ChatGPT

Ref.tools MCP Server – Technical Manual

Technical Architecture

Server Implementation (STDIO vs Streamable HTTP): The Ref MCP server can run either as a local process using STDIO or as an HTTP service. It has *robust support for both traditional STDIO and an experimental streamable HTTP server mode*, giving flexibility in deployment [1]. The HTTP mode is recommended (as of 2025) for most integrations [2], as it allows clients (e.g. IDEs or ChatGPT plugins) to connect via a `https://api.ref.tools/mcp` endpoint with an API key. The STDIO mode (considered legacy) can be launched locally (e.g. via `npx ref-tools-mcp`) if needed [3]. In STDIO mode, the server process communicates through standard input/output streams; in HTTP mode it listens on a port (default 8080) and uses a `/mcp` JSON-RPC endpoint for tool calls [4] [5]. You can choose the mode by setting the `TRANSPORT` environment variable (e.g. `TRANSPORT=http` with `PORT=...` for HTTP, or default to `stdio`) when running the server.

Available Tools: Ref.tools MCP provides two primary tools by default (plus an optional third):

- `ref_search_documentation`**:** A powerful search tool that **indexes technical documentation**. It finds facts or code snippets from **public docs** (for libraries, APIs, etc.) and can also search **private resources** (like your own repo docs or PDFs) if configured [6] [7]. You provide a `query` string (ideally a full question or description including relevant library/framework names [8]), and it returns an overview of results with documentation URLs. By default it searches *all sources*, but an internal `source` parameter can target `'public'` vs `'private'` vs `'web'` contexts [9]. (More on sources below.)

- `ref_read_url`**:** A companion tool to fetch and read content from a given URL. It retrieves the **full page content** (converted to Markdown) of a documentation page [10]. Typically, the agent will take a URL returned by `ref_search_documentation` and call `ref_read_url` on it to get the relevant section of docs. The server expects the *exact URL from the search result* as input [11]. This allows the AI to follow documentation links and read them in detail.

- **Optional –** `ref_search_web`**:** In scenarios where official docs don't have the answer, Ref includes an **optional web search fallback** tool [12] [13]. If enabled, `ref_search_web` performs a general web search (e.g. across forums, articles, etc.) to find relevant information outside the indexed docs. This is mainly used as a backup when normal doc search fails [14]. By default, this tool is **disabled** (Ref focuses on documentation first), but it can be enabled in configuration. You can **disable** the web search tool via an environment variable `DISABLE_SEARCH_WEB=true` (or conversely enable it by leaving that false) [15] [16]. In HTTP mode, a URL param `disable_search_web=false` can also control this at runtime [15]. Many setups choose to keep `ref_search_web` off to ensure the agent sticks to high-quality docs unless absolutely needed.

Session State & Search Trajectory Tracking: Ref is designed to work in a **stateful session** manner, closely mirroring how an AI agent would iteratively search. Each session persists context about past queries and results. This enables several optimizations:

- *No Repeat Results:* For repeated or refined searches in the same session, Ref **never returns the same link twice**, even across multiple queries [17]. It tracks which documentation pages have already been surfaced. This means an agent that adjusts its query will "page" deeper into results or find new sources instead of wasting tokens on duplicates. This approach lets the AI both page and refine its search simultaneously [17], which is more efficient than a fixed batch of top results.

- *Adaptive Search & Read:* The architecture separates "searching" from "reading". An agent can perform a lightweight search (few hundred tokens for the query and summary) and then **selectively read** only the pages that look relevant [18] [19]. This iterative search→read cycle means **token usage is adaptive** – simple questions might only retrieve one small doc page, whereas complex tasks can dig through several pages as needed [20] [21]. The agent controls how deep to go. This contrasts with batch retrieval approaches (like Context7's fixed snippet injection) which always use a large chunk of tokens regardless of need [21] [22].

- *Contextual Chunking ("Smart Chunking"):* When the agent uses `ref_read_url` on a documentation page, Ref doesn't naively return the entire page if it's huge. Instead, it **filters and extracts only the most relevant sections (up to ~5K tokens)** based on the query context [23]. It uses the session's search history and query terms to "drop out" less relevant parts of the page, returning a focused excerpt (for example, from a 90K-token API reference page, it might return only ~5K tokens that matter) [24] [23]. This smart chunking prevents the common issue with simple web-scraping tools where a large doc page could flood the context with tens of thousands of mostly irrelevant tokens [25]. Ref's chunking ensures the *agent gets deep links into documentation sections* – effectively the server pinpoints the subsection of a page that answers the query. (For instance, a search result might come with a URL fragment like `#merge` to jump directly to the relevant section [26].)

- *Pre-fetch Caching:* The server will cache results within a session so that if the agent needs to read a page it just searched, it can fetch it faster from cache [20]. This reduces latency and avoids duplicate external fetches.

In practice, these mechanisms mean Ref builds a search trajectory: the agent can do multi-step research and the server optimizes each step by remembering what's already seen and focusing on what's next. Both Ref and Context7 now use stateful sessions to avoid duplication [18], but Ref's approach is more agent-driven and fine-grained in token control.

Token Optimization Mechanisms: A core benefit of Ref is aggressive **token efficiency**. By returning only the needed documentation excerpts (rather than large pre-digested blobs), it dramatically cuts down tokens inserted into the prompt. The developers claim that Ref uses *on average ~60% fewer tokens than Context7, and in some cases up to 95% fewer* [27]. For example, Context7 might inject ~10K tokens for a single library query (and double that if two libraries are involved) [22], whereas Ref might only use a few hundred tokens for a search plus a few thousand for a targeted read. One user reported an extreme case of using only **2,800 tokens instead of a 100K token context dump** by switching to Ref (about a 97% reduction in context size) [28]. The table comparing Context7 vs Ref shows Context7's typical fixed ~3K tokens per query vs Ref's adaptive 500–5K range [29]. This efficiency not only saves cost (fewer tokens sent to the LLM = less money spent [30]) but also improves quality by avoiding *context rot* (large irrelevant context can confuse the model [31]). In short, Ref's architecture finds **"exactly the tokens your coding agent needs… and nothing else."** [27]

Private Repositories and PDF Indexing: In addition to public documentation, Ref lets you index **private docs** so your agent can search your internal knowledge. This is a major architectural difference from many other solutions. Out of the box, **Ref supports private GitHub repositories and PDF/Markdown files** as first-class search sources (included in the base plan, not as a paid add-on) [32] [33]. The Ref backend includes a web crawler and indexer: you can connect your GitHub account to it and specify which repos to index, or upload documentation files via the Ref web dashboard [34] [35]. Once indexed, those private resources become searchable via the same `ref_search_documentation` tool (the agent can specify `ref_src=private` or use the provided prompt to search personal docs [36] [37]).

