

WHITEPAPER

surf: Context Traversal without Ingestion*

Alexander R. Saint Croix
alex@saintx.us

September 2026

Abstract

surf is a command-line tool that returns one section of a markdown, TeX, or PDF document by heading, and returns the document's heading tree when no heading is given. I built it so that agent skills can be written as thin indexes: a `SKILL.md` that lists the intents an agent might arrive with and, for each one, a single command that pulls the exact section of reference material that intent needs. The reference corpus can then grow without the index changing, an agent loads only what its task requires, and the relationship between index and corpus can be checked by machine. The surf skill ships this paper as its TeX and PDF example, so every command it describes can be run against it.

1 Background

Three approaches to giving a language model context have a defect in common. Each tends to hand the model more than the task needs and leave it to sort out which part applies.

Retrieval-augmented generation (RAG) selects chunks of a corpus by their resemblance to a query and places them in the prompt. Resemblance to a query and relevance to the agent's task are different measures, and RAG can compute only the first. The model receives passages that look like the question and then spends attention deciding which of them bear on the work.

Long instruction files are the second approach. A `CLAUDE.md` or `AGENTS.md` loads on every turn, and a file that has grown to cover every situation a project might present is, on any given turn, mostly about other situations. The model attends over all of it to find the part that applies.

Skills are the third, and they improve on this by adding agency. A skill loads when the agent decides it applies, or when a person invokes it directly, so the decision to bring material into context moves from a RAG pipeline or a static file to the agent itself. But the decision stops at the skill boundary. Once invoked, the entire `SKILL.md` lands in context, and a long one carries the same situational content a long instruction file does. The agency costs the whole file.

In January 2026 I kept some skill reference material as JSON and wrote the rows of a `SKILL.md` as `jq` queries against those files. The agent ran the query it needed and received the value. The file itself never entered context. I call this context traversal without ingestion.

Most of my reference material is structured prose, though. A markdown file carries a heading tree the way a JSON file carries keys, and what it lacked was a command-line utility that would

*Marc McCahill is credited with coining the phrase “surfing the Internet.” This tool lets agents surf the seas of context.

list that tree for an agent or return the prose under one path through it. I wrote surf in March 2026 to be that utility. It addresses a section by its heading and returns the section. Run against the usage reference that ships with the surf skill:

```
$ surf references/usage.md --list
- Surf Usage
  - Default: file map
  - Frontmatter only
  - Heading tree only
  - Targeted section
  - TeX
  - PDF
  - Level
  - Wikilink and link input
  - Batch scanning
  - Platform aggregation (batch section extraction)
  - Additional options

$ surf references/usage.md "Level"
## Level

`--level 1` is the top rank. On markdown, rank 1 is `#`. A file whose
first heading is `##` needs `--level 2` to list it. On TeX, rank 1 is
the shallowest command in that file. In an article that starts at
`\section`, `--level 1` is `\section`. On PDF, rank 1 is the top
outline rank.

`--list --level N` prints ranks 1 through N. A named heading with
`--level N` matches only a heading at rank N.
```

The agent has seen twelve lines of headings and one section from the middle of the file, and it has not opened the file. By June I was building skill indexes shaped by intent on top of this, and in September surf gained TeX sectioning commands and PDF outlines so that the same skills could index academic papers.

2 A Thin Index Shaped by Intent

A skill built on surf has two parts. The `SKILL.md` is a short file: one sentence stating what the skill does, followed by a table grouped by the intents an agent might have on arrival. Each row pairs an intent with one command that resolves to a section of a reference file by its heading.

Intent	Section extraction
See what a file contains	surf references/usage.md "Default: file map"
Pull one section	surf references/usage.md "Targeted section"
Address a TeX paper	surf references/usage.md "TeX"

Those three rows come from the surf skill's own index, and each resolves against the `usage.md` that ships beside this paper.

The `references/` directory holds the corpus. I break reference files into sections of twenty to eighty lines and write each so that an agent arriving at it with no surrounding context can act from it alone.

The two parts have different jobs and change for different reasons. The index is intensional. It defines, by intent, which sections matter. The corpus is extensional. It holds the sections themselves. Datalog makes the same split between an intensional database of rules and an extensional database of ground facts [1], and the properties carry over. A row in the index holds a query, never a copy of the content, and the query resolves against the corpus at the moment the agent runs it. So the corpus can be rewritten or extended without touching the index, and the index can be regrouped by intent without touching the corpus. The only thing both sides depend on is the heading tree. The index changes when the structure of the corpus changes, and at no other time.

A `surf` call with no heading returns that structure. `surf file.md` prints the file's frontmatter and then its heading tree, and nothing from the body. An agent's first move on an unfamiliar file is to look at the map, and its second is to select a section. Neither move loads the file.

3 Agentic Context Composition

The question that separates this pattern from RAG is who decides what enters the context window. In RAG a component outside the model decides, before the model has seen anything. In a skill built on `surf` the model reads the intent index, judges what its current task needs, and issues the pulls itself. Agent inference, rather than a similarity score, composes the context.

