Slim omgaan met context-kosten + lokaal model

> Research-oriëntatie (13 juni 2026). Controleer exacte modelversies/-tags tegen de actuele Ollama-bibliotheek vóór installatie — modelnummers veranderen snel.

Samenvatting voor Jamal

Dit zijn de 5 belangrijkste conclusies:

- **Je CLAUDE.md + MEMORY.md zijn je grootste kostenpost.** Elke beurt lees je ze opnieuw. In tests gaf afslanken (van ~3.800 naar ~300 tokens) geen kwaliteitsverlies.
- **Cache-reads zijn al goedkoop (10% van normaal) — maar je betaalt voor wat erin zit.** Een grote CLAUDE.md betekent op élke beurt betalen voor die tokens. Kleiner = goedkoper per beurt.
- **Agent teams zijn duur.** Elke teammate draait zijn eigen context. Jouw opstelling (Conductor + Sjakie) is al een team. Houd ze klein en scherp.
- **Lokaal model is klaar voor routine-werk.** Qwen draait soepel op je Mac (afhankelijk van RAM) via Ollama/MLX. Geschikt voor bot-classificatie, samenvatten, simpele code.
- **Wat ik zou doen (volgorde):**
  1. CLAUDE.md onder ~150 regels brengen. Eerst.
  2. Zware workflows naar Skills (laden pas bij aanroep).
  3. Ongebruikte MCP-servers uitzetten.
  4. Coco + SEO + Mind laten draaien op Ollama voor routine; alleen complex naar Claude.
  5. `/compact` proactief gebruiken (vóór 70%, niet erna).

Deel 1: Context/geheugen-kosten verlagen

1.1 CLAUDE.md afslanken — het grootste lek

Een sessie start met tienduizenden tokens vóórdat je iets typt: systeem-prompt + CLAUDE.md + MEMORY.md + MCP-schema's + skill-namen. Anthropic's advies: houd CLAUDE.md onder ~200 regels en zet gedetailleerde workflows in **Skills** (die laden alleen bij aanroep).

JA — doe dit eerst. Verplaats project-workflows naar Skills; houd in CLAUDE.md alleen kernregels + credential-locaties + werkdiscipline.
Bron: https://code.claude.com/docs/en/costs

1.2 Skills voor progressieve disclosure

Skills laden bij opstart alleen hun beschrijving (~30–100 tokens); de volle inhoud pas bij aanroep. **JA** — directe vervanging van zware CLAUDE.md-inhoud.
Bron: https://www.firecrawl.dev/blog/claude-code-token-efficiency

1.3 MCP-server overhead beperken

Tool-definities zijn "deferred" (alleen namen), maar 5+ servers voegen alsnog overhead toe. **JA** — zet servers die je niet elke sessie nodig hebt uit via `/mcp`; gebruik `/context` om te zien wat ruimte inneemt.
Bron: https://code.claude.com/docs/en/costs

1.4 Prompt caching slim gebruiken

Cache-read = 10% van normale prijs; cache-write (5 min) = 1,25×, (1 uur) = 2×. Jouw 1,5 miljard cache-reads zijn al goedkoop — het probleem is de grote context die gecached wordt. Voor de bots op de API: zet `cache_control` op het systeem-prompt → 90% besparing na de eerste beurt.
Bron: https://platform.claude.com/docs/en/build-with-claude/prompt-caching

1.5 /compact proactief gebruiken

Gebruik `/compact` bij ~60–70%, met gerichte instructie (`/compact Focus on code changes and open decisions`). `/clear` bij nieuw onderwerp.
Bron: https://www.kdnuggets.com/7-practical-ways-to-reduce-claude-code-token-usage

1.6 Subagents voor verbose operaties

Delegeer logs/scraping/grote bestanden aan subagents — hun output blijft in hun eigen context, alleen een samenvatting keert terug. **JA**, maar niet voor triviale taken (overhead).
Bron: https://code.claude.com/docs/en/costs

1.7 Agent teams bewust inzetten

Teams gebruiken fors meer tokens (elke teammate = eigen context). **MISSCHIEN** — houd spawned teams klein (3–5), Sonnet voor teammates, opruimen na gebruik.
Bron: https://code.claude.com/docs/en/agent-teams

1.8 Hooks voor data-preprocessing

Een PreToolUse-hook kan een groot logbestand filteren naar alleen ERROR-regels vóór Claude het ziet (80–99% reductie). **JA** — log-filter + firecrawl voor schone markdown.
Bron: https://code.claude.com/docs/en/costs

