PRACTICAL GUIDE / 2026.11

A practical guide to reducing token usage

Claude / Gemini / ChatGPT

Chat, Claude Code, Codex, Gemini CLI and Antigravity

- Less irrelevant context
- Fewer repeated attempts
- Measurable usage and cost

English edition | 25 pages
Official sources checked: 17 November 2026

Reading map and quick start

This guide covers Claude and Claude Code, ChatGPT and Codex, Gemini chat and Google's coding tools. Both language editions contain the same scope, examples and sources. API chapters are included for readers building their own integrations.

Section	Pages
Foundations, priorities and prompt design	3-6
Claude, ChatGPT and Gemini chat	7-9
Coding workflow and product playbooks	10-14
Instruction files, tools and API costs	15-18
Documents, prompt templates and measurement	19-21
Troubleshooting and rollout plan	22-23
Clickable official sources	24-25

What to do in the first 15 minutes

- Choose one outcome: for example, a three-option decision table or a fix that passes a named test.
- Select only the necessary files, data range and reproducible error.
- Specify output length and format; avoid requesting the same information in three formats.
- Start a fresh chat for an unrelated task; do not paste the same history back into an ongoing task.
- Check the coding tool's usage screen and record model, effort level and enabled tools.

> Method: Product behavior is supported by numbered official references. Templates, starting ranges and workflows are recommendations from this guide, not guaranteed savings. Sources checked: 17 November 2026.

Your subscription, organization settings and client version can change availability. Command examples apply to the named CLI; do not assume that they work in a consumer chat box or every IDE extension.

Tokens, context, quotas and cost

A **token** is a unit used to process text and other input. One word may contain several tokens. Language, the model's tokenizer, code, numbers and punctuation all matter. Do not apply rough English character or word ratios directly to Turkish. [01]

Concept	What it measures	How to reduce it
Input	Instructions, selected history, files, tool results	Remove irrelevant context
Output	Answers, generated code and tool calls	Define the required deliverable
Reasoning	Hidden thinking used by some models	Match model and effort to task
Context window	Working space available to one request	Summarize or start a new session
Quota / credits	The product's allowance accounting	Use the product's own meter

Keep three objectives separate

- **Fewer tokens:** smaller input, shorter output and fewer retries.
- **Lower API bills:** model rates, caching and tool charges matter alongside token volume.
- **A longer-lasting subscription allowance:** this depends on the provider's quota rules; token savings do not guarantee proportional quota savings.

Input is approximately the combination of system instructions, project rules, relevant history, retrieved content and the new message. A tool-using agent can make many model calls for one user message. Message count alone therefore does not explain completed work or cost.

> "Answer briefly" can reduce visible output. It does not automatically shrink the history sent to the model or hidden reasoning. OpenAI documents reasoning tokens as billable output in the API. [24]

Prioritize the largest savings

Before deleting a few words from a prompt, inspect the complete workflow. Reading the wrong files, vague scope and failed attempts are often larger cost drivers than small wording differences. The order below is a starting recommendation; validate it with your own measurements.

Priority	Action	Watch for
1	Define the outcome and acceptance criteria	Missing requirements cause rework
2	Select relevant files, pages and lines	Do not cut critical dependencies
3	Separate unrelated tasks	Prepare a handoff summary
4	Reduce repeated reads and tool calls	Refresh evidence when necessary
5	Use suitable model and effort	Track error rate as well
6	Narrow the output format	Keep evidence and validation
7	Evaluate API caching and batch processing	Separate token cuts from discounts

Route by task difficulty

- Simple transformation, titles or classification: start with a fast, economical option available in your account.
- A bounded bug fix or small feature: try a balanced model with normal effort.
- Architecture, subtle failures or security-sensitive changes: use a stronger model with sufficient reasoning depth.

When an economical model fails, avoid repeatedly regenerating the same vague prompt. Identify the reason briefly, add the missing evidence or escalate the model. A stronger model can be cheaper per completed task when it prevents retries.

> Optimize total usage per accepted result. The shortest prompt and lowest single-call price are not sufficient measures of a good workflow.

