Recursive self-improvement for your AI assistant: the minimum-credit setup

Nov 7, 2026 · @Dan

For the Basking class, Steffi and Flora. Written from one day's work on Dan's own setup, 7 Nov 2026. Works with Claude, ChatGPT or Gemini, on Windows or Mac; nothing here needs you to be a programmer. Each step says where the thing lives in each app.

Why this works

Most of your credits are not spent on thinking. They go on four leaks, and every one of them can be closed in an afternoon.

- Every plugin, connector and tool you have switched on is loaded into every single message, whether you use it or not. Twenty idle tools cost you on every turn of every chat.
- Screenshots. Any time your assistant looks at your screen or drives a browser by pictures, it pays hundreds of times more than for reading text.
- Re-explaining. Every new chat that starts blind makes you type your situation again, and makes the assistant search its memory and old chats to catch up. That search costs more than your message.
- Scheduled jobs that read everything every morning on the most expensive model, to produce a page you never open.

The one idea behind the whole setup: your assistant reads one tiny file first and writes one tiny file last. Everything else is pruning. Dan's own account went from four heavy morning runs to one, and from "tell me again where we were" to a 300-token file the assistant reads before it speaks. The same moves work in Claude, ChatGPT and Gemini, because they are habits and files, not features.

What you need and what it costs

One paid assistant plan, the free desktop app for it, one folder on your computer, and about 40 minutes. Everything else in this guide is free.

Item	Cost	Notes
Assistant plan	the standard paid tier of one app (Claude Pro, ChatGPT Plus or Google AI Pro)	The free tiers work for the file habits but cap heavy use. Only move to a top tier if you hit the limit most days. Never two paid plans at once.
Desktop app of that assistant	free	Needed for reading a folder on your computer. Windows and Mac.
A text editor	free	Notepad on Windows, TextEdit on Mac, or Obsidian (free, better, optional).
Google account	free	For the free automation (Apps Script) in step 6 and the cheap helper in step 5.
Python	free	Only for the optional pulse script in step 4. Mac has it; Windows installs it from python.org in two clicks.
Spend limit	0	Set on day one: a monthly cap, auto-reload off. The cap is what makes experimenting safe.

Assume Windows unless you own a Mac; every step below has both. Phones are for reading, not for this setup.

Step 1, prune

Ten minutes, and the biggest single saving on the list. Everything switched on is loaded into every message.

- Open the app's settings and look at plugins, connectors, integrations, GPTs or extensions you have enabled. Switch off everything you have not used in the last two weeks. You can switch it back on in one click when you need it.
- Keep one automation connector, not two. If you have both IFTTT and Zapier, pick the one with something real in it.

- Memory: ask your assistant to show you what it remembers ("export everything you remember about me, by category"). Delete what is wrong or stale. Tell it in plain words what to keep: your name, your work, the rules you want followed. Good memory means less re-explaining; stale memory means confident wrong answers.
- Spend: find the billing page, set a monthly limit, turn auto-reload off. If your top-ups last month ran far above the flat plan price, the next tier up is cheaper than credits.

Where it lives. Claude: Settings, then Connectors and Plugins; memory under Settings, Memory. ChatGPT: Settings, then Apps and connectors; memory under Personalization, Manage memories. Gemini: Settings, then Apps; memory under Saved info. Spend limits sit on each app's billing page, or on the API console if you use one.

Step 2, the home folder and the three files

Make one folder your assistant can always see, and keep three small text files in it. This is the whole continuity system; it fits in 40 lines.

Make a folder in your home directory called `ops`. In the assistant's desktop app, connect it (Claude: the + button, Add folder; ChatGPT: Settings, Work with apps or the folder picker in the desktop app; Gemini: a Google Drive folder does the same job, since Gemini reads Drive). Then create these three files inside it with any text editor.

File	What it holds	Who writes it
`NOW.md`	Where you are right now: what you are working on, the last thing finished, the next step, the one thing waiting on you. 20 lines maximum.	You, in a minute, at the start and end of a work block. Or the pulse script in step 4, automatically.
`handoff.md`	Two lines per session: date, what got done, what is next. The newest entry on top.	The assistant, at the end of every working chat, when you say "write the handoff".
`hands.md`	A tick list of things only you can do: a password to type, a file to share, a button to click.	The assistant appends; you tick.

Templates for all three are in the appendix. The rule that makes them work: nothing in these files is secret, nothing is long, and the assistant reads them before it does anything else.

If you use Obsidian, put the folder inside your vault. Obsidian's Local REST API plugin, plus one line in the Claude desktop config, lets the assistant read and write the vault without attaching a folder each time; that is an evening's project for later, not day one.

