

# AI-Assisted Research Workflows with Claude Code & MCP

(One Opinionated and Suboptimal Setup)

---

Chris Schmitz

13.2.26

# Session Overview

What this session covers:

- A live demo of a few workflows in my workspace
- What Claude Code and MCP are, and why they matter
- How my workspace is structured and the design principles behind it
- Brief intro to each of four tools: Obsidian, Zotero, LaTeX/Overleaf, Google Docs
- How you manage context, set rules and workflows
- Live tutorial of setting most of this up

## Session Overview — Prerequisites

**To follow along with the first half, you'll need:**

- VS Code
- Claude Code (requires an Anthropic API subscription) — if you don't have it installed but do have the subscription, that's fine

**To follow the second half, you'll also need:**

- Accounts/installations for each tool you want to connect (Obsidian, Zotero, Google Cloud project for Docs, GitHub and Overleaf for LaTeX)
- For Zotero MCP: Python and uv
- For Obsidian and Google Docs MCP: Node.js (which includes `npm` and `npx`)

The second half is primarily a walkthrough and you shouldn't try to build along live. Claude and I wrote a detailed setup guide so you can replicate everything afterwards.

## Session Overview — What You Will Leave With

- Claude Code running in VS Code
- A workspace folder with the right structure, spec files, and CLAUDE.md
- A `.mcp.json` with placeholder entries ready for your connections
- Understanding of how to set up the integrations you need
- A mental model of the architecture/design of an agent context for knowledge work

# Agenda

## 1. Demo

2. Claude Code & MCP

3. Workspace, Design & Architecture

4. The Four Tools

5. Setting up CLAUDE.md, Specs, and MCP Servers

6. Summary, Next Steps, Getting Started

- Democratic deliberation project
- Obsidian note classification
- A google docs demo if time

## Key Takeaways — What Is vs. Isn't Important

### Important:

- Workspace and context management
- MCP: exposing tools to an agentic system

### Not important:

- Claude Code specifically vs. any other agentic system
- Exact setup of integrations — this will get consumerized pretty soon
- My exact workspace setup, which is definitely not optimal and changes daily

(That said, at the end there's detailed setup instructions + a downloadable template if you just want this exact workspace.)

# Agenda

1. Demo

**2. Claude Code & MCP**

3. Workspace, Design & Architecture

4. The Four Tools

5. Setting up CLAUDE.md, Specs, and MCP Servers

6. Summary, Next Steps, Getting Started



# What Is Claude Code?

- Anthropic's CLI/VS Code tool for agentic AI assistance.
- It can read & write files, run shell commands, and call external tools.
- This is equivalent to roughly anything you can do in a folder on your computer, plus whatever the connected tools allow it to do.
- *Our goal today is creating one folder and set of tools in which that's incredibly useful.*

## What Is Claude Code? — Portability

**It is not the only option.** Any MCP-compatible client works: Cursor, Windsurf, ChatGPT desktop, Gemini CLI, Claude Cowork. . . The workspace we build today is portable across all of them, including all integrations. If you want to switch, just stop using Claude Code and start using something else.

## What Is Claude Code? — Live

- Open VS Code
- Create a folder (e.g., `research-workspace/`)
- Open the VS Code terminal (`Ctrl+``, drag up from bottom of screen, or `View → Terminal`)
- Quick terminal orientation:
  - `cd folder-name` — move into a folder
  - `cd ..` — move up one level
  - `ls` (Mac/Linux) or `dir` (Windows) — list files
  - `mkdir folder-name` — create a folder
  - You can also just use the VS Code file explorer for most of this
- Install the Claude Code extension from the extensions tab in the left sidebar and authenticate
- Open the Claude Code panel (`Ctrl+Shift+P → "Claude Code: Open"`, or click the orange button)
- Verify it's working: type "Hello, what directory are we in?"

## What Is Claude Code? — Two Things to Know

- It will *ask for permission* for anything it tries to do by default. With Shift+Tab (in the VS Code extension), you can let it run automatically. Even then you'll have to allow each new type of action once.
- You can highlight parts of files to give Claude Code as context. If you don't, it reads whatever file you have open. You can also hide that — in that case it starts with just your CLAUDE.md.