Write concise, complete prompts

A useful prompt has five parts: **goal, necessary context, boundaries, output format and acceptance criteria**. Not every task needs an elaborate persona or many examples. Shorten the initial request without losing meaning; preserve details that affect decisions. Clarity and sufficient context also align with ChatGPT's official guidance. [11]

Weak starting point

```
Inspect this project, improve everything and explain in detail.
```

A more efficient starting point

```
Goal: Fix duplicate submission on /checkout.
Evidence: A fast double click creates two POST requests.
Scope: Checkout component and related test; keep API unchanged.
Accept: One POST; retry still works after an error.
Output: Change summary, test result, remaining risk; max 150 words.
```

Keep follow-ups specific

- Instead of "rewrite it": "Simplify paragraph two; preserve the other sections."
- Instead of "explain everything again": "Give only the decisions that changed from the previous proposal and why."
- Instead of "make it better": "Compare two options by setup time and maintenance burden, then choose one."

Group related questions in one request: "Using the same data, give a summary, three risks and a recommendation." Combining ten independent jobs into a giant prompt can instead obscure priorities and create expensive corrections. Batch work that shares inputs and acceptance criteria.

> Suggested starting point: 100-200 words or 3-5 bullets for a short chat answer. This is not a product limit. Increase the budget when explanation is necessary; an answer that is too short can create extra turns and erase the savings.

Manage chats and handoffs

Continue in the same chat when you are refining the same decision. Start fresh when the topic, dataset or outcome changes. Avoid fixed rules such as resetting every five messages; compare the context you must preserve with the irrelevant history you are carrying.

Situation	Preferred action
Editing one section of the same document	State only the change in the existing chat
New customer, repository or topic	Start fresh with a short relevant brief
Long task with accumulated tool output	Use supported compaction or a handoff
One incorrect assumption in the previous result	Correct that assumption without resending everything
Exact wording, numbers or citations matter	Preserve original evidence separately

Copyable handoff request

```
I will continue this task in a fresh chat. In at most 250 words:
1. Goal and acceptance criteria
2. Confirmed decisions and constraints
3. Relevant files/pages/ranges and version
4. Completed work, test results and open issues
5. The single next step
Preserve exact error messages and critical values.
Exclude secrets, old logs and abandoned options.
```

Read and correct the summary before transferring it with the essential evidence. A filename alone may be insufficient: confirm that the new environment can access the file. A compact summary is not a substitute for all original source material.

> Summarization can consume tokens and lose detail. Use it at natural milestones rather than constantly. A new chat or cleared history does not refund allowance already consumed.

Claude chat playbook

Claude usage depends on message length, attachments, conversation history, model, effort and tool-based work. Anthropic documents caching benefits when reusing Project content and notes that caches can expire after inactivity. Do not interpret this as a fixed discount or unlimited usage. [02]

A workflow for documents and writing

- Create a Project for one client or subject; add only reference material you expect to reuse.
- Keep Project instructions focused on audience, language, tone and output format.
- Do not turn long, one-off directions into permanent rules for every conversation.
- Request the relevant section first: “Evaluate only the assumptions in section three.”
- For long writing, agree on the outline and one sample section before generating the rest.
- On the next turn, request changed paragraphs rather than regenerating the entire document.

Example prompt

```
Use proposal-v3 in this Project.
Review only scope, delivery date and exclusions.
Output: At most 5 ambiguities, each with a section name
and one suggested replacement sentence. Do not write a new proposal.
```

Start a new chat when work shifts from editing one report to creating a report for a different client. Keeping durable references in a Project can reduce repeated uploads in the interface; it does not make the model’s context free or tokenless.

Checkpoint

Before starting, note the plan or model usage display. After completion, evaluate it alongside model choice, attachments, tool-based stages and retry count. Do not infer total token consumption from the final answer’s word count alone.

> These chat recommendations are not Claude Code commands. Do not treat CLI commands such as `/compact`, `/context` and `/usage` as universal controls in claude.ai chat.