- **GitHub Repo Indexing:** You may connect repos that you **are a member of** (for permission reasons) [38]. Repos under ~2,000 files will have *all files indexed (including code)*, whereas very large repos (>2000 files) index only docs (likely to stay within limits) [39]. The indexer runs on a schedule – currently repos sync about every 5 minutes, picking up new commits incrementally [40]. This means your internal code docs or README changes get reflected quickly. (If you rewrite git history, you might need to re-add the repo due to how incremental sync works [40].) Authentication uses a GitHub personal access token you provide, and currently only personal/organization repos via PAT are supported (a GitHub App integration is on the roadmap) [41].

- **PDF/Markdown Uploads:** You can directly upload PDF files, Markdown docs, or similar to your Ref account [35]. For example, teams might upload internal API specs, design docs, or any non-public documentation in PDF format. Ref will index the text of those files so that `ref_search_documentation` can find content within them. This effectively extends the "source of truth" beyond web docs to any proprietary documents you add. It's a simple way to give the AI access to company-specific knowledge. (One limitation noted by users is that currently you cannot arbitrarily add *web* URLs to your private index unless you manually download them as PDF/MD or the site is supported – a feature request to index arbitrary external URLs is pending [42].)

API Key and Service Model: The Ref MCP server is not purely offline – it acts as a client to Ref's cloud API (https://api.ref.tools). **You must obtain an API key** from ref.tools to use it (sign up on their website). The MCP server will pass this key in requests (via `X-Ref-Api-Key` or `X-Ref-Alpha` headers) to authenticate with the Ref API [43] [44]. Environment variables `REF_API_KEY` (for normal keys) or `REF_ALPHA` (for early alpha access keys) are used to configure the key for the server process [45]. If the key is missing or invalid, the tools will not function (the agent will get an error message like "Ref is not correctly configured. Reach out to hello@ref.tools for help.") [46]. This design means the heavy lifting (crawling, indexing, searching) is done by Ref's backend service – the local MCP server is essentially a bridge that formats your queries, calls the Ref API, and returns results in the MCP format.

Pricing Model: Ref operates on a **credit (query) based subscription**. There is a free tier for trying it out – *approximately 200 credits for free*, which the team says is about 10 weeks of light usage [47]. For higher usage, the **Basic plan is $9 USD/month for 1,000 credits** (i.e. roughly 1,000 queries or tool calls) [48] [47]. In comparison, Context7's base plan was $10 for 500 queries [30], so Ref offers about double the number of searches per dollar [30]. There's also a **Team plan** priced at $9/month per seat, also including 1,000 credits per seat [47] – this allows multiple developers or multiple AI agent instances to share an organization but each with their own quota. Enterprise and higher-volume options likely exist (Ref mentions teams and enterprise pricing in docs [49]), and additional credits can presumably be purchased if you exhaust the base amount, though specifics on overage pricing haven't been published in docs we have. The key takeaway is that each **tool invocation typically consumes one credit** (whether it's a search or a read), so an iterative workflow (search→read→search→read) might use a few credits. But thanks to token efficiency, you may still save money on your LLM bill even if you pay for Ref, because you're sending far fewer prompt tokens on average [27] [31].

Environment Variables Summary: To configure Ref MCP server, you will typically set: `REF_API_KEY` (your API key, required), optionally `DISABLE_SEARCH_WEB=true` (to turn off the web search fallback tool) [15], and if needed `REF_ALPHA` (if you have an alpha token instead of a normal key). During local development you might also set `TRANSPORT` and `PORT` if using the HTTP server locally. The server also honors an internal `REF_URL` var for pointing to a custom API endpoint (defaults to `api.ref.tools`) [50] – usually you won't change this unless instructed by Ref (e.g. testing a new backend). All other parameters are handled server-side through your account settings (like which private repos are indexed, etc.). The environment design keeps the client lightweight.

Use Cases & Workflows

Ref.tools MCP is designed to support a variety of coding assistant workflows by providing **on-demand documentation lookup**. Below are common use cases and how to leverage Ref in each scenario:

1. Public Documentation Lookup (APIs, Libraries): The simplest use case is when an AI coding assistant needs information from a popular library's docs. Instead of relying on possibly outdated training data or dumping whole docs, the agent can use `ref_search_documentation`. For example, if asked *"How do I post a comment via the Figma API?"*, the agent might issue a search query like *"Figma API post comment endpoint documentation"*. Ref will search Figma's official API docs and return a result URL (e.g. the endpoint reference page). The agent then calls `ref_read_url` on that URL to get the specific endpoint details (method, payload, etc.). In the Ref README, a real example is shown: the agent did `SEARCH 'Figma API post comment endpoint documentation'` (54 tokens) and then `READ https://www.figma.com/developers/api#post-comments-endpoint` (385 tokens) to retrieve exactly the needed snippet [51] [52]. This pattern – **search the official docs, then read the relevant page** – ensures the answer is up-to-date and authoritative, and uses only a few hundred tokens of context. Ref's ability to deep-link means the retrieved page is precisely where the answer lies, not just a homepage or generic link.

2. Internal/Private Documentation Query: Many organizations have private APIs, internal libraries, or just codebases with their own docs. After connecting these resources to Ref (via the Resources dashboard), your agent can search them similarly. If a developer asks the AI, *"How does our internal payment service handle refunds?"*, the agent can do something like: `SEARCH 'refund API PaymentService internal docs ref_src=private'`. The `ref_src=private` flag (which can be implied if using the built-in "my_docs" prompt [53] [54]) tells Ref to only search your indexed private resources. The agent might get back a URL to a markdown file in your repo or a section of a PDF you uploaded. It then uses `ref_read_url` to read that content and answer based on it. This workflow essentially brings **your company's knowledge base into the AI's reach** in real-time. It's especially valuable for on-boarding (the AI can answer questions about internal systems) or for referencing things like internal API endpoints, config schemas, etc. without manually feeding those into the prompt. As long as those docs are indexed by Ref, the AI can find and quote them.

