

A Personalised AI Research System

Skills, Agents, Hooks, Rules, and a Context Library
That Eliminates Repetitive Prompting

Open-Source Infrastructure for PhD Researchers

github.com/flonat/claude-research

Every new AI session starts from zero

The prompting tax

Multiple active research projects. Several supervisors. Separate teaching duties.

Every conversation begins with the same 500-word preamble: who you are, what you work on, where you left off.

Time spent re-explaining context is time not spent on research.

Before this system

"I'm a PhD student with several supervisors and multiple active projects, and I'm working on. . ."

After this system

"Plan my day."

A persistent context library turns lazy prompts into precise actions

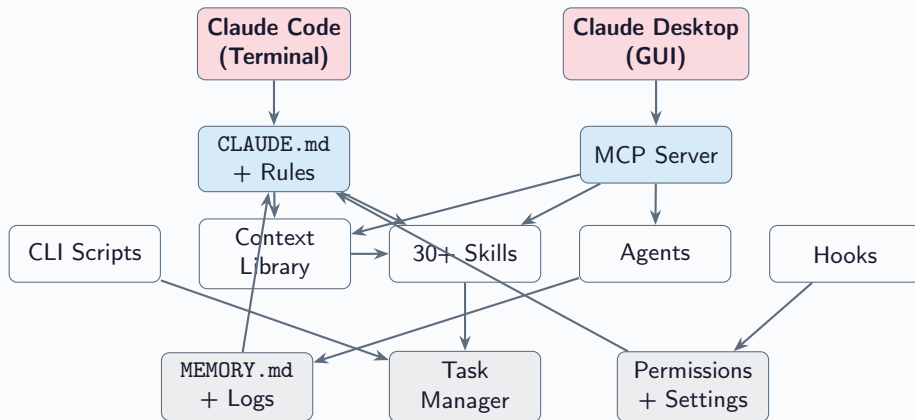
Three ideas, one system

1. **Context files** so the AI knows who you are before you speak
2. **Reusable skills** so recurring tasks need zero instructions
3. **Guardrails** so the system enforces safety without manual effort

Design principles

- **Lazy prompting** — say "Plan my day", not a paragraph
- **Hybrid storage** — Markdown (local) + task manager (dynamic)
- **Question-driven** — AI asks before acting
- **Read-only audits** — review, never auto-edit source
- **Session continuity** — every session resumes seamlessly

The system connects two interfaces to a shared foundation



Everything lives in one cloud-synced directory

```
Task Management/  
  CLAUDE.md                # Auto-read project instructions  
  GLOBAL-CLAUDE.md         # Symlinked to ~/.claude/CLAUDE.md  
  MEMORY.md                # Accumulated [LEARN] tags  
  .claude/  
    agents/                # Agent definitions  
    rules/                 # Auto-loaded rules  
    settings.local.json    # Auto-generated permissions  
  .context/                # AI context library  
  profile.md, current-focus.md, projects/, workflows/  
  .scripts/                # CLI tools (task, tasks, done, focus, papers, query)  
  .mcp-server-desktop/     # MCP server for Claude Desktop  
  .mcp-server-bibliography/ # Multi-source scholarly search MCP  
  skills/                  # Reusable skill definitions  
  templates/               # LaTeX templates (working paper, referee comments)  
  hooks/                   # Hook scripts  
  log/                     # Session logs + saved plans  
  resources/               # Cloned external repos
```

The context library eliminates repetitive explanations

What it stores

- `profile.md` — who you are, roles, preferences
- `current-focus.md` — what you are working on *now*
- `projects/_index.md` — all active projects
- `people/supervisors.md` — collaborators
- `preferences/` — priority rules, naming
- `workflows/` — daily review, meeting actions
- `resources.md` — useful links and references

Why it matters

- Claude reads these *automatically*
- No need to re-explain your situation
- Updated at end of every session
- Version-controlled with git
- Synced via cloud storage across machines

Key file

`current-focus.md` acts as working memory between sessions

A task manager provides the dynamic layer that Markdown cannot

Tasks Tracker

- GTD-style: Inbox → Not started → In progress → Waiting → Done
- Properties: name, status, priority, due date, project, source, task type

Research Pipeline

- Stages: Idea → Exploring → Drafting → Pre-submission → Submitted → R&R → Accepted
- Tracks papers, co-authors, target journals

How Claude accesses your tasks

Via MCP tools or CLI scripts. Any task manager with an API works (Notion, Todoist, Linear, etc.). The repo includes example CLI scripts you can adapt.