Deel 2: Lokaal model op de Mac

2.1 Welk model past op welke Mac (indicatief)

RAM	MODEL	GESCHIKT VOOR
8 GB	Qwen klein (4B) Q4	classificatie, korte chat
16 GB	Qwen ~9B Q4	samenvatten, classificatie, simpele code
24 GB	Qwen ~27B Q4	goede code, langere teksten
32–48 GB	Qwen MoE (A3B) Q4	code + redeneren, snel
48–64 GB+	Llama 70B Q4	maximale kwaliteit lokaal

Belangrijk: memory-bandbreedte telt zwaarder dan chip-generatie voor tokens/sec.
Bron: https://insiderllm.com/guides/best-local-llms-mac-2026/

2.2 Qwen — beste keuze voor routine

Klein/middel Qwen bij Q4 draait goed op 16 GB+; MoE-varianten (weinig actieve parameters) zijn snel met veel kennis. **JA** voor bot-routine.
Bronnen: https://www.promptquorum.com/local-llms/run-qwen-locally-guide-2026 · https://willitrunai.com/blog/qwen-3-5-mlx-apple-silicon-guide

2.3 DeepSeek R1 / V3

Sterk in redeneren/code, maar volle modellen vragen 40+ GB; distills (7B/14B) scoren lager dan Qwen voor jouw use-cases. **MISSCHIEN** — tweede keuze.
Bron: https://tokenmix.ai/blog/deepseek-for-mac-local-setup-2026

2.4 Ollama (met MLX) is de praktische keuze

Ollama draait nu op MLX op Apple Silicon (fors sneller dan het oude GGUF). Geeft een OpenAI-compatibele API op `localhost:11434` → werkt direct met je Python-bots. **JA**.
Bron: https://ollama.com/blog/mlx

2.5 Geschiktheid per taak

TAAK	LOKAAL?	BESTE KEUZE
Telegram-bericht classificeren (haar/niet-haar)	JA	Qwen ~9B
Korte samenvatting (blog, Plaud-note)	JA	Qwen ~9B
Embeddings	JA	nomic-embed-text
Simpele scripts/hooks	JA	Qwen 27B / MoE
Ingewikkelde redenering	NEE	Claude
Strategie/architectuur	NEE	Claude
Lange synthese (100K+)	NEE	Claude

Deel 3: Concreet stappenplan voor Haarvisie

Stap 1 — CLAUDE.md verkleinen (max ~150 regels; workflows → Skills). Verwachte besparing 60–80% baseline-context.

Stap 2 — Ollama + bots routen. `brew install ollama` → `ollama pull` een Qwen-model → simpele vragen lokaal, complex naar Claude. Verwachte besparing 60–80% bot-API-kosten.

Stap 3 — MCP-servers per project activeren (Shopify/Canva/Drive/Higgsfield niet standaard aan). 5.000–15.000 tokens minder per start.

Stap 4 — /compact proactief bij ~70%, met bewaar-instructie.

Stap 5 — Prompt caching op de bots (`cache_control` 1u TTL op systeem-prompt). 90% besparing op systeem-prompt per call.

Realistisch gecombineerd: 50–70% minder tokenverbruik bij gelijke kwaliteit voor complexe taken.

Bronnenlijst

1. https://code.claude.com/docs/en/costs
2. https://code.claude.com/docs/en/agent-teams
3. https://platform.claude.com/docs/en/build-with-claude/prompt-caching
4. https://www.kdnuggets.com/7-practical-ways-to-reduce-claude-code-token-usage
5. https://www.firecrawl.dev/blog/claude-code-token-efficiency
6. https://ollama.com/blog/mlx
7. https://insiderllm.com/guides/best-local-llms-mac-2026/
8. https://www.promptquorum.com/local-llms/run-qwen-locally-guide-2026
9. https://willitrunai.com/blog/qwen-3-5-mlx-apple-silicon-guide
0. https://www.sitepoint.com/hybrid-cloudlocal-llm-the-complete-architecture-guide-2026/
1. https://www.mindstudio.ai/blog/run-local-ai-models-with-claude-code-cut-costs
2. https://tokenmix.ai/blog/deepseek-for-mac-local-setup-2026
3. https://todatabeyond.substack.com/p/claude-code-memorymd-everything-you
4. https://www.finout.io/blog/anthropic-api-pricing