3. Multi-Step Research (Complex Queries): For more complex problems, an AI might need to gather information from multiple places or refine its search as it learns. Ref shines in these multi-step research workflows, because it lets the agent iterate efficiently. Consider a real example from the docs involving the **n8n automation tool's "Merge" node**. The question might be: *What is the best way to combine data from multiple sources in n8n – using a Merge node or a different approach?* This isn't answered in one go, so the AI conducts a series of searches and reads:

- First, it searched *"n8n merge node vs Code node multiple inputs best practices"* [26] . Ref returned a doc page about the Merge node.
- The agent `READ` that page (`.../n8n-nodes-base.merge/#merge`) which was ~4961 tokens [26] , learning what the Merge node does.
- It then read another small page about merging data (138 tokens) [26] .
- Next, it searched a refined query *"n8n Code node multiple inputs best practices when to use"* (107 tokens) [55] to see if using a Code node is documented.
- It got a Usage page for the Code node (read 80 tokens) [55] .
- Then it searched *"n8n Code node access multiple inputs from different nodes"* (370 tokens) [56] , and another variant, to find specifics on how a Code node handles multiple inputs.
- Finally it read a page about accessing other nodes' output in code (2310 tokens) [57] .

Through this **search → read → search → read** process, the agent gathered nuanced info: how Merge works, how Code nodes work with inputs, examples of each, etc. All of this was done within one Ref session, which meant no duplicate results and focused reading each time. The token counts in parentheses show that only relevant chunks were read at each step. In the end, the agent can synthesize an answer like "Use a Merge node if you want to combine streams; but if you need custom logic on multiple inputs, a Code node can access them via `$input`... etc." – and crucially, it can cite the n8n docs for those statements. This example illustrates a **deep research pattern**: the AI progressively narrows down the info needed, with Ref enabling it to efficiently fetch each piece. Without such tools, the AI might have either hallucinated or required a human to provide the docs.

4. On-the-Fly Web Search for Coding Issues: Sometimes the question is about a programming problem not covered in official docs (e.g. "Why am I getting this error when using X library?" which might be answered on StackOverflow). In these cases, after trying the doc search, an agent can use `ref_search_web` as a fallback. For instance, if asked about an obscure error message, `ref_search_documentation` might return "No results found" (if the error isn't explicitly in docs). The AI could then switch to `ref_search_web` (if enabled) to search the broader web. That might return a StackOverflow thread or a blog post URL. The agent can then `ref_read_url` that page to get the solution. This multi-source workflow is useful for troubleshooting or "how to" questions that developers commonly google. Ref essentially lets the AI do that googling itself, but still in a controlled, cite-able way (since the results are returned with URLs that can be read and quoted). **Important:** Web search should be used judiciously – the content quality is not guaranteed like official docs. Many teams disable `ref_search_web` by default [58] , preferring the AI to say it didn't find anything in docs (and maybe prompt the user for guidance) rather than blindly pulling from random web content. But for an autonomous agent (like in Cursor or Claude Code), having the web fallback can be a lifesaver when documentation is incomplete.

5. Integration with Developer Tools and Agents: Ref.tools MCP is compatible with a range of AI coding environments. This includes IDE assistants (like Cursor, VS Code extensions, etc.), standalone agent frameworks, and even chat interfaces:

- **Cursor IDE:** Cursor (a popular AI-focused IDE) supports MCP servers. Ref can be added to Cursor either as an HTTP service or via npx. For example, Cursor's settings might include `"Ref": { "type": "http", "url": "https://api.ref.tools/mcp?apiKey=YOUR_API_KEY" }` for cloud mode [59] , or a local config using the `command` approach [60] . Once configured, the Cursor AI will automatically use `ref_search_documentation` when it needs to look up docs (especially if you follow their prompt best practices). Cursor's integration is seamless – it even has UI (the "Inspector") to show MCP calls in action. With Ref in Cursor, you can hover over code and if the AI needs to explain a library usage, it might behind-the-scenes call Ref to get the docs for that function. Many other IDEs or clients integrate similarly (the Ref docs list setups for VS Code, Zed, Vim (Augment), etc. [61] [62]).

- **Claude Code (Anthropic):** Claude's coding assistant (Claude Code or Claude in Code mode) can use MCP servers too. One can integrate Ref so that Claude will call the tools when needed. Claude is known for large context, but using Ref with Claude Code can greatly reduce the context it needs to ingest (since Claude can fetch answers on the fly rather than have everything preloaded). In practice, you'd configure Claude's environment or the proxy (if using something like Goose.ai or an orchestrator) to include Ref's MCP endpoint. The Ref documentation provides guidelines for Claude integration (likely similar to Cursor's). Once set up, you might see Claude Code issuing queries like "SEARCH ref… " in its chain-of-thought when you ask for something involving an API.

- **ChatGPT (OpenAI) Plugins/Deep Research:** OpenAI's tools ecosystem has a notion of "Deep Research" where an agent uses search and fetch actions. Ref is built to align with that pattern [63] . In fact, when Ref detects an OpenAI client, it will alias its tools to the generic names `search` and `fetch` to comply with OpenAI's expected schema [64] [65] . That means if you use Ref with the ChatGPT plugin system (or via an orchestrator that mimics it), the OpenAI model will see tools named just "search" and "fetch" (with appropriate descriptions), making it compatible with OpenAI's requirements for web browsing or research plugins [63] . Practically, you could have a ChatGPT plugin that routes to Ref for documentation; the plugin manifest would point to the Ref MCP HTTP URL. The agent's prompts can then say "use search to find docs…" and ChatGPT would be invoking Ref under the hood. (OpenAI has explicitly signaled that this *search+read pattern is the future of agentic search*, and Ref follows that pattern closely [63] .)

