Elite Context Engineering with Claude Code

https://www.youtube.com/watch?v=Kf5-HWJPTIE&t=1s

Levels of context engineering:

Beginner → Intermediate → Advanced → Agentic

Context Engineering: Let you manage the context window your AI Agents are running on.

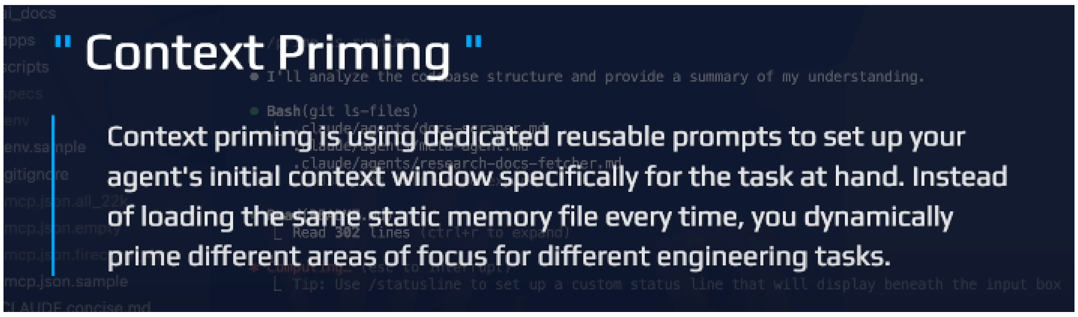
R&D: Reduce & Delegate

- Do not load MCP Servers unless you need them! (/context)
- Delete default .mcp.json for your codebase (saves you tokens by not preloading stock MCP Servers)

Context Prime > Claude.md

CLAUDE.consice.md

/prime



" Context Priming "

Context priming is using dedicated reusable prompts to set up your agent's initial context window specifically for the task at hand. Instead of loading the same static memory file every time, you dynamically prime different areas of focus for different engineering tasks.

Project Setup:

→ Claude → Commands → Prime

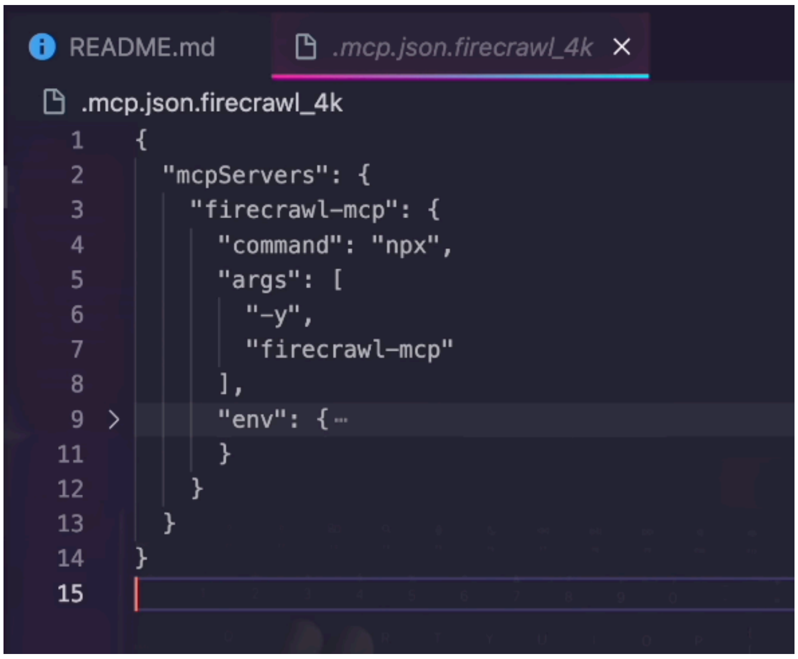
Be very porpouseful with your MCP Servers

.mcp.json

→ Always loading into context window, chewing expensive tokens

Solution: Delete this file, don't use this file for your default code base.

