5 GitHub Repos That Fix 95% of Claude Code's Problems — the list

You commented REPOS. Here it is — the five repos, the real links, the exact `/command` for each, and the live star count so you know these aren't some random gists.

The thing nobody tells you

Claude Code out of the box is already good. But most of the stuff that drives people nuts about it — it can't watch a video, it burns your whole context window on research, it re-reads your codebase every single session, its frontends all look the same, it over-engineers a two-line fix into a framework — none of that is a Claude problem anymore. Somebody already fixed each one and put it on GitHub for free.

The catch is nobody installs them, because nobody tells you they exist. So you sit there fighting the same five things every day while the fix is a one-line install away.

That's the whole list below. Five repos. Each one kills one specific pain. Real links, real stars, real commands.

First — how to install a Claude Code skill (10 seconds)

Most of these are Claude Code *skills*. Installing a skill is stupidly simple: **you drop its folder into `~/.claude/skills/`.** That's the whole thing.

```
# clone the repo, then move its skill folder into your skills
dir
git clone <repo-url>
cp -r <repo-name>/<skill-folder> ~/.claude/skills/
```

Restart Claude Code (or start a fresh session) and the `/command` shows up. That path — `~/.claude/skills/` on Mac/Linux, or `%USERPROFILE%\.claude\skills\` on Windows — is where Claude Code looks for every skill you've got. Drop a folder in, it's live. Delete the folder, it's gone. No config file, no registry.

Now the five.

---

1. claude-video — Claude can finally watch video

The pain it fixes: Claude can't watch video. You paste a YouTube link or a screen recording and it just... can't see it. Guesses from the title.

The repo: github.com/bradautomates/claude-video — **7.7k stars**

How to use it: the `/watch` skill downloads the video, pulls out the frames, transcribes the audio, and then answers you from what's *actually on the screen* — not from the URL, not from vibes.

```
/watch https://youtube.com/watch?v=... what tool are they using
at 4:30?
```

Now "summarize this demo" or "what's the exact command they typed" actually works.

---

2. notebooklm-py — stop burning your context window on research

The pain it fixes: deep research eats your context window alive. You ask Claude to research something and it pulls in 40 pages, and now half your context is gone before you've even started the real work.

The repo: github.com/teng-lin/notebooklm-py — **17.6k stars**

How to use it: it's an unofficial NotebookLM API/skill. Claude hands the research off to Google's NotebookLM — the grounded synthesis engine, with sources, podcasts, quizzes — and gets back a clean answer at near-zero tokens. The heavy lifting happens over there, not in your window.

```
/notebooklm research "the current state of AI agent memory" and
give me the sourced summary
```

Your context stays free for the actual build.

---

3. graphify — stop re-grepping your codebase every session

The pain it fixes: every new session, Claude re-reads and re-greps your whole codebase from scratch. Slow, and it burns tokens re-learning what it already knew yesterday.

The repo: github.com/Graphify-Labs/graphify — **82.8k stars**

How to use it: `/graphify` turns any folder — code, docs, SQL, PDFs, whatever — into a queryable knowledge graph. Instead of searching your files every time, Claude *walks the graph*. It already knows how things connect.

```
/graphify index ./my-project
# then just ask normally — it walks the graph instead of re-reading everything
```

Faster answers, way fewer tokens, and it stops forgetting your project between sessions.

---

4. impeccable — so your AI frontend doesn't look like every other AI frontend

The pain it fixes: every AI-built frontend looks the same. Same purple gradient, same rounded cards, same font. You can spot it from across the room.

The repo: github.com/pbakaus/impeccable — **45.8k stars**

How to use it: `/impeccable` gives your AI harness an actual design language — real taste, real rules — so Claude designs *with* a point of view instead of reaching for the same template every time.

```
/impeccable redesign this landing page with a real design system, not the default AI look
```

The output stops screaming "an AI made this."

---

5. ponytail — stop Claude from over-engineering everything

The pain it fixes: Claude over-engineers. You ask for a small thing and it hands you an abstraction, a config layer, and three new files. The best code is the code you never wrote — and Claude forgets that.

The repo: github.com/DietrichGebert/ponytail — **81.1k stars**

How to use it: `/ponytail` makes the agent think like the laziest senior dev on the team. It runs the decision in order: **skip it → reuse what's there → use the standard library → add a dependency → and only then write new code.** Writing code is the last resort, not the first move.

```
/ponytail add rate limiting to this endpoint
# it'll check for an existing lib before writing a custom
limiter
```

Smaller diffs, less to maintain, fewer bugs you have to babysit later.

The list, one glance

- **claude-video** — 7.7k★ — `/watch` — Claude can watch video now.
- **notebooklm-py** — 17.6k★ — `/notebooklm` — research at near-zero tokens.
- **graphify** — 82.8k★ — `/graphify` — a knowledge graph it walks instead of re-greps.
- **impeccable** — 45.8k★ — `/impeccable` — a real design language, not templates.
- **ponytail** — 81.1k★ — `/ponytail` — thinks like the laziest senior dev.

Install path for all of them: clone the repo, drop the skill folder in `~/.claude/skills/`, restart.

The bigger picture

Here's what nobody says out loud: your agent is only as good as what you bolt onto it. Stock Claude Code is a coder. Add these five and it can watch, research without bleeding tokens, remember your whole codebase, design with taste, and stop over-building. Same tool, completely different ceiling.

This is exactly how I build client systems at Axionox — real production systems, not no-code duct tape. The skill layer is where a generic AI turns into something that actually does *your* job. Pick the pain that bugs you most today and install that one first.

Your next step

Don't install all five tonight. Pick the one pain that's been costing you the most and install just that. If it's "Claude keeps re-reading my codebase," start with **graphify**. If it's "my frontends all look the same," start with **impeccable**. Clone it, drop the skill folder in `~/.claude/skills/`, restart, use the command once. Feel the difference, then add the next.

Copy the block, drop the folder, run the command. That's the whole ask.

Built by Saad Bhatti — Axionox. I build production AI systems for a living (axionox.com). Follow for more like this.