- **Other Agents:** Any agent framework supporting the Model Context Protocol (MCP) can interface with Ref. This includes emerging tools like Smithery.ai, Lyra, and various AI agent orchestration platforms. For example, Lyra (Waylight) listed a "Ref AI Agent" that one-click connects Ref – once added, their AI agents can use `ref_search_documentation` and `ref_read_url` automatically [66] [67] . Another example is integration with **Antigravity or Windsurf** (AI coding assistants) which the docs mention [68] . Essentially, Ref becomes a drop-in module that any AI agent can call when it needs an answer from documentation. This modularity is one reason MCP is gaining traction: you can pair a coding model with specialized tools like Ref for docs, Exa for code search, etc., to cover all bases.

6. When to Use Each Tool (Practical Guidance):

- Use `ref_search_documentation` **first** whenever the query is likely answerable by official docs or your indexed resources. It excels at *API references, library usage, configuration options, function definitions, code examples from docs*, etc. Formulate the query as a natural question or descriptive phrase including keywords (library name, function, error, etc.) – the search uses a combination of semantic and keyword search, so providing context helps [8] . For example, "Node.js fs writeFile usage example" or "Python requests SSL disable verification docs".

- If `ref_search_documentation` returns nothing useful (no results or irrelevant), and you suspect the information might be on the broader web (forums, blogs, etc.), then consider `ref_search_web` (if enabled). This might be the case for *very new libraries (docs not indexed* *yet), obscure error messages, or conceptual "best practice" discussions*. When using `ref_search_web`, treat it like a normal web search – you might include error codes or Stack Overflow style queries. E.g. "error EADDRINUSE node js how to fix". Be cautious: always follow up by reading the page and verifying info, as web content can be user-generated.

- Use `ref_read_url` **for any URL you want detailed content from**. In practice, the URL will usually come from the search tool's output. The search results from Ref include an `overview` (a brief snippet or title) and a `url` for each doc found [69]. The agent should pick the one that looks most relevant and call `ref_read_url` on it. *Do not try to summarize from the overview text alone!* – the overview is often just the first lines of the doc section and not enough for a precise answer. Instead, always read the page to get the full context (Ref will still only return the relevant portion).

- It's generally **not useful to call `ref_read_url` on something that isn't a documentation page**. If you pass a random URL (not returned by search), Ref will still fetch it, but it's intended for doc pages or web pages that relate to your query. For example, you could directly read a known URL (like if the user explicitly says "See X URL"), and Ref will return it in markdown. But the typical flow is search → read. In summary: **Search to get candidate links; Read to retrieve content from those links.**

- **Iteration strategy:** If the first search didn't get exactly what you need, refine the query rather than immediately going to web search or giving up. For instance, if searching "React native image cache" yields nothing relevant, try a more specific query like "React Native FastImage cache example" or a more general one depending on the case. Because Ref won't repeat results, you can run a second query without getting the same junk. This iterative approach (search, read, then possibly search again based on new info or using different terms) is how human developers often search, and Ref enables the AI to do the same.

- **Session resets:** Within one conversation or task, keep using the same session (most MCP clients handle this automatically by reusing the session ID) so that Ref can optimize results. If you intentionally want to "start fresh" (maybe the topic completely changed), you could end the session (in HTTP, by a DELETE to /mcp or closing the tool) and start a new one. This will clear the search history used for filtering results. But generally, you let the session tie to the user's problem scope.

In summary, the workflow patterns with Ref.tools MCP involve an interplay of search and read tools, guided by the agent. It closely mirrors how a developer uses a search engine and documentation site, but it's the AI doing it for you. By understanding when to search vs when to read, and which source (public/private/web) to target, you can get optimal results from Ref.

Best Practices & Patterns

Using Ref effectively requires some care in how you prompt and structure the AI's approach. Here are best practices and proven patterns:

Query Formulation for Documentation Search: When prompting an AI agent that has access to `ref_search_documentation`, **phrase the query in natural language with specific keywords**. Avoid overly generic terms. The Ref team suggests including the relevant technology names (programming language, framework, library, etc.) in the query [36]. For example, instead of asking the agent *"How do I authenticate?"* (too vague), ask *"How do I authenticate using the Dropbox API in Python?"*.

This provides context so the search can pinpoint the right docs. In general, treat the query like a Google search string – **clear and detailed**. The Ref JSON schema even notes the query “should be a full sentence or question” [8], implying the search has semantic understanding. Often, questions (who/what/how) yield good results, but including error messages or function names as needed is fine. Also, if you know the library (e.g. TensorFlow) include its name; if you suspect the answer might be in a specific domain (e.g. a GitHub README), mention keywords like “GitHub” or “StackOverflow” only if necessary. Usually, Ref’s index covers official docs and popular repos, so just naming the library or API is enough.

Iterative Refinement: Don’t expect the first search query to be perfect. A strong pattern is: *issue an initial broad query, then refine based on what you see.* If the search results come back too broad (e.g. many possible pages), the agent can narrow the query by adding more detail. Conversely, if nothing comes up, broaden or try synonyms. For example, if searching “Python pillow image resize function” didn’t help, you might drop “function” or use the library’s older name (“PIL”). The AI can do this autonomously: it has the power to formulate new search queries. Encourage it in prompts to use the content of partially relevant pages to guide the next search – this mimics how a human would find a hint in one page and then search for that term in another context. Ref’s session memory will ensure new queries don’t retread old ground, so refining is efficient [17].

Combine Search and Read in One Workflow: Always pair `search` with `read` in the agent’s reasoning. A common mistake (antipattern) is to search the docs and then try to answer from just the search summaries. The best practice is: *Search, then read, then answer.* The search result overview is not meant to be the final answer; it’s to help choose which link to read. So, prompt your agent that after a `ref_search_documentation` it should almost always call `ref_read_url` on one or more of the returned URLs to gather facts before formulating an answer. This pattern ensures the agent’s answers are grounded in actual documentation text (which you can have it quote or cite). For instance, an agent might say in its reasoning: “I found a relevant page, now I’ll read it for details,” which is exactly what we want. By combining the two tools in a chain, you essentially let the AI dig as deep as needed.