Have a seperate MCP Servers file:



.mcp.json.firecrawl_4k

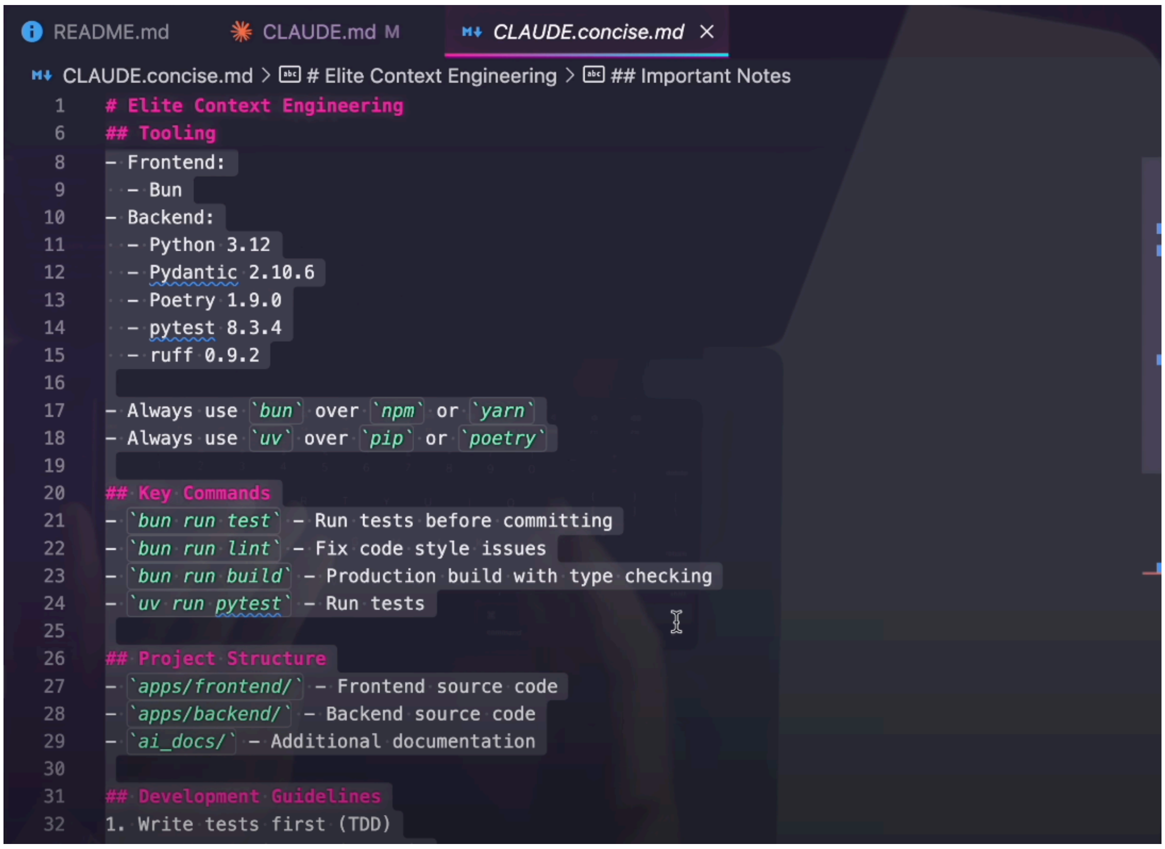
```
{
  "mcpServers": {
    "firecrawl-mcp": {
      "command": "npx",
      "args": [
        "-y",
        "firecrawl-mcp"
      ],
      "env": {…
      }
    }
  }
}
```

Inject it via terminal:

$ claude —mcp-config .mcp.json.firecrawl_4k —strcik-mcp-config (fires globals)

Context Prime > Claude.md (Not for beginners)

Things that we want every single agent to have!


README.md
CLAUDE.md M
CLAUDE.concise.md
CLAUDE.concise.md > # Elite Context Engineering > ## Important Notes

```markdown
# Elite Context Engineering
## Tooling
- Frontend:
  - Bun
- Backend:
  - Python 3.12
  - Pydantic 2.10.6
  - Poetry 1.9.0
  - pytest 8.3.4
  - ruff 0.9.2

- Always use `bun` over `npm` or `yarn`
- Always use `uv` over `pip` or `poetry`

## Key Commands
- `bun run test` - Run tests before committing
- `bun run lint` - Fix code style issues
- `bun run build` - Production build with type checking
- `uv run pytest` - Run tests

## Project Structure
- `apps/frontend/` - Frontend source code
- `apps/backend/` - Backend source code
- `ai_docs/` - Additional documentation

## Development Guidelines
1. Write tests first (TDD)
```

(Not full example)

Replace it on the CLAUDE.md file and reset the agents.