That's the basics! Play around with CC: make it create folders, paste some text into a file in this folder and ask for comments in it, etc.

# What Is MCP?

MCP (Model Context Protocol) is an open protocol that connects AI agents to external tools and data.

**Architecture:** **Client** (Claude Code, Cursor, etc.) ↔ **Server** (Obsidian, Zotero, Google Docs, ...)

Each server exposes a set of tools the client can call.

# What Is MCP? — Mental Model

- You have some sort of program (google docs, obsidian, ...) that has some sort of programmatic interface (analogous to a website/GUI for humans)
- An MCP server wraps that interface into a list of callable tools for Claude
- These tools have intuitive names and functions, e.g. the google docs server has a tool called `replaceDocumentContentWithMarkdown`
- Claude sees this list of tools and chooses to use them if relevant while completing a task
- MCP is a standard for how those tools are listed and exposed to models.

There is one short file in each workspace called exactly `.mcp.json` where you list every server Claude has access to. These servers set what tools to offer.

# What Is MCP? — Modularity

## Modularity is the key insight:

- Add a server → the agent gains new capabilities.
- Remove a server → nothing else breaks.
- Swap the client (e.g. Claude Code to Codex) → your servers and config still work.
- MCP servers have to be built by someone, and usually you can find them online. I did for all of mine.

Example: if you use **Notion** instead of Obsidian, just swap the Obsidian MCP server for a Notion one. Everything else stays.

# Agenda

1. Demo
2. Claude Code & MCP
- 3. Workspace, Design & Architecture**
4. The Four Tools
5. Setting up CLAUDE.md, Specs, and MCP Servers
6. Summary, Next Steps, Getting Started



## My Workspace (1/2): Setup

```
research/
+-- CLAUDE.md          <- Agent instructions -- read every conversation
+-- .mcp.json          <- MCP server config -- read on startup
+-- LATEX_SPEC.md      <- LaTeX conventions -- only for LaTeX tasks
+-- OBSIDIAN_SPEC.md   <- Note classification -- only for Obsidian
+-- PROJECTS_SPEC.md   <- Project entry format -- only when updating
+-- PROJECTS.md        <- Project registry
|
+-- projects/          <- Papers and articles, mostly LaTeX
|                      (each own git repo -> GitHub -> Overleaf)
+-- zotero/exports/    <- Read-only Zotero library exports (.bib, .json)
|
+-- implementation/    <- Custom scripts & services (not relevant today)
```

## My Workspace (2/2): Connected Tools

### Connected tools (via MCP):

- **Obsidian** — knowledge base: meeting notes, paper summaries, project planning, idea sketches
- **Zotero** — reference manager: ~500 papers, semantic search, annotations, citation export
- **Google Docs** — reading and writing docs and sheets incl. comments

### Built-in (no MCP needed):

- **LaTeX / Overleaf** — Claude Code reads/writes `.tex` and `.bib` files directly, uses git to sync with Overleaf
- **Git / GitHub** — built into Claude Code's shell tools

## Design Principles (Most important slide!)

1. **One workspace.** All context for all projects. I could set up a separate workspace for each project. Currently I think being able to draw on stuff across projects is better (e.g., “I’m pretty sure I cited some explainability stuff in my oversight paper, add those same citations here”), but maybe I’m wrong.
2. **Good guidance.** These agents are hungry for context and will, if not steered, read all kinds of stuff, burning through your rate limit or budget.
  - Modular specs keep them on track.
  - A source-of-truth list of projects is helpful for quick context. The most important part is consistent project identifiers, e.g. `flooding`, used everywhere.
3. **Everything version-controlled separately.** All projects/files have individual version tracking: (a) in Google Docs, (b) in a git repo, (c) in Obsidian, or (d) in Zotero. No files “live” primarily in the research workspace, and the workspace is not version-controlled as a monolith.