ChatGPT chat playbook

ChatGPT Projects group related chats, files and instructions. Project memory and file access depend on settings and plan. Starting another chat does not guarantee that all other Project context disappears. Do not equate Projects with a guaranteed API cache discount. [10]

A practical daily workflow

- Use an available fast option for routine transformation or short drafts; choose suitable reasoning for complex decisions. Availability varies by account.
- Avoid permanent instructions such as “think deeply and explain at length” for every task. Match the answer format to the need.
- Use web research for current information or sources; do not request unnecessary searches for a simple edit of supplied text.
- For tables and calculations, identify the relevant sheet, columns, date range and expected calculation.
- For coding in chat, request one working example rather than a long tutorial, five alternatives and a complete project simultaneously.

Example prompt

```
Use only 2026 Q3 and Q4 in the attached sales table.
Identify missing data first. Then calculate revenue difference
and percentage change. Output: 1 small table and at most
3 comments. State which columns were used.
```

When no token meter is available, do not invent a precise remaining-token count or treat the model’s estimate as a bill. Record plan notices, selected model and completed work. For file- or tool-based work, use a workflow that recognizes that visible message count is not the whole consumption story.

> ChatGPT chat, Codex CLI and the API are different surfaces. Typing `/status` or an API parameter such as `max_output_tokens` into a regular chat box does not turn it into a product control. See page 12 for Codex.

Gemini chat playbook

Google describes Gemini Apps limits as compute-based, influenced by prompt complexity, model and feature choice, and chat length. Shorter messages therefore do not produce a fixed increase in allowance on every plan. Do not automatically apply personal-account documentation to work or school accounts. [12]

Gems and recurring tasks

Gems help create reusable instructions and context for particular jobs. Google recommends considering persona, task, context and format. Creating a Gem does not mean that usage is free or that every request uses fewer tokens. [13]

- Put short, recurring work rules in Gem instructions; keep one-off data dumps out.
- When using Drive or attached files, name the exact document, version and relevant section. Confirm access and appropriate sharing.
- Specify when you want only a summary of supplied sources. For fresh research, define the search range and decision question.
- For a long PDF, choose the primary deliverable instead of requesting a summary, translation, quiz, presentation and report at once.

Example prompt

```
Apply this Gem’s writing rules. Use only the Executive Summary
and Conclusions sections of the attached report.
Audience: a nontechnical manager.
Output: a 120-word summary + source section names.
Do not add figures or explanations absent from the source.
```

When reformatting the same data, say “preserve the figures from the previous result; change only the format.” For a new subject, transfer essential details in a short handoff; do not assume automatic access to the full previous conversation.

> Gemini chat and Google’s coding products require separate treatment. Pages 13-14 cover the 2026 Code Assist / Gemini CLI transition and current Antigravity guidance. [14]

A shared coding-agent workflow

Coding-agent consumption is not just generated code. Repository discovery, file reads, terminal output, tool schemas, failure loops and long history also matter. State the outcome, execution environment and verification criteria early to reduce unnecessary exploration.

Discover > change > validate > finish

- **Discover:** Search filenames and symbols before reading relevant files. Avoid printing the entire repository.
- **Narrow:** Provide the failing input, expected and actual behavior, and reproduction steps.
- **Plan:** For work across several files, request a short file-and-change plan. Avoid adding a planning turn for a trivial edit.
- **Change:** Make the smallest coherent fix; keep unrelated refactoring out of the task.
- **Validate:** Run relevant behavior checks and broaden coverage when the results justify it.
- **Finish:** When acceptance criteria pass, stop with a summary, evidence and remaining risks. Avoid an endless “improve it a little more” loop.

Examples of scoped, read-only commands

```
rg --files -g "*checkout*" -g "!node_modules" -g "!dist"
rg -n "submitOrder|isSubmitting" src tests
git diff --stat
git diff -- src/checkout.tsx
```

Adapt paths and symbols to your project. Find test commands in the README or package scripts instead of making the agent guess. Keep large test logs locally and send the failing test name, first relevant error and necessary lines to the model.

> Shortening output does not mean skipping tests. Do not hide failed validation or omit critical failure details to meet a token target. Work closed without noticing a failure can be more expensive in the next round.

