Contesto, memoria e costi ◆

Come funziona davvero la finestra di contesto, come si comporta la cache, dove vive la memoria e come non bruciare token e limiti di piano.

Aggiornato: luglio 2026 · Fonte: documentazione ufficiale Anthropic (code.claude.com/docs) · Vol. 3 della serie bignami Claude Code.

1. La finestra di contesto e la cache

perché certe azioni costano e altre no

Il modello non ricorda nulla tra una richiesta e l'altra: a ogni tuo messaggio Claude Code ri-invia **tutto** — system prompt, contesto di progetto, l'intera conversazione, il messaggio nuovo. Il **prompt caching** è ciò che evita di rielaborare la parte identica: l'API confronta l'inizio della richiesta (il *prefisso*) con quanto già processato di recente. Il match è **esatto**: una modifica in un punto qualsiasi del prefisso ricalcola tutto ciò che viene dopo. Per questo Claude Code ordina la richiesta mettendo prima ciò che cambia raramente:

Layer	Contenuto	Cambia quando...
System prompt	Istruzioni core, definizioni dei tool, output style	Un MCP si connette/disconnette, o Claude Code viene aggiornato
Contesto progetto	CLAUDE.md, auto memory, regole non scoped	Avvio sessione, o dopo /clear e /compact
Conversazione	Messaggi, risposte, risultati dei tool	Ogni turno (si **appende** in coda: la cache regge)

✗ Azioni che invalidano la cache (turno lento e costoso)

- **Cambiare modello** (/model): ogni modello ha la sua cache; la richiesta successiva rilegge tutta la history senza hit. Nota: `opusplan` cambia modello a ogni toggle del plan mode.
- **MCP che si connette/disconnette** a metà sessione (anche da solo: processo stdio che esce, sessione HTTP scaduta, riconnessione automatica).
- **Deny di un intero tool** (regola nuda tipo `Bash`): rimuove il tool dal contesto. Le deny scoped come `Bash(rm *)` invece non toccano la cache.
- **/compact**: per design sostituisce la history con un riassunto (ma la richiesta di riassunto riusa la cache: il costo vero è la generazione, non il miss).
- **Aggiornare Claude Code**: cambia il system prompt. Riprendere una sessione lunga dopo un upgrade rielabora tutta la history: può essere la richiesta più costosa che mandi.

✓ Azioni che la preservano

- **Modificare file del repo**: i contenuti entrano solo quando Claude li legge; l'edit viene segnalato con un reminder appeso.
- **Modificare CLAUDE.md a metà sessione**: la cache regge... ma **la modifica non si applica**! Il file è letto una volta all'avvio; il nuovo contenuto arriva al prossimo /clear, /compact o riavvio. Stessa cosa per l'output style.
- **Invocare skill e comandi**: iniettano istruzioni come messaggi in coda.
- **Cambiare permission mode** (Shift+Tab) e **cambiare effort**: cache-safe.
- **/rewind**: tronca a un prefisso **già in cache** — se hai preso una strada sbagliata, riavvolgere costa meno che compattare.
- **Subagenti**: lavorano in una finestra separata, la tua resta intatta.

▶ La regola pratica

Scegli il modello e connetti i server MCP **a inizio sessione**, poi non toccarli più. Riserva `/compact` alle pause naturali tra un task e l'altro (con istruzioni: `/compact mantieni i pattern di error handling`) invece di aspettare l'auto-compattazione a metà lavoro. Meno cambi a metà task = più cache hit = sessione più veloce ed economica. Controlla cosa riempie la finestra con `/context`.

2. CLAUDE.md, regole e auto memory

la memoria persistente del progetto

Ogni sessione parte da zero; due meccanismi portano conoscenza tra le sessioni, entrambi caricati all'avvio. Attenzione: sono **contesto, non configurazione applicata a forza** — per bloccare un'azione a prescindere da cosa decide Claude serve un hook PreToolUse o una regola di permesso.

	CLAUDE.md	Auto memory
Chi scrive	Tu	Claude (impara da correzioni e preferenze)
Contenuto	Istruzioni e regole	Apprendimenti e pattern scoperti

Scope	Progetto, utente o organizzazione	Per repository, condivisa tra i worktree
Caricamento	Ogni sessione, per intero	Ogni sessione (prime 200 righe o 25KB)
Usa per	Convenzioni, comandi di build, architettura	Insight di debug, comandi scoperti, preferenze

◆ Dove vivono i CLAUDE.md (in ordine di caricamento)

Scope	Posizione	Per cosa
Managed (org)	`/etc/claude-code/CLAUDE.md (Linux)`	Standard aziendali via MDM/Ansible; non escludibile dai singoli
Utente	`~/.claude/CLAUDE.md`	Preferenze personali per tutti i progetti
Progetto	`./CLAUDE.md` o `./.claude/CLAUDE.md`	Condiviso col team via git
Locale	`./CLAUDE.local.md`	Preferenze personali di progetto; mettilo in .gitignore