# Architecture

- CLAUDE.md contains high-level guidance. CC reads this first every time (this is a default)
- **Spec files** encode domain-specific conventions. They make agent behavior consistent, and are only read when a task requires them. CLAUDE.md points to these as necessary (I did this)
- .mcp.json configures which external MCP tools are available. (this is a default)
- **Separation of concerns:** Obsidian vault is separate (MCP only). Zotero exports are read-only; changes to Zotero go through MCP. Each LaTeX project is its own repo.

## Workflow loop:

1. Tell the agent what you need (natural language).
2. Agent reads relevant specs, uses MCP tools, edits files.
3. You review, iterate, push to GitHub/Overleaf/Google Docs when ready.

### Option A — do it yourself:

1. Create the directories (terminal or VS Code file explorer):  

```
mkdir projects  
mkdir zotero  
mkdir zotero/exports  
mkdir implementation
```
2. Create CLAUDE.md in the root — describe what's here, your key rules, and a table pointing to spec files.
3. Create at least one spec file (e.g., OBSIDIAN\_SPEC.md) — describe your conventions for that tool.
4. Create PROJECTS.md — add one project entry with tag, status, description.
5. Create .mcp.json — we'll fill this in next.

**Option B — let Claude do it.** Copy-paste this into Claude Code:

Set up this folder as a research workspace. Create:

1. Directories: `projects/`, `zotero/exports/`, `implementation/`
2. A `CLAUDE.md` with basic agent instructions for a research workspace that uses Obsidian for notes, Zotero for references, LaTeX for papers, and Google Docs for collaborative writing. Include key rules and a table of specs pointing to the spec files below.
3. An `OBSIDIAN_SPEC.md` with a basic vault structure, note classification decision tree, and frontmatter template.
4. A `LATEX_SPEC.md` with conventions for academic writing and a citation workflow.
5. A `PROJECTS_SPEC.md` defining the format for project entries.

# Agenda

1. Demo
2. Claude Code & MCP
3. Workspace, Design & Architecture
- 4. The Four Tools**
5. Setting up CLAUDE.md, Specs, and MCP Servers
6. Summary, Next Steps, Getting Started

## Tool: Obsidian — Local-First Knowledge Base

Markdown-based note-taking app. Notes, meeting logs, paper summaries, project planning.

**MCP server:** `@mauricio.wolff/mcp-obsidian`

- Zero dependencies, no Obsidian plugins needed
- Requires: Node.js / npm
- Install & run: `npx @mauricio.wolff/mcp-obsidian@latest /path/to/vault`

**Agent capabilities:** read, write, search, move, and delete notes; manage frontmatter and tags; batch operations; vault statistics.



## Tool: Obsidian — How I Use It

- I don't make good use of Obsidian's core feature, note interlinking. Seen some wizards with it though.
- I like it as a *Zettelkasten*: just create a new note and start typing, think about filing and context later.
- A classification decision tree in `OBSIDIAN_SPEC.md` lets the agent know how to handle each note type. I occasionally just go “classify all my obsidian notes according to the instructions.”
- AGENT-TODO markers: I leave tasks in notes, marked by “AGENT-TODO”, that the agent executes during vault cleaning — updating todos, searching for stuff someone mentioned, cleaning up messy meeting notes, etc.

## Tool: Zotero — Reference Management

Open-source reference manager. Stores papers, metadata, annotations, full text.

**MCP server:** `zotero-mcp`

- Python-based, uses the cloud API or the local one (communicates with Zotero on your PC). Not much difference: Zotero syncs your library to the cloud anyway.
- Requires: Python 3.10+ and `uv`
- Install: `uv tool install "git+https://github.com/54yyu/zotero-mcp.git"`
- Credentials: API key + library ID from [zotero.org/settings/keys](https://www.zotero.org/settings/keys)

**Agent capabilities:** keyword search, advanced multi-criteria search, **semantic search** (AI embeddings); read metadata, full text, annotations; manage tags, collections, notes.