Invocation loads the index and nothing else. This holds whether the agent chose the skill or a person invoked it by name. When I tell an agent to `surf` my holdings for papers on a topic, I have forced the skill to load, and I have forced only the index. Which papers it lists, which sections it pulls, and in what order remain the agent's decisions against the task I gave it.

The cost structure follows from this. In a long `SKILL.md`, presenting an option to the agent costs the full text behind that option, whether or not the agent takes it. In a thin index, presenting an option costs one row, and taking it costs one section. A skill can therefore lay out every route it supports, and an agent that needs one route pays for one section while an agent that needs none pays for a table.

Models perform worse as the volume of irrelevant material in the context window grows, even when the material they need is present [2, 3]. Every token the model attends to that has no bearing on the task is a token of inference spent on sorting rather than on working. A skill that hands the agent only the sections it asked for spends that inference on the work.

4 Skills You Can Check

Because the index is data, a script can verify the relationship between index and corpus.

Every row is an executable command. After any edit to a skill, a script can run every listed command and confirm that each resolves to a section. A row that fails is a defect, and the most common cause is a renamed heading the index did not follow. Renaming a heading is a schema change under this pattern, and the check treats it as one.

The reverse check is more interesting. A script can list every section in the corpus and count how many index rows reach each one. A section reached zero times is a finding, and the count says which questions to ask. The index may have fallen behind the corpus and be missing a row. The section may be content nobody routes to, and belong trimmed or folded into a neighbor. The corpus may have outgrown one index and need a second. Or the file may already be structured

for navigation by intent, so that an agent told to surf its heading tree directly needs no row for it at all. Skills written as prose cannot be audited this way, because there is nothing to count.

Two conventions make the checks reliable. The index quotes headings literally from **surf** **--list**, macros and punctuation included, so it never guesses at an address. And structural divisions use real headings rather than bold text, because surf sees the first and cannot see the second.

5 Indexes over Indexes

An index lists what a corpus contains. A procedure, a numbered list of steps, says what to do with it. Most of what a skill carries is the first kind, and a thin index hands the agent that inventory without ordering it. The order comes from the task.

Indexes compose, and composing them is cheaper than enlarging one. A family of twenty skills with forty sections each is eight hundred rows in a single index, all of them loaded on every invocation to use one. Split into a top index that routes by broad intent to member skills and a member index in each that routes by narrower intent to its sections, the same destination costs twenty rows and then forty, and the agent sees no others. A row in one skill's index may also point at a section in another skill's corpus, so shared reference material lives in one place and every skill that needs it routes there. The agent descends as far as the task requires and at each level pays for one small table, never for the corpora beneath it. A family grows by adding a member index and one row above it.

6 Markdown, TeX, and PDF

surf addresses a markdown file by its ATX headings, a TeX file by its sectioning commands and its **abstract** environment, and a PDF by its outline. The agent's two moves are the same in each case: list the section titles, then select the one to read.

```
surf paper.tex --list --level 2
surf paper.tex "abstract"
surf paper.pdf "Experiments#Programming"
```

A markdown wiki adds a second kind of structure. In an Obsidian vault, a section's body names other sections by link, `[[note#Heading]]` or `[[note#Parent#Child]]`, and surf accepts that link text as a target. An agent reading one section follows a link with the same command it used to arrive, so the vault is traversed one section at a time along the links its authors wrote, and no note is loaded whole.

```
surf "[[agents/memory#Episodic buffer]]"
```

A paper already has the structure the pattern needs, maintained by its authors. One skill can therefore index a family of related papers at subsection granularity, with rows that resolve to one experiment section of one paper. An agent working the topic reads the sections its question touches across the family and nothing else.

Shared structure across a corpus goes further. Most papers have an abstract, so a loop over a directory of papers that runs **surf \$f "abstract"** projects that one section out of most of the collection. If every skill in a library keeps a **references/about.md** with the headings

Overview and **When to use**, two commands over the library return a digest of every skill for a few hundred tokens each. Mine does, and the digest is how I orient an agent to some eighty skills without loading one of them. **Frontmatter**, read with `surf -f`, filters the collection to the files that share a structure before the projection runs. A heading convention, seen this way, is a schema declaration, and a corpus that follows one can be queried like a table. In every format the address is the text that `surf --list` prints for the heading.

7 This Paper

The surf skill needs a TeX file and a PDF that it may ship, and this paper is both. The commands below run against it as distributed.

```
surf surf.tex --list
surf surf.pdf --list
surf surf.tex "abstract"
surf surf.pdf "Skills You Can Check"
```

The two listings return the same tree. The skill's build checks that claim mechanically, in the way Section 4 describes, each time this file changes.

References

- [1] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.
- [2] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.
- [3] K. Hong, A. Troynikov, and J. Huber. Context rot: How increasing input tokens impacts LLM performance. Chroma technical report, July 2025.