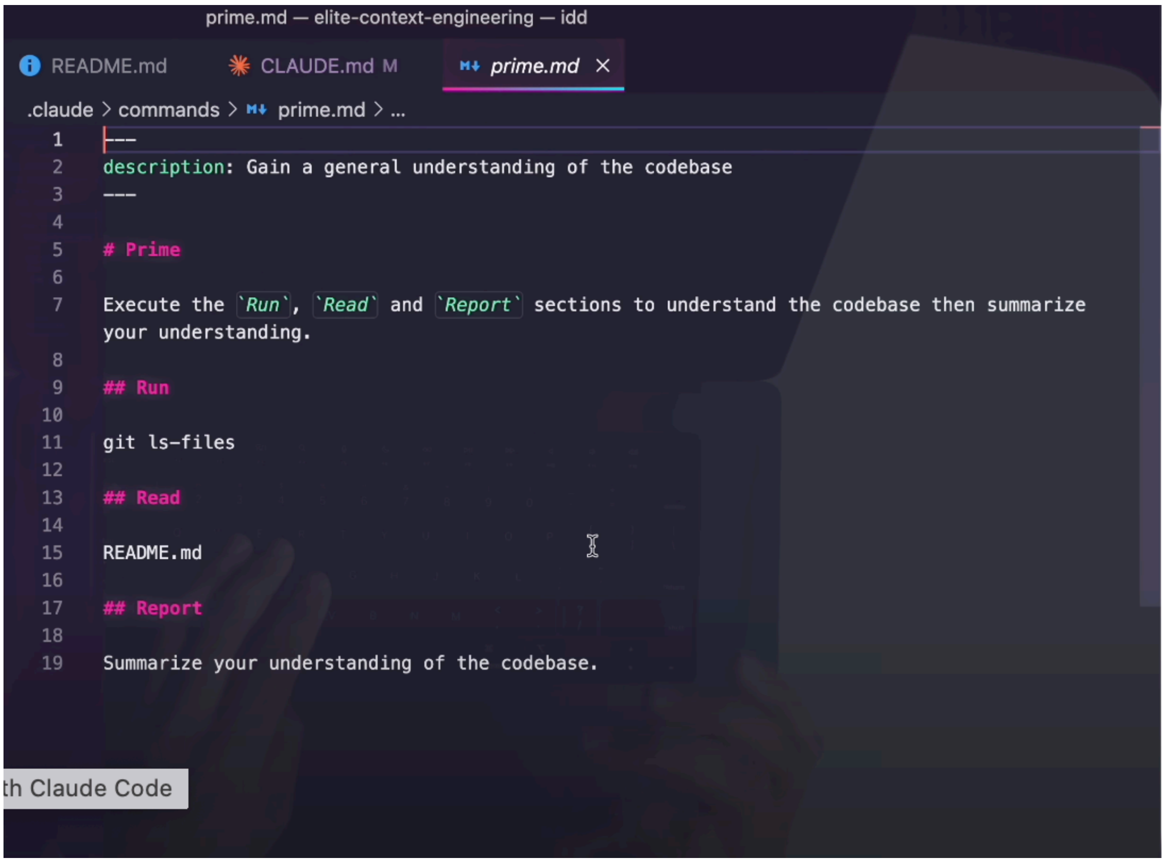
Check context usage, and see its lighter:

/context

Context Priming

/prime

Using dedicated reusable prompts to set up your agents initial context window specifically for the task at hand. Instead of loading the same static memory file every time, you dynamically prime different areas of focus for different engineering tasks.


prime.md — elite-context-engineering — idd
README.md
CLAUDE.md M
prime.md
.claude > commands > prime.md > ...
th Claude Code

```
---
description: Gain a general understanding of the codebase
---

# Prime

Execute the `Run`, `Read` and `Report` sections to understand the codebase then summarize
your understanding.

## Run

git ls-files

## Read

README.md

## Report

Summarize your understanding of the codebase.
```

/context to check the new context usage with this prime context

Imagine:

/prime_bug to be an bug expert

/prime_feature to develope new features

etc

This way you build many areas of focus for every agent!