## Tool: Zotero — How I Use It

- Everything I want to cite and most stuff I read goes in Zotero — mostly academic, but also substacks, reports, etc.
- I try to read and annotate in Zotero, because those annotations are also accessible via MCP.
- Zotero is basically a wrapper around a .bib file, which is great because those can be used in LaTeX.
- I have an export of my entire Zotero library in `zotero/` as fast local context for the agent, in addition to the MCP access.
- What I actually use it for: (a) adding references to documents/papers, and (b) having Claude Code clean up my Zotero — tag and file papers, etc.
- **Semantic search** is the killer feature: “find papers about democratic legitimacy of algorithmic decision-making” works even if those exact words aren’t in any title.
- I have a `check_citations.py` script to avoid hallucinations: after every addition of citations to a LaTeX doc, Claude Code runs that deterministic script to confirm

## Tool: LaTeX & Overleaf — Academic Writing

LaTeX for typesetting; Overleaf for collaborative cloud editing.

**No dedicated MCP server needed** — Claude Code's built-in file editing + git tools handle everything.

Requires a local LaTeX distribution (e.g., TeX Live, MiKTeX) to compile locally (sort of a mess), or just compile on Overleaf.

**Workflow:** Each paper is its own git repo under `projects/`. Git syncs to GitHub → Overleaf pulls from GitHub. Always fetch/pull before starting work.

**Agent capabilities:** Mostly just editing files. The value is in all these files being in one context.

## Tool: LaTeX & Overleaf — How I Use It

- Before, I would write in Overleaf (google docs for latex basically). Now, I write locally
- `LATEX_SPEC.md` encodes writing conventions, build instructions, citation workflow
- Each project has its own `.bib` (curated), distinct from the full Zotero export

## Tool: Google Docs — Collaborative Documents

**MCP server:** `google-docs-mcp`

- Node.js-based
- Requires: Node.js / npm, a Google Cloud project with OAuth credentials
- Most complex setup of the four (Google Cloud Console → new project → enable Docs & Drive APIs → OAuth consent screen → create credentials → download JSON), but instructions in the repo are really good.

**Agent capabilities:** read documents (plain text, markdown, or raw JSON with character indices); insert, delete, and replace text; apply formatting; manage comments; insert tables and images; work with Google Sheets.

## Tool: Google Docs — How I Use It

**Hard-won lesson:** `replaceDocumentWithMarkdown` wipes all comments. For existing docs, always use surgical tools (`insertText`, `deleteRange`, `applyTextStyle`).

### How I use it:

- Not really established yet, but I don't think I like using Google Docs as a main working location.
- A hard limitation is the
- Instead, I create **local markdown files** that I iterate on, then push changes to Google Docs when I'm ready. Google Docs are usually basically just `.md` files anyway.

## Sidebar: Get comfortable with markdown!

- Markdown files are everywhere: the `claude.md` and specs, every obsidian note. google docs are basically also markdown files (support copy from/paste to markdown)
- Markdown files have incredibly intuitive formatting.
- It's useful working on stuff right in this folder, where Claude has access to all context and can make line-level edits.



# Agenda

1. Demo
2. Claude Code & MCP
3. Workspace, Design & Architecture
4. The Four Tools
- 5. Setting up CLAUDE.md, Specs, and MCP Servers**
6. Summary, Next Steps, Getting Started

## Managing Context — CLAUDE.md and Specs

The agent is only as good as the instructions you give it. But context is expensive — every token of instruction costs time and money, and too much context makes the agent *worse*, not better.

**Principle: keep context minimal and layered.**

- CLAUDE.md is loaded automatically on every conversation. Keep it short: rules, pointers to specs, workspace layout.
- Spec files (LATEX\_SPEC.md, OBSIDIAN\_SPEC.md, etc.) contain domain-specific detail. The agent reads them *only when needed*.
- PROJECTS.md is a lightweight registry — just enough for the agent to know what exists and what's active.

# Managing Context — Why This Matters

## Why this matters:

- If you put everything in CLAUDE.md, every task pays the cost of reading everything.
- If you split into specs, a Zotero task only reads CLAUDE.md + the relevant spec. Much cheaper, much more focused.