Leverage Session Memory for Multi-Step Queries: As noted, the session keeps track of prior searches and pages read. A good pattern is to **use that context to your advantage**. If you read one page and it mentions a concept you need more info on, you can directly search for that concept next, knowing Ref will not return the same page again but likely the new concept’s page. For example, you read about a class `FooClient` in AWS SDK docs, and it mentions “see Authentication section for credentials”. Next, search for “FooClient authentication” – it should retrieve that specific section or page, not the one you just read. The session memory and filtering enable a kind of guided traversal through documentation space. Encourage the agent (or design your prompt strategy) to use continuity: refer to things it just learned when formulating the next query. This yields a coherent research thread instead of disjointed questions.

Token Budget Management: Although Ref cuts down token usage drastically, you still should manage how much you have the AI read. One best practice is to **target specific sections rather than entire large pages whenever possible**. Ref will chunk out irrelevant parts, but if a documentation page is enormous (say a whole API reference), even the relevant section might be a few thousand tokens. If that’s too much for your application (maybe you’re on GPT-4-8k context or similar), consider splitting the query. For instance, instead of reading the entire “All API endpoints” page for one endpoint, search directly for that endpoint by name. This way Ref returns a smaller chunk. The good news is Ref’s design already does a lot of this for you (returning at most ~5k tokens per read). However, if you ask a very broad question and the agent decides to read 3 different pages of 5k each, that’s ~15k tokens, which might strain some models. So a pattern is: **ask focused questions**. If the user question is too broad, have the AI break it into sub-questions, search those individually, and answer step by step. This aligns with general chain-of-thought best practices and uses Ref in an iterative fashion rather than trying to swallow an entire manual at once.

Setting Up Private Sources Effectively: To get the most out of Ref for your private data, follow these practices:

- **Keep resources relevant and updated:** Only connect repositories that contain docs or code relevant to the questions you expect. The free tier allows a few repos; the team plan likely allows more. If you index a giant monorepo, remember that only its docs (or everything if small) are indexed, so perhaps isolate important docs in a specific repo if you can. For PDFs, upload the latest versions. If you have documentation in Confluence or HTML, consider exporting to PDF or Markdown so you can feed it to Ref. The more comprehensive your private index, the more useful `ref_search_documentation` becomes for internal queries.

- **Use Teams for collaboration:** If multiple developers or multiple AI instances need access to the same private resources, use the Teams feature. This ensures they all share the same index (no need for each person to upload the same PDF) [34] [35]. Team accounts can manage resources centrally, and new team members instantly get those docs in their Ref searches. This is a best practice for scaling Ref usage in an organization.

- **Monitor indexing and sync:** Be aware of the sync schedule (5 min intervals for GitHub) [40]. If you just updated your README and ask the AI about it 1 minute later, it might not find it until the next sync. If something isn't being indexed (perhaps because it exceeded the file limit or size limit), consider breaking it up or using PDF upload. Currently, files like large binaries or extremely long single documents might be skipped or truncated. Check the Ref dashboard for resource status – the dashboard (`ref.tools/resources`) will show your indexed files and any errors. Regularly verify that your key internal docs are indeed searchable.

Deep Research Patterns with Ref: "Deep research" refers to using the AI to autonomously gather information over multiple steps (like an AI agent conducting research to answer a complex question). Ref is practically built for this. Here are patterns to maximize success in deep research scenarios:

- **Start with an overview:** If tackling a broad topic (say "Explain how technology X works"), the agent might first use Ref to retrieve a high-level overview from official docs (maybe the Introduction or Guide). This sets context. Then it can drill down on sub-questions one by one using Ref for each. This structured approach prevents confusion and keeps each query focused. Essentially, the AI constructs an outline and uses Ref to fill in each part from sources.

- **Ask for specific sections by name:** If you (or the agent) know that documentation is structured in a certain way (e.g., "User Guide", "API Reference", "Troubleshooting" sections), you can directly search for those terms. For example: "XYZ library troubleshooting error codes" or "XYZ API authentication section". Often, docs have pages titled in a way that Ref's index will pick up. This gets you straight to the relevant section instead of scanning multiple pages.

- **Combine multiple tools when needed:** Ref can be used alongside other MCP tools. For example, some users pair Ref with Exa (another code-oriented search) [70]. The pattern could be: use Ref to get official docs explanation, use Exa to search actual code examples on GitHub, then compare. If your agent has both, knowing when to use which is key: **Ref for official knowledge, Exa (or others) for community examples or source code**. Another example, if an agent has a "calculate" tool or a sandbox, it could even fetch an algorithm from docs via Ref, then test it in code. So, best practice is to let each tool play its role – Ref is your documentation oracle, so use it whenever factual or reference info is needed.

- **Validate with multiple sources:** If time permits, a very thorough agent might search more than one source to cross-verify. For instance, search library docs via Ref, and if something is unclear, search the web to see if there are known issues or discussions. This can catch cases where official docs are incomplete. However, this should be balanced with efficiency and avoiding unnecessary calls.

Prompting Style for Agents: When writing system or user prompts for an AI that will use Ref, instruct it clearly on the usage of these tools. For example: *“If the user question involves an API, first use ref_search_documentation to find the relevant official documentation. Then use ref_read_url to read the details, and incorporate those into your answer with references.”* Encourage the agent to quote the docs (Ref returns content as markdown text). A good pattern is having the AI provide answers like, *"According to the official docs, doing X requires Y 【source】."* This not only gives the user confidence but also leverages the actual text retrieved. Since Ref returns markdown-friendly content, the agent can often copy a code snippet or a definition directly from the tool output into the final answer.

In summary, best practices with Ref center around *asking the right questions* and *following through with reading*, iterating as needed, and managing context. By following these patterns, you get precise, up-to-date information injected into your AI’s responses with minimal fuss.

Antipatterns & Pitfalls

While Ref.tools is powerful, there are some common pitfalls and failure modes to be aware of. Avoiding these antipatterns will lead to much smoother use:

1. Skipping the Read Step (or Answering from Memory): A major antipattern is when an agent does a documentation search but then attempts to answer **without calling** `ref_read_url` on any result. This defeats the purpose of Ref – the agent ends up guessing or relying on its training data (which could be outdated or wrong). For example, if the AI searches for “Django filter queryset by date” and gets a URL, but then *doesn’t read it*, it might hallucinate an answer like “Use filter(date__exact=...)” which could be incorrect. Always ensure the agent follows through by reading the content. Many MCP clients have guardrails or few-shot examples to enforce this, but if you notice answers coming without direct quotes or with uncertainty, it could be the agent not utilizing the read step properly.