CLAUDE.md turns a blank session into an informed collaborator

What it does

Auto-loaded whenever Claude Code opens the project folder. Provides *every instruction* Claude needs.

Key sections:

- **Quick commands** — natural-language triggers ("Plan my day", "What's overdue?")
- **Context file pointers** — which files to read first
- **Conventions** — LaTeX to out/, Python via uv, no push without remote
- **Task manager integration** — direct access to task and paper databases
- **Skills, agents, hooks catalogue** — what is available and when to use it
- **File structure reference** — where everything lives

Two levels

Project CLAUDE.md (in repo) + **Global** ~/.claude/CLAUDE.md (symlinked, applies everywhere)

Auto-loaded rules enforce planning, safety, and documentation hygiene

Planning & Process

plan-first

Draft plan → approval → implementation
multi-file edits

read-docs-first

Check project docs before searching

scope-discipline

Only make changes explicitly requested

design-before-results

Lock research design before examining
estimates

Safety & Protection

break-the-glass

Warn before modifying infrastructure
(skills, hooks, rules)

data-sensitivity

data/raw/ is read-only; blind scripting
for restricted data

overleaf-separation

No code/data in paper/ — LaTeX
source only

Learning & Documentation

learn-~~first~~tags

Record corrections as [LEARN:*] in
MEMORY.md

learn-~~first~~claude-md

Instructions only; extract references to
docs/

Hygiene

ignore-agents-md

Skip files from external agents (OpenAI)

ignore-gemini-md

Skip files from external agents (Google)

All rules live in .claude/rules/ and are
auto-injected into every session as system
instructions. No manual loading needed.

30+ skills cover the full research lifecycle

Research & Writing

- literature — find papers, create .bib (incl. OpenAlex)
- split-pdf — deep-read via 4-page chunks
- devils-advocate — challenge assumptions
- interview-me — formalise ideas via interview
- research-ideation — generate research questions
- latex — compile with latexmk
- latex-autofix — **default compiler**, auto-fix errors
- proofread — 7-category academic check
- bib-validate — \cite{} cross-ref
- retarget-journal — switch paper to new journal
- paper-writing — multi-agent paper drafting
- humanizer — remove AI writing patterns
- project-deck — meeting status decks
- beamer-deck — rhetoric-driven presentations
- quarto-deck — Reveal.js HTML presentations

Code & Safety

- code-archaeology — review old code
- project-safety — safety rules, dry runs
- code-review — 11-category scorecard
- audit-project-research — audit vs template

Session & Context Management

- session-log — timestamped logs
- session-close — end-of-session checklist
- save-context — persist info to context library
- update-focus — session rotation, open loops
- update-project-doc — refresh stale docs
- init-project — bootstrap new project
- session-health — context usage monitor
- skill-extract — record and query learnings

Publishing

- replication-package — assembly, anonymization, audit
- pre-submission-report — pre-submission checklist

Utilities

- sync-resources — pull cloned repos
- sync-permissions — sync global permissions
- insights-deck — archive insights, build deck
- system-audit — parallel system audits

Skills are invoked in one word and work from any project

Anatomy of a skill

Each skill is a SKILL.md file in `skills/<name>/` with structured instructions.

Three ways to invoke:

1. **Slash command** in Claude Code: `/proofread`, `/latex-autofix`
2. **MCP tool** in Claude Desktop: `skill-proofread`, `skill-latex-autofix`
3. **Natural language**: “Proofread my paper” triggers the right skill

Cross-project access:

- SessionStart hook symlinks `~/.claude/skills/` → `Task Management/skills/`
- All skills available from *any* project folder
- Edits to SKILL.md files take effect immediately

Splitting PDFs into chunks produces section-level comprehension

The problem

Feeding a full 40-page PDF to an LLM produces shallow summaries and risks context-window overflow.

