`, `[class*='widget']`, `[class*='related-post']`

Cache Format

Files stored as `.web_cache/{sha256[:12]}.md` with YAML frontmatter:

```
---
url: https://example.com/article
title: How to Build a Web Pipeline
fetched: 2026-02-15T10:30:00+00:00
hash: a1b2c3d4e5f6
---
Title: How to Build a Web Pipeline
URL Source: https://example.com/article
Markdown Content:
[... clean article content ...]
```

Cache is **permanent and project-scoped** — pages persist across sessions. Use `list_cache` to see what's available before re-fetching.

Performance

Scenario	Pages	Cached	Actually Used	Compression	Triage Time
Narrow single-page query	2	24K chars	1.2K chars	**95%**	~4s
Multi-page comparison	3	29K chars	8.5K chars	**70%**	~3s
Broad 5-page research	5	83K chars	10.5K chars	**87%**	~5s

Compression scales with irrelevance — narrow queries save more. Triage latency is flat because agents run in parallel.

Architecture

Three files. That's it.

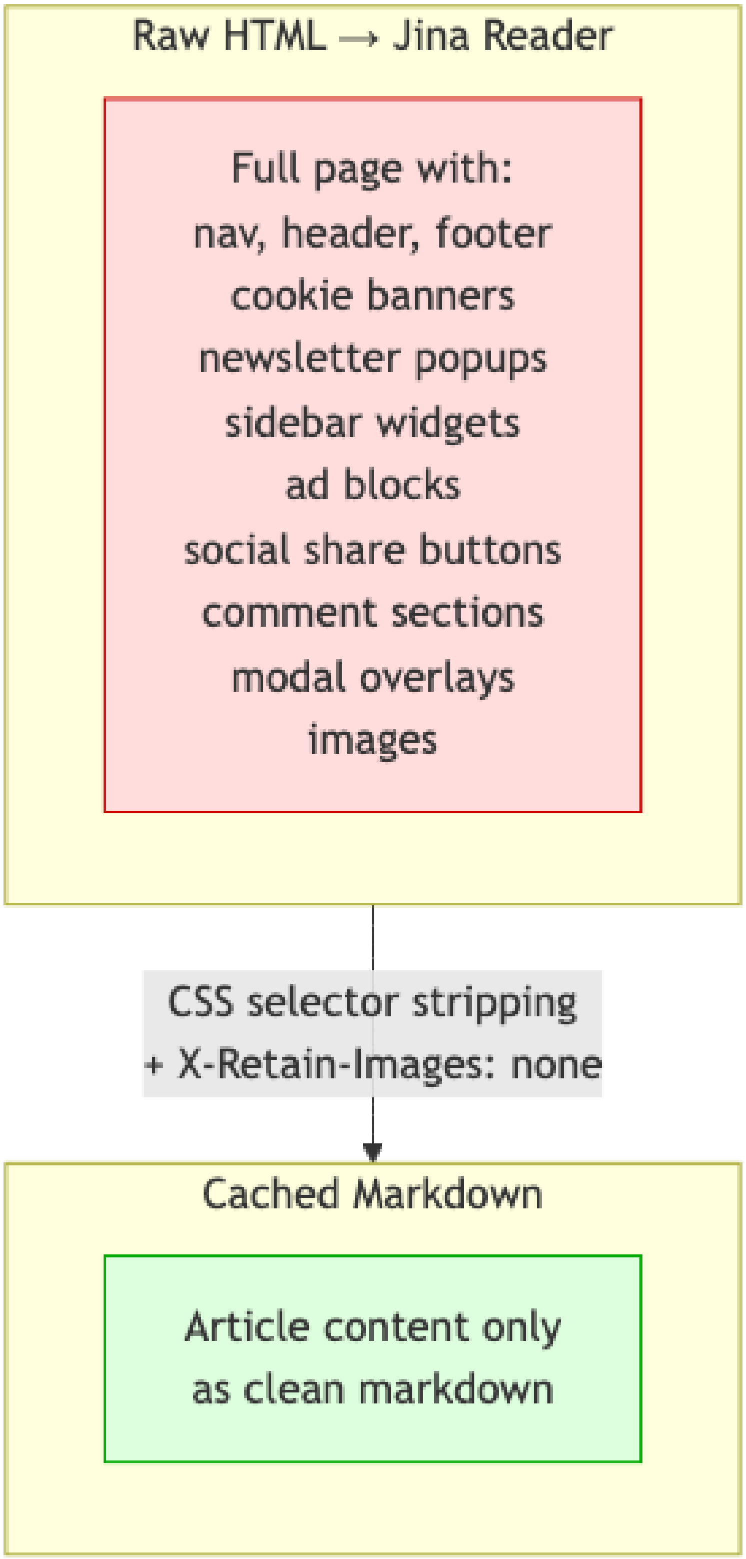

Figure 3: CSS selector stripping: raw HTML junk → clean markdown

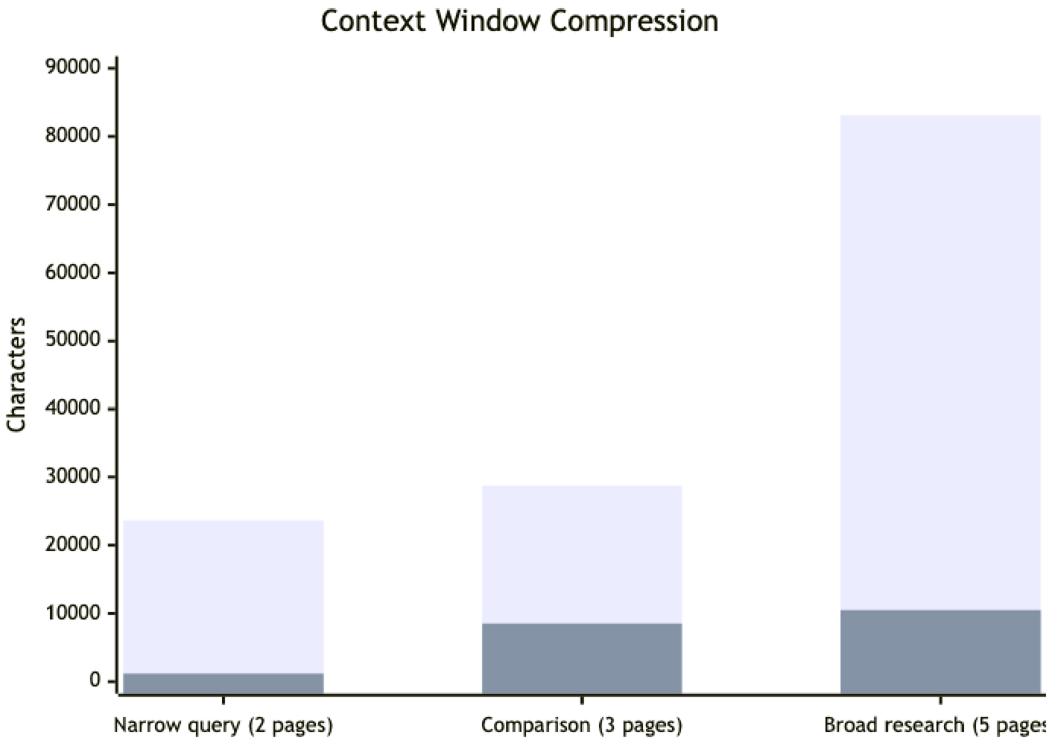

Figure 4: Context window compression: cached vs actually used

File	Role
`~/.claude/mcp-servers/jina-reader.py`	The MCP server. 3 tools (`read_url`, `batch_read_urls`, `list_cache`), CSS stripping, Jina API integration, caching. Single self-contained Python script — runs via `uv run --script` with inline PEP 723 deps.
`~/.claude.json`	Registers the server globally under `mcpServers.jina-reader` with the `JINA_API_KEY` env var.
`~/.claude/CLAUDE.md`	Instructions that tell Claude Code to always use this 5-step workflow for any web task.