Step 3, the Ops Project and the opening instruction

Create one Project (Claude: Project; ChatGPT: Project or a custom GPT; Gemini: a Gem, or on personal accounts after 17 November 2026, a Saved info entry plus a pinned Drive folder) and paste this as its instructions:

```
On the first turn of every chat, before anything else, read ops/NOW.md,
ops/handoff.md and ops/hands.md. Open by saying in one line where I am and
what I was doing, then take my message. No screenshots and no searching old
chats until those three files are read.

At the end of a working session, when I say "write the handoff", add two
lines to the top of ops/handoff.md: the date, what was done, what is next.
Anything that needs my own hands goes as a tick-box line into ops/hands.md.

Style: concise, no moralising, no numbered lists, no em dashes. Ask before
anything that cannot be undone: sending, deleting, sharing, publishing,
paying. Never type a password; I type those.
```

Attach the `ops` folder to the Project itself, not to a single chat, so every new chat can see it. Do almost all your work inside this one Project. The cross-Project blindness most apps have stops mattering once your work lives in one place.

The effect: a new chat costs you nothing to start. The assistant opens with "You were halfway through the newsletter draft, the Takeout is still running, and the printer needs paper", and you carry on.

Step 4, the pulse (optional, the part that feels like magic)

A tiny script rewrites `NOW.md` every five minutes with what you are actually doing, so you never have to update it by hand. It records three things: the app in front and its window title, the files you touched in the last 15 minutes, and the top of `hands.md` and `handoff.md`. Nothing leaves your computer and it never takes a screenshot.

Privacy is built in, not bolted on. The script carries a short list of apps whose window titles are never logged (password managers, Signal, banking) and skips any private-browsing window. If an app is not on the allowed list, only its name is recorded. Edit those lists before you install.

- Windows: save `pulse.ps1` from the appendix into `ops`, then open Task Scheduler, Create Basic Task, trigger "When I log on", action "Start a program": `powershell.exe -ExecutionPolicy Bypass -File "C:\Users\you\ops\pulse.ps1"`, and in the task's properties set it to repeat every 5 minutes. One dialog, done.
- Mac: save `pulse.py` and `install-pulse.command` from the appendix into `ops`, double-click the `.command` file once, click OK on the two permission prompts.

Check it works: open `NOW.md` after five minutes. If the top line carries the time, the loop is closed, and from now on every chat opens knowing where you are.

Gemini users: point the script's output at a Drive folder instead of a local one, and Gemini reads it from there.

Step 5, cheap helpers

Your main assistant is the expensive one. Give the bulk reading to a cheaper one and hand back only the result.

- Long files. Never paste a 200-page PDF into your main chat. Ask a cheap model to summarise it first (Gemini's free tier in AI Studio, or the free tier of any of the three apps), then give your main assistant the two-page summary. Ten times cheaper, often better, because the main assistant now reasons over a clean digest.
- Second opinions. For a decision, one strong answer beats four. Use several models only when you want them to disagree, and then ask for a synthesis.
- Paste text, never upload. A file uploaded to a multi-model service can sit on their storage with no delete button; Dan learned this the expensive way. Paste the text you want read, scrubbed of names, addresses, health and client details, and nothing else. Free tiers may use what you paste to train, so the scrub is not optional.
- The one script worth having, for the technically curious: a 40-line Python file with your Gemini key in a file beside it, that summarises any document from the command line so your main assistant never has to load it. Dan's version is `council.py`; ask him for it. A spend limit on the key, a few euros a month, makes it safe to forget about.

Where it lives. Gemini keys: AI Studio, API keys; the paid tier is pennies and Google states paid-tier data is not used for training. Claude and ChatGPT keys: their developer consoles, same idea, same limits. Not needed on day one.

Step 6, scheduled tasks that earn their keep

A scheduled task is a chat that runs without you. It costs the same as a chat, every day, forever, so each one has to pay for itself.

- Keep the ones whose output reaches you as a push notification you actually read. A morning line with today's calendar and the two emails that matter is worth it. A beautiful page you never open is not.
- Put each task on the cheapest model that can do it. Reading a list and filtering it is a small-model job. Reading receipts and getting the amounts right needs a mid-size model. Almost nothing needs the top model on a timer.
- Rule-based jobs do not need a model at all. Labelling and archiving mail from known senders is a Gmail filter. Watching a shared folder for someone else's edits is a six-line Google Apps Script on a timer, free, and it sees who edited, which the assistant cannot. Calendar sanity checks are a regex. Ask your assistant to write the script; you paste it into script.google.com and click Allow once.
- Once a month, list the tasks and ask of each: did I read it last week? If not, cut it or halve it.

Where it lives. Claude: scheduled tasks, visible in the sidebar, each with its own model setting. ChatGPT: Tasks. Gemini: Scheduled actions. Google Apps Script works for all three, since it lives in your Google account, not in the assistant.

