CLAUDE CODE TOKEN OPTIMIZATION GUIDE

TOKEN OPTIMIZATION RESEARCH (5 AGENTEN, 60+ QUELLEN) APRIL 2026 EXTRAHIERT: 2026-04-06

Bis zu 90% Kostenreduktion mit 6 sofort umsetzbaren Tricks. Basierend auf 60+ Quellen und 30+ GitHub Repos.

INHALTSVERZEICHNIS

01 Das Problem: Warum deine Claude-Kosten explodieren

- Jeder Subagent (Explore, Plan, General-Purpose) laedt den kompletten Context neu — System Prompt, CLAUDE.md, Tool-Definitionen
- CLAUDE.md wird bei JEDER einzelnen Message geladen — egal ob relevant oder nicht. Bei 13KB sind das ~3.500 verschwendete Tokens pro Nachricht
- Alle Subagents erben standardmaessig das teuerste Modell (Opus: $5/$25 pro 1M Tokens)
- Extended Thinking verbraucht bis zu 32.000 Output-Tokens pro Request — oft unnoetig

> *Das Ergebnis: Eine komplexe Research-Session verbrennt 5-6 volle Opus-Sessions. Zwei 20x Max-Subscriptions reichen nicht mehr.*

02 Trick 1: Haiku-Subagents — 80% Ersparnis mit einer Zeile

- Eine einzige Zeile in `settings.json` aendert alles: `"CLAUDE_CODE_SUBAGENT_MODEL": "haiku"`
- Spart ~80% auf jeden Subagent-Call (von $5/$25 auf $1/$5 pro 1M Tokens)
- Der Orchestrator (Opus/Sonnet) trifft weiterhin ALLE Entscheidungen — Subagents fuehren nur aus
- Explore-Agents laufen bei Anthropic selbst schon auf Haiku — das bestaetigt die Strategie
- Fuer den Plan-Subagent optional `model: sonnet` im Frontmatter setzen (mehr Power fuer Planung)

TERMINAL

```
// ~/.claude/settings.json
{
  "env": {
    "CLAUDE_CODE_SUBAGENT_MODEL": "haiku"
  }
}
```

03 Trick 2: Thinking Tokens begrenzen — 70% weniger Denkkosten

- Default Thinking Budget: ~32.000 Tokens pro Request. Das sind ~$2.40 bei Opus
- Mit `MAX_THINKING_TOKENS=10000`: Nur ~$0.75 — bei gleichbleibender Qualitaet fuer Standard-Tasks
- Thinking Tokens kosten normale Output-Token-Preise — kein Sonder-Multiplier
- Fuer triviale Tasks: `/effort low` reduziert noch weiter auf ~$0.15

TERMINAL

```
// ~/.claude/settings.json
{
  "env": {
    "MAX_THINKING_TOKENS": "10000"
  }
}
```

04 Trick 3: Lokale Modelle — Der Game Changer

- Lokale Modelle auf Apple Silicon kosten $0 pro Token — unbegrenzt, ohne API-Abhaengigkeit
- Gemma 4 26B MoE: Nur 3.8B aktive Parameter von 25.2B total (Mixture-of-Experts Architektur)
- Passt in 18GB RAM bei Q4 Quantisierung — laeuft auf jedem 24GB MacBook
- Volle 256K Context in 23GB — Dense-Modelle schaffen nur ~64K bei gleichem RAM
- LMArena Score #6 global — ueber vielen Dense-Modellen mit mehr aktiven Parametern

TERMINAL

```
# Installation (2 Minuten)
brew install ollama
ollama serve
ollama pull gemma4:26b
```

> *Alternative: Qwen 3.5 27B (besser bei Agentic Tasks, aber nur ~64K Context bei 24GB)*

---

05 Trick 4: Model Routing — 60-88% Gesamtreduktion

- Claude Code Router (31.600+ GitHub Stars) routet Tasks intelligent zum guenstigsten Modell
- Background-Tasks (File Search, Exploration) → lokales Ollama-Modell = $0
- Architektur-Entscheidungen, Security-Reviews → Claude Cloud (Opus/Sonnet)
- Ergebnis: 60-88% Gesamtkostenreduktion bei gleichbleibender Qualitaet