Claude Code **risale l'albero delle directory** dalla working directory raccogliendo tutti i CLAUDE.md; i contenuti si **concatenano** (dalla radice verso il basso, quindi i più vicini a te vengono letti per ultimi). Quelli nelle sottodirectory si caricano on demand quando Claude tocca file lì dentro. In un monorepo, `claudeMdExcludes` salta i file di altri team. I commenti HTML `<!-- -->` vengono strippati prima dell'iniezione: note per i manutentori a costo zero di token.

Scrivere istruzioni efficaci

- **Sotto le 200 righe**: file più lunghi consumano contesto e riducono l'aderenza.
- **Specifico e verificabile**: "usa indentazione a 2 spazi", non "formatta bene"; "lancia npm test prima di committare", non "testa le modifiche".
- **Coerente**: due regole in conflitto = Claude ne sceglie una a caso. Rivedi periodicamente.
- **Import** con `@path/to/file` (fino a 4 livelli; dentro i backtick resta testo letterale). `@AGENTS.md` in cima riusa le istruzioni di altri agent tool.
- Aggiungi una riga quando Claude sbaglia la stessa cosa **due volte** — le procedure multi-step vanno invece in una skill.

.claude/rules/ — istruzioni modulari

File markdown per argomento (`testing.md` , `security.md` ...), scoperti ricorsivamente, anche via symlink condivisi tra progetti. Il superpotere è il frontmatter `paths` :

```
---
paths:
  - "src/api/**/*.ts"
---
# Regole API
- Ogni endpoint valida l'input
- Formato errori standard
```

Le regole con `paths` si caricano **solo quando Claude lavora su file corrispondenti**: contesto pulito e token risparmiati. Regole utente in `~/.claude/rules/` valgono ovunque (priorità più bassa delle regole di progetto).

3. Ridurre i token

le leve, dalla più efficace in giù

Strategia	Come
Pulisci tra i task	`/clear` quando passi a lavoro non correlato: il contesto stantio costa su **ogni** messaggio successivo. Prima fai `/rename` , così ritrovi la sessione con /resume.
Modello giusto per il task	Sonnet regge bene la maggior parte del coding e costa meno di Opus; riserva Opus ad architettura e ragionamento multi-step. Per i subagenti semplici: `model: haiku` .
Effort e thinking	L'extended thinking è fatturato come output e il budget di default può valere decine di migliaia di token a richiesta: per task semplici abbassa con `/effort` o disattiva il thinking in /config.
Subagenti per il rumore	Test, log, fetch di documentazione: delegali a un subagente, in main torna solo il sunto.
CLI > MCP dove possibile	`gh` , `aws` , `gcloud` , `sentry-cli` non aggiungono definizioni di tool al contesto. Disabilita da /mcp i server che non usi; le definizioni MCP sono comunque deferite di default (tool search).
Hook di preprocessing	Invece di far leggere a Claude 10.000 righe di log, un hook filtra con grep e restituisce solo le righe con ERROR: da decine di migliaia di token a centinaia.
Da CLAUDE.md a skill	Le istruzioni di workflow specifici (review PR, migrazioni DB) caricate a ogni sessione sono token sprecati quando fai altro: spostale in skill on-demand.
Prompt specifici	"Migliora questo codebase" scatena scansioni ampie; "aggiungi validazione input alla funzione login in auth.ts" lavora con letture minime.

Plan mode e correzioni precoci	Shift+Tab per pianificare prima di implementare; se Claude sbaglia strada, **Esc subito** e /rewind: il rework è la spesa più evitabile.

Quanto costa, in pratica

Su deployment enterprise via API la media è ~13$ per sviluppatore per giorno attivo (150-250$/mese), con il 90% degli utenti sotto i 30$/giorno. Su abbonamento (Pro/Max) il costo in dollari di /usage è solo informativo: contano le barre dei limiti di piano, con breakdown per skill, subagenti, plugin e server MCP (tasti `d`/`w` per 24h/7gg). Esiste anche un piccolo consumo di fondo (<0,04$/sessione) per riassunti e comandi di stato.

Occhio agli agent team

Ogni teammate è un'istanza Claude completa con la sua finestra: in plan mode il consumo arriva a ~7× una sessione normale. Se li usi: Sonnet per i teammate, team piccoli, prompt di spawn asciutti, e spegni i teammate quando hanno finito.

Fonti: code.claude.com/docs — prompt-caching, memory, costs, context-window · /context e /usage sono i tuoi strumenti diagnostici quotidiani.