How the split-pdf skill works:

1. Download or locate the PDF
2. Split into 4-page chunks
3. Read chunks in small batches (not all at once)
4. Produce structured reading notes per section
5. Synthesise into a full summary with page references

Result

Deep, section-by-section comprehension instead of a vague one-paragraph summary.

Six agents run as autonomous sub-processes with persistent memory

What agents are

- Markdown files in `.claude/agents/`
- Launched via Task tool as a **separate sub-process**
- Own context, tool set, and model preference
- Memory persists in `.claude/agent-memory/`
- Solve the “grading your own homework” problem
- Fresh context = adversarial independence by design

Also available in Claude Desktop via MCP as `agent-*` tools.

6 specialised agents

1. **Referee 2** — review *your own* work
 - 5-audit protocol: code, replication, directory, output, econometrics
 - **Never modifies author code**
2. **Peer Reviewer** — review *external papers*
 - 3 parallel sub-agents: citations, novelty, methodology
3. **Proposal Reviewer** — assess *ideas/proposals*
 - 2 sub-agents: novelty and feasibility
4. **Paper Critic** — adversarial LaTeX auditor
 - Read-only; produces `CRITIC-REPORT.md`
5. **Domain Reviewer** — substantive correctness
 - Math, theory, code-theory alignment
6. **Fixer** — implement review fixes
 - Prioritises: Critical → Major → Minor

The daily review asks questions before making plans

Questions first:

1. How is your energy today?
2. Any hard constraints (meetings, deadlines)?
3. Where did we leave off?
4. Any pressure points?

Data sourced from:

- Task manager — overdue, due today, upcoming
- `current-focus.md` — session continuity
- Meeting transcripts — unprocessed actions

Output format:

- **Must Do** (1–2 items)
- **Should Do** (1–2 items)
- **Could Do** (stretch goals)
- **Parking Lot** (not today)

Trigger

Just say: "Plan my day"

Hooks enforce safety without manual effort

Hook	Event	What it does
Init permissions	SessionStart	Copies global permissions to local settings; creates sym-links
Startup context	SessionStart	Surfaces project docs on fresh sessions
Resume context	SessionStart	Surfaces current-focus.md and latest log on resume
Post-compact restore	SessionStart	Restores state after context compression
Protect sources	PreToolUse	Blocks edits outside CWD, ~/.claude/, Task Mgmt
Block destructive git	PreToolUse	Catches <code>reset --hard</code> , <code>push --force</code> , <code>rm -rf</code>
Enforce uv	PreToolUse	Blocks bare <code>python/pip</code> ; forces <code>uv run python</code>
Context monitor	PostToolUse	Tracks tool calls and warns before context compaction
Promise checker	Stop	Blocks if Claude promises to remember without actually writing
PreCompact save	PreCompact	Saves snapshot to log/ before context compression

All configured in `~/.claude/settings.json` under the "hooks" key.

The MCP server gives Claude Desktop full context awareness

How it works

1. Scans `skills/` and `.claude/agents/` directories
2. Each skill → `skill-*` tool; each agent → `agent-*` tool
3. `load-context` returns the full context library
4. Edits to skill or agent files are live immediately

Tools exposed

- `load-context` — profile, focus, projects, memory
- `skill-*` tools (one per skill)
- `agent-*` tools (referee2, peer, proposal, critic, domain, fixer)

CLI scripts (`.scripts/`)

- `task, tasks, done` — task CRUD
- `focus, papers` — focus and pipeline
- `inbox, week, conf` — reviews
- `query` — search files

Every session makes the next one better

Between sessions

- `current-focus.md` updated at session end
- Session logs saved to `log/`
- Plans saved to `log/plans/`
- Resume hook surfaces context automatically

Recovery protocol:

1. Read latest plan in `log/plans/`
2. Read latest session log
3. Read `current-focus.md`

Accumulated learning

- `MEMORY.md` stores [LEARN] tags
- Auto-loaded every session
- Captures notation, citation, code, method, domain corrections
- Prevents repeating the same mistakes

Compounding returns

Corrections made in January still inform sessions in December.

The system that explains itself never needs to be explained.

Skills · Agents · Hooks · Rules · Task Manager

Context library + persistent memory + session continuity

Available across Claude Code and Claude Desktop

github.com/flonat/claude-research