Dan's account before and after one audit: four heavy weekday runs on the top model became one; two folder-watching runs a day became a free script; the rest moved to small models. Same information reaching him, a fraction of the credits.

The safety rules that travel with all of this

These are not caution for its own sake. Each one exists because skipping it once cost somebody real time or privacy.

- Passwords and API keys are typed by you, into a file on your own computer, never into a chat. A key pasted into a chat lives in that transcript. If it happens, make a new key and delete the old one the same day.
- Nothing irreversible without your yes: no sending, deleting, sharing, publishing or paying. Put this line in your Project instructions and the assistant will ask every time.
- Paste text to outside models, never files. Scrub names, addresses, health, money and client details first. Keep a list of words that must never leave your machine and check the text against it before you paste.
- Folders named Private stay private. Tell the assistant which folders it may never read, and keep that list in the Project instructions too.
- Window titles and recent files are a log of your life. Keep the pulse's deny list honest, and keep `NOW.md` out of anything you share.
- A refusal from the assistant's permission system is not an obstacle to route around. It is the system doing its job; the step goes on your `hands.md` list instead.

- Keep your own words your own. When you forward something the assistant wrote, say so. When it drafts for you, keep your punctuation and your voice; polish is the tell.

The weekly fifteen minutes

This is what makes it recursive: the assistant improves its own setup, with you as the hand that clicks.

Once a week, open the Ops Project and say: "Read hands.md and handoff.md. Tell me what I am paying for that I did not use this week, what you had to ask me twice, and what one change would save the most credits. Then update the Project instructions with what we agree." Tick what you can on `hands.md` while it thinks.

Three questions keep the loop honest:

- What did I re-explain this week? That is a line missing from the Project instructions or from memory.
- What did the assistant screenshot or search for that a file could have told it? That is a line missing from `NOW.md`.
- What ran on a schedule that I never read? That is a task to cut.

Write the answers into the instructions, not into your head. In a month the setup fits you like a glove and the credit bill is a fraction of what it was. Then teach it to one friend; explaining it is how you find out which parts you still do by habit.

No em dashes anywhere, no numbered lists, your own punctuation in anything you send as yourself. Over and out with love.

Appendix: templates and scripts to copy

The three files, as they should look on day one.

```
# NOW.md
Working on: the newsletter draft, section 3
Last finished: sent the invoice to the printer
Next: proofread section 3, then export the PDF
Waiting on me: reply to Ana about Thursday
```

```
# handoff.md (newest on top)
2026-11-07 Set up the ops folder, Project and pulse. Next: prune the plugins,
set the spend cap.
```

```
# hands.md: things only I can do
- [ ] set the monthly spend limit and turn auto-reload off
- [ ] connect the ops folder to the Project
```

Windows pulse, `pulse.ps1`. Save in `ops`, schedule with Task Scheduler every 5 minutes. Edit the two lists at the top first.

```
# pulse.ps1: rewrites ops\NOW.md with what you are doing. Nothing leaves this
computer.
$ops = "$HOME\ops"
$deny = @("1Password","Bitwarden","KeePass","Signal","Passwords")          #
never logged at all
$titleOk =
@("Claude","ChatGPT","Code","Obsidian","chrome","msedge","firefox","WINWORD",
"EXCEL","notepad","explorer","WhatsApp","OUTLOOK","Zoom")  # window title
logged only for these
Add-Type @"
using System; using System.Runtime.InteropServices;
public class W { [DllImport("user32.dll")] public static extern IntPtr
GetForegroundWindow();
[DllImport("user32.dll")] public static extern uint
GetWindowThreadProcessId(IntPtr h, out uint pid); }
"@
$h = [W]::GetForegroundWindow(); $procId = 0;
[W]::GetWindowThreadProcessId($h, [ref]$procId) | Out-Null
$p = Get-Process -Id $procId -ErrorAction SilentlyContinue
$app = if ($p) { $p.ProcessName } else { "unknown" }; $title = ""
if ($deny -contains $app) { $app = "[private app]" }
elseif ($titleOk -contains $app -and $p.MainWindowTitle -and
$p.MainWindowTitle -notmatch "InPrivate|Incognito|Private") { $title =
$p.MainWindowTitle.Substring(0, [Math]::Min(90, $p.MainWindowTitle.Length)) }
$since = (Get-Date).AddMinutes(-15)
$files = Get-ChildItem
"$HOME\Documents","$HOME\Desktop","$HOME\Downloads",$ops -Recurse -File -
ErrorAction SilentlyContinue |
  Where-Object { $_.LastWriteTime -gt $since -and $_.FullName -notmatch
"\\Private\\|Signal|\\\.git\\|node_modules|NOW\.md" } |
  Sort-Object LastWriteTime -Descending | Select-Object -First 8
$lines = @("# NOW (auto, $(Get-Date -Format 'ddd dd MMM HH:mm'), pulse.ps1,
read this first)", "Active: $app" + $(if ($title) { " `"$title`"" } else { ""
}))
$lines += "Files touched (last 15 min):"
if ($files) { $files | ForEach-Object { $lines += "  " +
$_.FullName.Replace($HOME, "~") } } else { $lines += "  none" }
foreach ($f in @("hands.md","handoff.md")) { if (Test-Path "$ops\$f") {
```

```
$lines += "From ${f}:"; Get-Content "$ops\$f" | Where-Object { $_.Trim() -and
-not $_.StartsWith("#") } | Select-Object -First 5 | ForEach-Object { $lines
+= "  $_" } } }
$lines | Select-Object -First 22 | Set-Content "$ops\NOW.md" -Encoding UTF8
```