2. Using Web Search for Everything: Over-reliance on `ref_search_web` is a bad pattern. If an AI constantly goes to web search even for things that have official docs, you’ll get suboptimal results: web content might be less reliable, and often the official docs (accessible via `ref_search_documentation`) have the answer with more authority. This can happen if the agent’s prompt or logic isn’t nudging it toward documentation first. It might think “search the web” whenever it’s unsure. To avoid this, emphasize in instructions that the documentation tool should be primary. Only fall back to web search when documentation yields nothing or when explicitly looking for community answers or tutorials. A specific pitfall is if `ref_search_web` returns something like a forum page with a lot of irrelevant chatter – the agent might waste time reading through it. That not only uses tokens but could confuse the answer. It’s often better to either refine the doc query or reformulate the question than immediately scraping random web pages. In short, *web search is powerful but use it sparingly* – don’t treat it as a first resort for things that likely have docs.

3. Vague or Under-specified Queries: If you feed Ref a very vague query, you'll either get no results or too many broad results. For instance, searching "permissions" alone will not be useful (permissions for what?). Similarly, a query like "Java error" is far too general. The pitfall here is expecting Ref to magically know context that wasn't given. This often happens if the user's question to the AI was generic and the agent just forwards it to search. The agent should be trained (or instructed) to add context – e.g., if the conversation is about AWS S3 and the user asks about "permissions", the agent's search query should be "S3 bucket permissions policy documentation" rather than just "permissions". Not providing enough detail will produce poor search results, which then leads to either "No results found" or irrelevant pages, wasting time. The fix is the opposite of the best practice above: always include key context in queries. If you catch your agent searching for single-word terms or extremely short phrases, that's a red flag.

4. Chasing the Wrong Lead (Tunnel Vision): Sometimes the first search result might seem on topic but isn't actually what you need. An antipattern is when the agent locks onto an initial result and keeps reading or searching around it even if it's not yielding an answer. For example, maybe the agent searched a term that led to a library with a similar name but it's the wrong one, and it keeps querying within that library's docs. Or it reads one page, doesn't find the answer, but instead of searching a new topic it keeps reading other sections of the same page or site out of stubbornness. This can happen if the agent doesn't properly analyze the results. A human would realize "hmm this page isn't relevant, I should backtrack," but an AI might not unless guided. The pitfall is wasting tokens on a dead-end. To avoid this: encourage strategies like *checking the overview text of results* and picking the result that looks most relevant. If after reading one page, the answer isn't there, the agent should try a different search query altogether, not necessarily just a different section of the same doc (unless it's very sure the answer is somewhere on that site). Essentially, avoid **tunnel vision** on a single documentation source if it's not paying off – pivot to a different library's docs or to web search if needed.

5. Missing Documentation Coverage: Ref's index is extensive but not infinite. There may be cases where a particular library or framework's docs are not (yet) indexed. In such cases, `ref_search_documentation` might consistently return nothing useful. An antipattern would be to keep trying the same search expecting a different result. If you suspect something isn't covered, consider alternatives: check Ref's docs site for a "Request Docs" link (they allow users to suggest adding new sources [71]). In a live scenario, the agent could gracefully handle it by saying "I'm not finding any official documentation via Ref" and maybe attempt a `ref_search_web` or ask the user for more info. But hammering the doc search repeatedly is a waste. One example mentioned by users: early on, some found that ref.tools didn't have certain niche libraries indexed, leading to "hallucinated documentation" when the AI tried to answer anyway [72]. The developer actively solicited feedback in such cases [73]. So be aware: **if Ref yields no docs, don't force it** – either pivot to web or acknowledge the gap. (The good news is the index improves over time, and you can add your own sources. But always handle "No results found" gracefully.)

6. Potential for Hallucination if Misused: If the agent is not properly constrained, there's a risk it might hallucinate content that *sounds* like docs. For instance, if it doesn't find something, a poorly designed agent might fabricate a plausible answer with a fake citation. This is obviously a serious pitfall. To avoid it, the agent's prompt should enforce that it only uses information from `ref_read_url` outputs when answering technical questions. One should double-check that any answer content from the AI actually came from a source (Ref returns real docs text). In a debugging scenario, if you see the AI giving an answer with details but there's no corresponding content in the retrieved docs, it's hallucinating – possibly because it didn't properly use the tools or the tools failed. The fix is tightening the loop: e.g., have the agent show the retrieved content in its reasoning (some devs do a "show me the content you got" hidden prompt to ensure it's using it). In terms of the MCP server itself, it has a form of prompt injection protection via Centure.ai [63], but that's more for malicious content. Hallucination is more on the AI's side – just remember that **Ref gives you sources, but the AI must decide to trust and use them**.

7. Configuration/Setup Pitfalls: On a more practical note, pitfalls can occur if you misconfigure the server. For example, forgetting to set the `REF_API_KEY` will cause every search to fail (the agent might see only the error message) [46] . Or using the wrong case (it's all caps environment). Ensure your key is verified – Ref requires verifying your email on signup before the key works, otherwise you'll get a 401 "verify your email" message [74] . Another pitfall is not updating the MCP server package – if you installed an old version, you might miss new features like `ref_search_web`. Always keep `ref-tools-mcp` updated (they do monthly updates with new features and fixes). If using Docker, note an issue: the Docker image as of late 2025 had a hardcoded transport mode issue [75] – meaning you might have to explicitly set `TRANSPORT` env if using the Docker MCP gateway. Keep an eye on the GitHub issues for such bugs (there were open issues about Docker and about certain client integrations like Gemini CLI not working smoothly [76]). Being aware of these ensures you don't spend hours wondering why it's not working when it's an environment quirk.