Setup

1. **Get a Jina API key** from jina.ai
2. **Drop the server script** at `~/.claude/mcp-servers/jina-reader.py`
3. **Register it** in Claude Code:

   ```
   claude mcp add -s user -e "JINA_API_KEY=your_key_here" -- jina-reader \
     uv run --script ~/.claude/mcp-servers/jina-reader.py
   ```

4. **Add workflow instructions** to `~/.claude/CLAUDE.md`
5. **Restart Claude Code** to pick up the new MCP server

Known Limitations

- **Cloudflare-protected sites** (Medium, Hacker News) may return challenge pages (~500 chars). Skip these.
- **Nav-heavy sites** (The New Stack) still have junk despite CSS stripping — content may be buried deep.

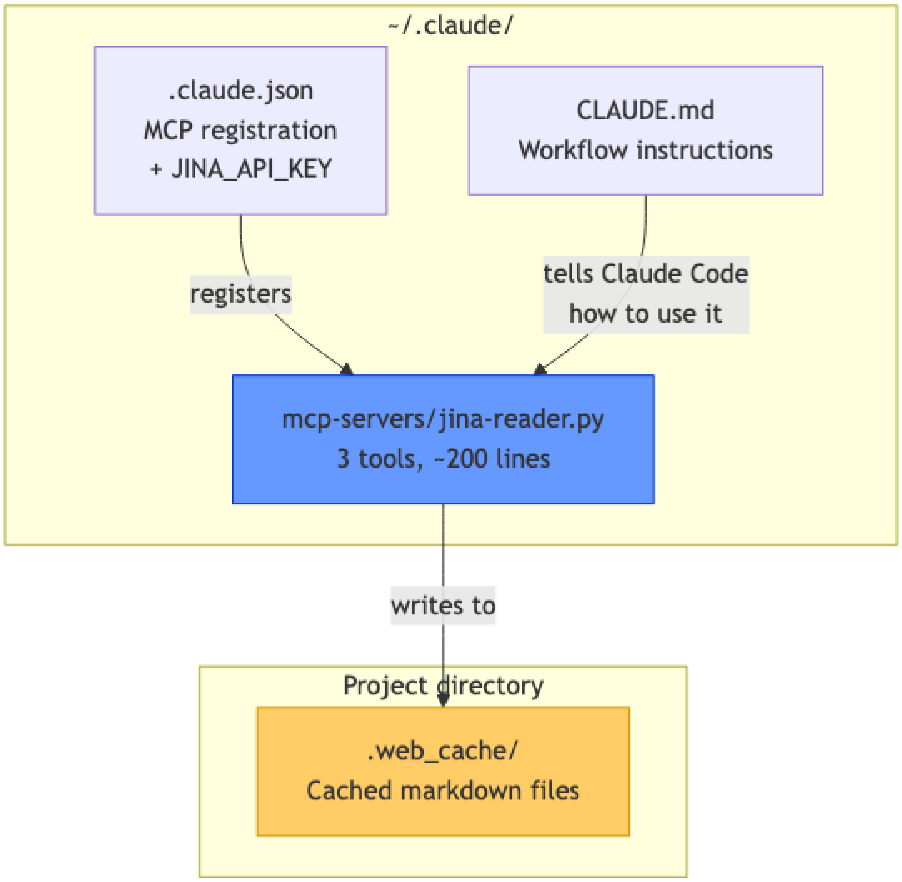

Figure 5: Architecture: 3 config files → project cache

- **MCP server restarts** are required after editing `jina-reader.py` — restart Claude Code to apply changes.
- **`cache_dir` must be absolute** — the MCP server's working directory may not match the project directory.