Mac pulse, `pulse.py` plus `install-pulse.command`. Save both in `ops`, double-click the `.command` once.

```
#!/usr/bin/env python3
# pulse.py: rewrites ops/NOW.md with what you are doing. Nothing leaves this
computer.
import os, subprocess, datetime, re
HOME = os.path.expanduser("~"); OPS = os.path.join(HOME, "ops")
DENY = {"Signal", "Passwords", "1Password", "Keychain Access", "Bitwarden",
"Messages", "Photos"}
TITLE_OK = {"Claude", "ChatGPT", "Obsidian", "Google Chrome", "Safari",
"Finder", "Pages", "Numbers", "TextEdit", "Notes", "Mail", "WhatsApp",
"Zoom", "Code"}
EXCLUDE = ("/Private", "Signal", "/.git", "node_modules", "/Library/",
"NOW.md")
def osa(s):
    try: return subprocess.run(["osascript", "-e", s], capture_output=True,
text=True, timeout=8).stdout.strip()
    except Exception: return ""
app = osa('tell application "System Events" to get name of first application
process whose frontmost is true')
title = ""
if app in DENY: app = "[private app]"
elif app in TITLE_OK:
    title = osa('tell application "System Events" to tell (first application
process whose frontmost is true) to get name of front window')[:90]
    if re.search(r"incognito|private", title, re.I): title = "[private
window]"
files = []
for d in ("Documents", "Desktop", "Downloads", "ops"):
    r = subprocess.run(["mdfind", "-onlyin", os.path.join(HOME, d),
"kMDItemFSContentChangeDate >= $time.now(-900)"], capture_output=True,
text=True)
    files += [p for p in r.stdout.splitlines() if os.path.isfile(p) and not
any(x in p for x in EXCLUDE)]
files = sorted(set(files), key=os.path.getmtime, reverse=True)[:8]
lines = ["# NOW (auto, %s, pulse.py, read this first)" %
datetime.datetime.now().strftime("%a %d %b %H:%M"), "Active: %s%s" % (app or
"unknown", ' "%s"' % title if title else "")]
lines.append("Files touched (last 15 min):"); lines += ["  " + p.replace(HOME
+ "/", "~/") for p in files] or ["  none"]
```

```
for f in ("hands.md", "handoff.md"):
    p = os.path.join(OPS, f)
    if os.path.exists(p):
        lines.append("From %s:" % f); lines += ["  " + l.rstrip() for l in
open(p, encoding="utf-8") if l.strip() and not l.startswith("#")][:5]
open(os.path.join(OPS, "NOW.md"), "w", encoding="utf-
8").write("\n".join(lines[:22]) + "\n")
```

```
#!/bin/bash
# install-pulse.command: runs pulse.py every 5 minutes via launchd. Double-
click once.
OPS="$HOME/ops"; PL="$HOME/Library/LaunchAgents/com.me.pulse.plist"; mkdir -p
"$HOME/Library/LaunchAgents"
cat > "$PL" <<PLIST
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0"><dict><key>Label</key><string>com.me.pulse</string>
<key>ProgramArguments</key><array><string>/usr/bin/python3</string>
<string>$OPS/pulse.py</string></array>
<key>StartInterval</key><integer>300</integer><key>RunAtLoad</key><true/>
</dict></plist>
PLIST
launchctl bootout "gui/$(id -u)/com.me.pulse" 2>/dev/null; launchctl
bootstrap "gui/$(id -u)" "$PL" && echo "installed"
python3 "$OPS/pulse.py" && cat "$OPS/NOW.md"
```

Gemini Gems, if you have them on a personal Google account: export them before 17 November 2026 (Google Takeout, deselect all, tick only "Gemini"); the instructions come out as text you can paste into a Project or a Saved info entry.