8. Self-Hosting Expectations: A common question is whether you can self-host Ref fully (so that it wouldn't rely on the ref.tools API). Currently, **Ref is not a fully offline solution** – the MCP server code is open, but it calls the proprietary API and uses Ref's cloud index [50] [77] . If someone tries to run it without an API key or without internet access, it won't function. Trying to bypass this (like pointing REF_URL to a local endpoint without an index) is not supported. So the pitfall is assuming you have all the data by just running the npm package. If your use case requires completely local documentation search (for example, in a highly secure environment with no external calls), Ref.tools might not fit unless the team introduces an on-prem solution or allows loading a custom index. There is an open issue (#10) about self-hosting [78] – likely the answer is that it's not possible at the moment. So plan accordingly: you need internet access and an active subscription to use Ref. If those go down, have a backup plan (maybe a static vector index of your most critical docs as a fallback, or Context7 as a plan B, etc.).

By being mindful of these pitfalls, you can avoid common failure modes. In essence: ensure the agent truly uses Ref (search+read) for answers, don't misuse the tools on irrelevant queries, and handle the cases where Ref can't find something. With these in check, you'll maintain the high quality and efficiency that Ref is meant to provide.

Community Insights

Ref.tools MCP has garnered attention in the developer and AI agent community. Here are some insights, feedback, and tips gleaned from real users and discussions:

Ref vs Context7 – User Experiences: Many developers have tried both Ref and Context7 (another popular docs search server) and reported their comparisons. One common sentiment is that **Context7 tends to inject a lot of tokens** (preloading large doc snippets) which can overwhelm the model, whereas **Ref delivers more targeted information** [79] . In a Hacker News discussion, a user mentioned Context7 had a "very low signal/noise ratio" and that switching to ref.tools gave *"much more targeted docs."* [79] . This aligns with Ref's design goal of minimal context. On the flip side, there were reports early on that Ref's results could sometimes miss the mark or "hallucinate" if something wasn't indexed [72] . A couple of users noted that in their tests, Ref didn't find what they needed and the AI ended up with incorrect info. The developer of Ref (Matt) actively responded to such feedback, asking for examples and noting that new search quality improvements were being rolled out to address those cases [73] . This shows the service is under active development and improving based on community input. Overall, those who value fewer tokens and more agent-driven search lean towards Ref, while some who just want a quick injection might have stuck to Context7. But as of late 2025, many in the MCP community recommend trying Ref first for documentation needs, citing its efficiency and broader feature set.

Token Savings in Practice: A Medium article by a user vividly titled "I Reduced My Context Window By 97% Using These Two MCPs" detailed how combining Ref.tools and another tool (Exa) drastically cut down their token usage [28] [70]. The author described how they used to shove entire docs (like Tailwind CSS docs, etc.) into the prompt (tens of thousands of tokens) and the model's performance suffered ("my AI got dumb… context rot" [80]). By switching to an approach where the AI only pulls in exactly what it needs via Ref (for docs) and Exa (for code search), they preserved model performance and saved money. They shared an example of using only 2.8k tokens instead of 100k, which is a **1.4% context size** of the original, calling it "the 1.4% solution" [81]. This anecdote backs up Ref's claims with a real scenario. It highlights that beyond just cost, *model quality improved when extraneous context was removed*. So community consensus is that focusing the context via tools like Ref is a best practice for agent development.

Feature Requests & Roadmap Hints: Users have been actively suggesting features. A notable request is the ability to **add arbitrary URLs or documentation sources** to Ref's index. Currently, you are limited to GitHub repos and uploaded files, and public docs that Ref's team has indexed. Issue #32 on the GitHub repository, "Support for adding external document URLs to the index," is one such feature ask [42]. While not implemented yet, this could be on the roadmap – allowing users to point Ref at, say, an internal Confluence or a website, and have it crawl that. Another roadmap hint is the **GitHub App for organizations** (mentioned in docs as "on the roadmap" [41]) to make connecting org repos easier. There's also mention of an **Inspector tool** (in DXT and UBOS descriptions) which provides a visual interface to test and monitor the MCP server [82] [15]. Some community members in Discord/Reddit have shown interest in this "Ref Inspector", which presumably lets you see the queries your agent is making and the responses (great for debugging agent behavior). While not a direct feature of the MCP API, it's part of the ecosystem the Ref team is building for users. So, expect improvements in tooling around Ref – easier setup, monitoring, and more data source integrations.

Tips from the Ref Team: The Ref team has been responsive on platforms like Reddit and Discord. A few nuggets from Matt (the developer):

- *On difference from Context7:* He explained that **Context7 returns a fixed ~10k tokens per library**, whereas Ref *"has the goal of finding exactly the tokens the agent needs"*, saving on average 5.5k tokens per request (and sometimes 95% of tokens) [22] [27]. He also highlighted that Ref supports **private repos, PDFs, and on-the-fly web scraping (like a tool called Firecrawl)** in the base version [33] – basically selling the point that you get more sources without extra charges. This was a convincing point for those on the fence about the subscription cost: you get double the queries and more capabilities [30].

- *On search quality:* When users pointed out missing results, Matt encouraged them to reach out (matt@ref.tools) with what they searched [83]. He noted that new improvements were coming and indeed the service has seen frequent updates (the docs site even has monthly changelogs). This openness suggests that if you find Ref failing for a particular library or type of query, letting the team know could result in that doc being indexed or the algorithm being tuned. In one Reddit thread, he even invited someone to try again after an update that week improved search relevance [84].

- *On cost-effectiveness:* Matt acknowledged that while Ref is a paid add-on, if you use AI coding help a lot, **Ref likely pays for itself in saved API costs** [85] [86]. He didn't provide hard eval metrics for answer quality, only token usage, but logically less noise means potentially better answers [87]. He suggested that even if you don't use it heavily, the improved code generation from having the right context can be worth it [88] (less hallucination, less trial and error).

Success Stories & Use Cases: Some users on Reddit have described building complex agents with Ref at the core. For example, there was mention of integrating Ref and another server (Exa) to have an AI **build n8n workflows** by itself [89]. By prompting Claude with instructions to use the reference tools (Ref for docs, Exa for code examples), the user was able to get the AI to assemble working automation workflows. This showcases how Ref can empower agents to do non-trivial tasks – the AI could look up each node's usage in n8n's docs and properly configure it. Another user wrote a Medium piece listing "Top 5 MCP Servers" where Ref was dubbed *"The Efficiency Expert"* – they noted that *Context7 "burns more tokens and gives generic answers," whereas Ref returns precise context and keeps the model focused* [90]. In AI circles, that kind of endorsement underscores that Ref isn't just theoretically better, but in practice, people are seeing improved results.