Using Sub-Agents

Partially forked instanced window

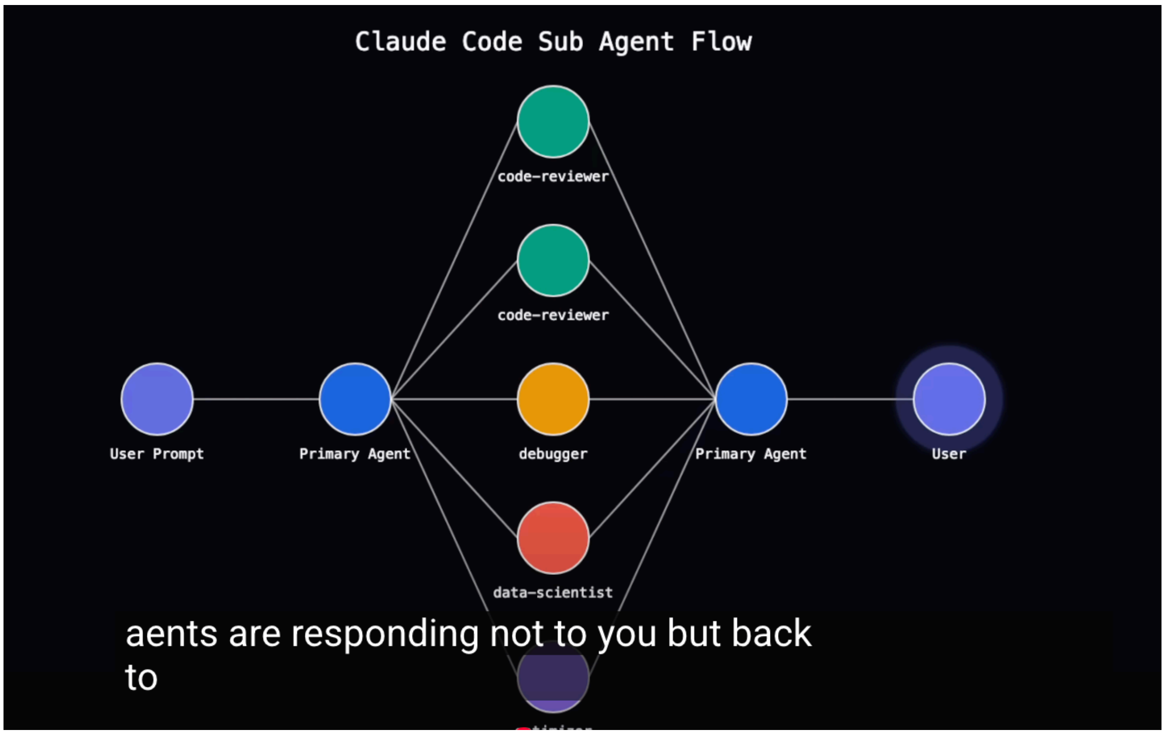
Working with system prompts instead of user prompts!

We keep context out of the primary agent context window.

Sub-agent delegation, do work that the primary agent can avoid, for example to read external documentation, we can delegate that work to a specific sub agents.

You still need to track this sub-agents.

Focused agents are performative agents!