## What goes in CLAUDE.md:

- Who the agent is and what this workspace is for (1–2 sentences)
- Key rules (never fabricate citations, never overwrite exports, etc.)
- Table of specs with what each covers
- Workspace layout summary

### What goes in specs:

- Detailed conventions (file naming, frontmatter format, citation workflow steps)
- Tool-specific instructions (how to build, how to search, what to avoid)
- Decision trees and templates

## Configuring MCP Servers

`.mcp.json` is a list of MCP servers that lives in your workspace root. Claude Code reads it on startup. Each server entry has:

- `command` — the executable to run (e.g., `node`, `uv`, `npx`)
- `args` — command-line arguments
- `env` — environment variables (API keys, config values)

Servers start automatically when Claude Code opens the workspace. But this assumes they're installed!

Installing a new MCP server can be a bit annoying, but instructions are usually great. You almost always just need two things installed:

- **Node.js** (which includes npm and npx) — <https://nodejs.org/>
- **Python + uv** — <https://www.python.org/>, <https://docs.astral.sh/uv/>

Then find the MCP server's GitHub repo and follow the detailed instructions there. That's it.

## Adding an MCP Server — Step by Step

This pattern applies for any server including the ones I use. Using Zotero as the example:

1. **Find it.** Search “zotero MCP server” → find the GitHub repo.
2. **Check prerequisites.** Zotero MCP needs Python 3.10+ and uv.
3. **Install the server.** This is just code that runs locally whenever Claude Code accesses the tool. Follow the server’s own install instructions.
4. **Get credentials.** You’ll usually need to prove you’re allowed to access the service.
  - `zotero.org` → Settings → Feeds/API → Create new API key (read/write access)
  - Note your Library ID (in URL: `zotero.org/users/XXXXX/`)

## Adding an MCP Server — Step by Step (cont.)

5. **Add to .mcp.json:**

```
"zotero": {  
  "command": "uv",  
  "args": ["run", "zotero-mcp"],  
  "env": {  
    "ZOTERO_LOCAL": "false",  
    "ZOTERO_API_KEY": "your-key",  
    "ZOTERO_LIBRARY_ID": "your-id"  
  }  
}
```

6. **Restart Claude Code** (close and reopen the panel, or restart VS Code).

7. **Test it.** Ask Claude: “Search my Zotero library for papers about [topic].” In the terminal CLI, type `/mcp` to see connected servers.

8. **Discover capabilities.** Ask Claude: “What Zotero tools do you have access to?”



# Agenda

1. Demo
2. Claude Code & MCP
3. Workspace, Design & Architecture
4. The Four Tools
5. Setting up CLAUDE.md, Specs, and MCP Servers
- 6. Summary, Next Steps, Getting Started**

# Getting Started

I think you should start small and build up your workspace yourself:

1. Install Claude Code → create a `research` workspace with a `CLAUDE.md` (you are here!)
2. Add one MCP server for the tool you use most (one of mine, or e.g. Notion, Email, ...)
3. Write a spec for your conventions in using that tool
4. Iterate: add servers, refine instructions, build habits, dump what doesn't work

**Template repo:** Alternatively, just use my template and follow the instructions to set my exact workspace up: [github.com/YOUR-USERNAME/research-workspace-template](https://github.com/YOUR-USERNAME/research-workspace-template)

## Takeaways

- My setup isn't so important. What matters is (a) giving an agent like Claude Code lots of context and (b) managing how it navigates it.
- You do those via (a) rich folders and MCP, and (b) specs and CLAUDE.md.
- To set up an MCP server, you need (a) the “program to interface with” either locally (obsidian) or online (google docs), (b) the “wrapper” /thing to execute/MCP server itself (usually a git repo you download into another folder), and (c) the MCP server's entry into your repo's .mcp.json
- Sounds complicated but usually you do this by (a) installing that program and (b) finding the MCP server's git repo and following the instructions there!

- Google Docs MCP: <https://github.com/a-bonus/google-docs-mcp>
- Obsidian MCP: <https://github.com/bitbonsai/mcp-obsidian>
- Zotero MCP: <https://github.com/54yyyu/zotero-mcp>
- Claude Code docs: <https://docs.anthropic.com/en/docs/claude-code>
- MCP specification: <https://modelcontextprotocol.io>
- uv (Python package manager): <https://docs.astral.sh/uv/>