Community Support: There is a Discord (likely the MCP or Ref discord) and an r/mcp subreddit where people discuss these tools. New users often ask for setup help – a common tip is to use the hosted version via HTTP for ease. If someone has trouble, the fix is often ensuring the API key is set and verified (a few had issues not realizing they needed to verify their email). The Ref documentation suggests contacting help@ref.tools for support [91] [92]; anecdotal evidence shows they do respond. Also, since Ref is part of the larger MCP ecosystem, community-driven marketplaces like Smithery, FastMCP, etc., list Ref and sometimes provide install stats and uptime info. For instance, Lyra's site showed 10,000+ installs of Ref [93], and DXT.so lists a 54.6% uptime (that might be an artifact or a specific stat, possibly indicating maybe the HTTP service uptime – which seems low and could be a reporting quirk) [94]. In any case, thousands of users have tried it, which means you can often find someone who has encountered your issue before.

Competitors and Combination: The community often mentions using Ref alongside others like **Exa** (for code search), **Tavily** or **Firecrawl** (other search tools), etc. [27]. Some advanced setups involve chaining multiple MCP servers for an AI (search documentation with Ref, search StackOverflow with another, etc.). Ref is generally seen as complementary to such tools – focusing on official and user-provided docs, whereas others might search open source code or Q&A sites. A Hacker News user mentioned a new server "Nia" that gives more docs and better integration; in that thread, Ref was discussed as well [95]. It's a space with active innovation. The consensus seems to be that **Ref is a solid all-around documentation tool** for coding agents and has set a high bar that new entrants compare to.

In conclusion, community feedback reinforces that Ref.tools MCP is highly valued for its token efficiency and comprehensive approach to documentation (public and private). When set up correctly, it provides a noticeable boost in an AI assistant's capability – less guessing, more quoting the manual. As one user humorously put it, using Context7 vs using Ref is like *"screaming 47 Wikipedia articles at the model vs calmly handing it the paragraph it actually needs"* (a paraphrase of the Medium article [96]). The community is rallying around these "tools-not-prompts" approach to give AI better grounding, and Ref is at the forefront of that movement for coding assistants. With ongoing improvements and an active developer, it's likely to remain a go-to solution for keeping AI's knowledge up-to-date and accurate.

Sources: The information above is drawn from Ref's official documentation and blog [21] [30], the Ref MCP GitHub repo [26] [17], as well as community discussions on Reddit and Hacker News where Ref's creator and users have shared experiences [27] [97]. These real-world insights complement the technical details, illustrating how Ref performs in practice and what best practices have emerged around it.

[1] [12] [13] [15] [82] Ref Tools - MCP Server by ref-tools
https://dxt.so/mcp-server/coding-agents/ref-tools-mcp

[2] [16] [58] Ref – FAQ | MCP Marketplace
https://ubos.tech/mcp/ref/faq/

[3] [17] [23] [25] [26] [51] [52] [55] [56] [57] [59] [60] GitHub - ref-tools/ref-tools-mcp: Helping coding agents never make mistakes working with public or private libraries without wasting the context window.
https://github.com/ref-tools/ref-tools-mcp

[4] [5] [36] [37] [43] [44] [45] [46] [50] [53] [54] [64] [65] [69] [74] [77] index.ts
https://github.com/ref-tools/ref-tools-mcp/blob/53392c95db2c066acce677c9192df7a62e6e68d9/index.ts

[6] [7] [8] [9] [10] [14] tools.json
https://github.com/docker/mcp-registry/blob/390d1f8fd6a2ee4d1d7597933192d820698fb5de/servers/ref/tools.json

[11] [66] [67] [93] [94] Ref AI Agent - General Agent for Lyra | Ref MCP
https://www.trylyra.com/agents/%40ref-tools/ref-tools-mcp

[18] [19] [20] [21] [24] [29] [30] [31] [32] [48] [63] Ref vs Context7 - Ref
https://docs.ref.tools/comparison/context7

[22] [27] [33] [85] [86] [87] [88] How Ref takes advantage of MCP to build documentation search that uses the fewest tokens : r/mcp
https://www.reddit.com/r/mcp/comments/1mc9uvw/how_ref_takes_advantage_of_mcp_to_build/

[28] [70] [80] [81] [96] I Reduced My Context Window By 97% Using These Two MCPs | by Yagyesh Bobde | Medium
https://bobde-yagyesh.medium.com/i-reduced-my-context-window-by-97-using-these-two-mcps-2f5722bbe61f

[34] [35] Resources Overview - Ref
https://docs.ref.tools/resources/overview

[38] [39] [40] [41] GitHub - Ref
https://docs.ref.tools/resources/github

[42] [71] [75] [76] [78] GitHub · Where software is built
https://github.com/ref-tools/ref-tools-mcp/issues

[47] Top Context7 MCP Alternatives : r/mcp
https://www.reddit.com/r/mcp/comments/1p06582/top_context7_mcp_alternatives/

[49] Case Studies - Ref
https://docs.ref.tools/examples/case-studies

[61] [62] [68] Tools - Ref
https://docs.ref.tools/mcp/tools

[72] [73] [79] [83] [84] [95] [97] Context7 injects a *huge* amount of tokens into your context, which leads to a v... | Hacker News
https://news.ycombinator.com/item?id=44681524

[89] Best AI model and prompt strategy for generating valid n8n workflow ...
https://www.reddit.com/r/n8n/comments/1q21y08/best_ai_model_and_prompt_strategy_for_generating/

[90] Top 5 MCP Servers Tested and Recommended - Medium
https://medium.com/@harpalonsoftware/top-5-mcp-servers-tested-and-recommended-b658cbf49fe6

[91] Cursor - Ref.tools
https://docs.ref.tools/install/cursor

92 Gemini CLI - Ref.tools
https://docs.ref.tools/install/gemini-cli