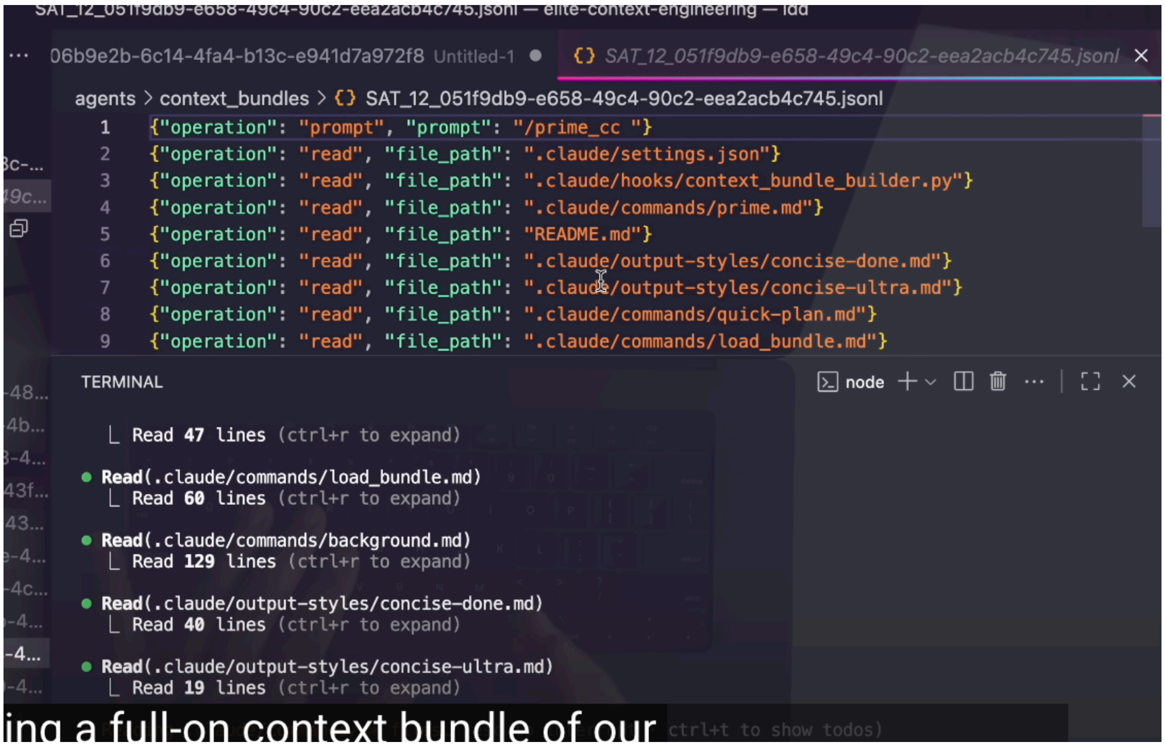
aents are responding not to you but back to

Use Context Bundles

Give us a solid understanding of 60 to 70% of what our previous agents have done.

Tells a fuller story to subsequent agents to execute.

We are getting logs!



```
agents > context_bundles > {} SAT_12_051f9db9-e658-49c4-90c2-eea2acb4c745.jsonl
{"operation": "prompt", "prompt": "/prime_cc "}
{"operation": "read", "file_path": ".claude/settings.json"}
{"operation": "read", "file_path": ".claude/hooks/context_bundle_builder.py"}
{"operation": "read", "file_path": ".claude/commands/prime.md"}
{"operation": "read", "file_path": "README.md"}
{"operation": "read", "file_path": ".claude/output-styles/concise-done.md"}
{"operation": "read", "file_path": ".claude/output-styles/concise-ultra.md"}
{"operation": "read", "file_path": ".claude/commands/quick-plan.md"}
{"operation": "read", "file_path": ".claude/commands/load_bundle.md"}
```

```
TERMINAL
  ⎿ Read 47 lines (ctrl+r to expand)

● Read(.claude/commands/load_bundle.md)
  ⎿ Read 60 lines (ctrl+r to expand)

● Read(.claude/commands/background.md)
  ⎿ Read 129 lines (ctrl+r to expand)

● Read(.claude/output-styles/concise-done.md)
  ⎿ Read 40 lines (ctrl+r to expand)

● Read(.claude/output-styles/concise-ultra.md)
  ⎿ Read 19 lines (ctrl+r to expand)
```

ing a full-on context bundle of our

/load bundle to an agent when the context expload

It lets a certain agent get to know the work done

Multi-Agent Delegation

One agent per porpouse, if something goes wrong you fix that piece.

Delegate ASAP

- background.md
    - $cldyo → $/background
    - We can prompt into $/background [OPTION] ...
    - Report to files while working in background