Task-Typ	Empfohlenes Modell	Kosten
Architektur	Claude Opus (Cloud)	$$$
Complex Coding	Claude Sonnet (Cloud)	$$
Code Review	Lokales Modell	$0
File Exploration	Lokales Modell	$0
Tests schreiben	Lokales Modell	$0
Commit Messages	Lokales Modell	$0

---

06 Trick 5: Prompt Caching optimal nutzen — 90% Rabatt

- Claude Code aktiviert Prompt Caching automatisch — kein Setup noetig
- Cache Hit = 90% Rabatt auf Input-Tokens (statt $5 nur $0.50 pro 1M)
- Cache TTL: 5 Minuten (reset bei jedem Hit)
- Optimale Sessions erreichen ~95% Cache Hit Rate

Best Practices:

- CLAUDE.md stabil halten — jede Aenderung invalidiert den Cache
- Requests innerhalb von 5 Minuten batchen = maximale Cache Hits
- Nicht zwischen vielen Projekten hin- und herspringen

---

07 Trick 6: Context Management — 60-80% weniger Tokens

- `/compact` nach Research- und Explorations-Phasen — komprimiert den gesamten Conversation History
- `/compact Focus on code changes` vor Implementation — gezielte Komprimierung
- `/clear` zwischen komplett verschiedenen Tasks — 100% Context Reset
- `/effort low` fuer triviale Aufgaben — minimales Thinking Budget

> *Nicht auf Auto-Compact verlassen — es triggert erst bei ~83.5% Context-Fuellstand, oft mitten in einem Task.*

---

08 Die Zahlen: Vorher vs. Nachher

Metrik	Vorher	Nachher	Ersparnis
Kosten pro Request (Best Case)	$0.47	$0.05	~90%
Subagent-Kosten	$5-25/1M	$1-5/1M	80%
Thinking-Kosten	$0.80/Request	$0.25/Request	70%
Lokale Tasks	$0.47	$0	100%
Cache-Effizienz	~50%	~95%	90%

---

09 3-Phasen Implementierung

Phase 1 — Sofort (5 Minuten, Zero Infra)

- `settings.json` anpassen (Haiku + Thinking Limit)
- `/compact` und `/clear` als Gewohnheit
- `npx ccusage@latest` fuer Monitoring
- Erwartete Einsparung: 30-50%

Phase 2 — Diese Woche (1 Stunde)

- Ollama installieren + Modell pullen
- Claude Code Router konfigurieren
- CLAUDE.md auf unter 200 Zeilen kuerzen
- Erwartete Einsparung: 60-80%

Phase 3 — Naechste Woche (2 Stunden)

- Output-Filter Hooks einrichten
- Cost Dashboard Tools installieren
- Optional: LiteLLM Proxy auf zweitem Geraet
- Erwartete Einsparung: 85-97%

---

10 Das wichtigste Prinzip

> *Planung ist alles. Der Orchestrator trifft ALLE Entscheidungen. Subagents sind nur ausfuehrende Haende. Schlechte Planung = schlechtes Ergebnis, egal welches Subagent-Modell. Gute Planung + guenstige Subagents = gleiche Qualitaet, 80% billiger.*

Die Qualitaet deiner Ergebnisse haengt nicht davon ab, wie teuer dein Subagent-Modell ist — sondern wie gut dein Orchestrator die Aufgaben plant und delegiert.

---

11 Quellen

- Anthropic Pricing (Stand April 2026): Opus 4.6 $5/$25, Sonnet 4.6 $3/$15, Haiku 4.5 $1/$5 pro 1M Tokens
- Anthropic Prompt Caching Docs: 90% Rabatt auf Cache Hits, 5 Min TTL
- Anthropic Claude Code Subagents Docs: CLAUDE_CODE_SUBAGENT_MODEL Setting
- Google Gemma 4 Blog + HuggingFace: 26B MoE, 3.8B aktive Params, 256K Context
- Claude Code Router (GitHub, 26k+ Stars): Task-basiertes Model Routing
- ccusage (GitHub, 12k+ Stars): Token Usage Tracking CLI
- everything-claude-code (GitHub, 142k Stars): Community Guide mit Token Optimization Docs
- Sabrina.dev: "6 Ways I Cut My Claude Token Usage in Half"
- claudefast.com: Claude Code Context Window Optimization Guide