Claude Code playbook

The current cost documentation uses `/usage` for session tokens and plan usage. Older versions and some help pages may show `/cost`; check your installed version's help. `/context` inspects context and `/model` shows available model choices. [03]

Goal	CLI command / approach
Inspect usage	/usage
Inspect context	/context
Summarize the same long task	/compact with preservation instructions
Switch to unrelated work	/clear
Match the model to the task	/model

Do not treat a session dollar estimate as a subscription invoice. Use plan indicators for subscriptions and the provider's billing dashboard for API spending. Clearing a session does not refund usage already consumed. [03]

Keep project instructions lean

Use `CLAUDE.md` for run/test commands, important paths and real constraints. Current Claude Code documentation also describes `AGENTS.md` support; verify your version and avoid duplicating the same rules in both files. [04]

Move long procedures needed only occasionally into a skill or task document rather than embedding them in the main file. Skills provide reusable task instructions; their content can still enter context when invoked. [05]

Example compaction request

```
/compact Preserve acceptance criteria, changed file paths,
exact failing test names, key decisions and the next step.
```

> Practical routine: inspect usage and context before a task, then check them again afterward. Research, tool results and reasoning may remain substantial even when the visible answer is short. If a command is unavailable, do not assume an equivalent from another product.

Codex playbook

In Codex CLI, `/status` can show remaining usage limits. Indicators and panels vary by client. OpenAI recommends reducing unnecessary context, source volume, large instruction files and unused MCP servers. [08]

Goal	Codex CLI
Inspect remaining allowance	/status
Summarize a long chat	/compact
Start a new chat	/new
Explore an alternative branch	/fork; copies existing history

`/compact` replaces earlier turns with a short summary. `/new` starts a fresh chat. Since `/fork` copies the existing chat, it should not be treated as a history-reset savings command. Desktop and IDE versions may expose these actions differently. [07]

AGENTS.md and task boundaries

Codex discovers project instructions hierarchically; broader rules and rules nearer the working directory may both apply. Do not expand the file merely because a large limit exists. Keep general defaults short and put component-specific rules in the appropriate subdirectory. [09]

```
Task: Fix refund amount rounding.
Start with: src/refund and tests/refund.
Accept: Remove the 0.01 discrepancy; preserve currency behavior.
Check relevant instructions and the test first.
Keep unrelated files unchanged. Deliver: fix + test evidence.
```

Execute small, well-defined work directly. For ambiguous or large work, use a brief plan to establish file scope. Define completion through acceptance criteria to help prevent endless feature additions or refactoring. [06]

> Match the app's reasoning effort and model choice to the job. Avoid making the highest setting a permanent default. A faster service option does not automatically mean fewer tokens or a lower price.

Gemini CLI and Code Assist: version boundaries

Current-status note: Google's migration notice states that from 18 June 2026, legacy IDE extensions and Gemini CLI stopped serving the individual Code Assist, Google AI Pro and Ultra tiers through Login with Google. Affected users are directed to Antigravity. This does not mean all Gemini API use ended. [14]

New Code Assist Standard/Enterprise subscriptions also stopped being sold on 9 October 2026. Existing contracts have a separate support and renewal schedule. Identify the product and contract behind your account before budgeting with old "free daily request" figures. [15]

For eligible setups still using Gemini CLI

Goal	Gemini CLI command
Inspect session / model usage	/stats or /stats model
Summarize history	/compress
Start a fresh session	/clear
Inspect loaded instructions	/memory show

`/clear` clears active conversation context; `Ctrl+L` only clears the display. `/compress` replaces history with a summary. Do not assume identical Antigravity behavior for these commands. [16]

`GEMINI.md` stores recurring instructions; `/memory show` inspects the loaded instructions. Oversized global rules can create unnecessary context for every job. [31]

`.geminiignore` narrows file scope for tools that honor it; it is not a security boundary blocking every access path. Exclude generated output without accidentally excluding necessary source and test files. [17]

> The Gemini CLI caching page distinguishes API-key / Vertex AI setups from OAuth setups. Cache statistics do not establish an equivalent discount for ordinary Gemini chat quotas. Read the older page alongside current authentication policy. [18]

Current Google coding workflow: Antigravity

Google's current migration guidance makes Antigravity a necessary addition. Identify whether you are using Antigravity CLI, an IDE extension or legacy Gemini CLI, then interpret commands and usage reports for that surface. [14]

Control	Antigravity CLI
Active context and categories	/context
Reasoning effort	/effort; supported models only
AI Premium credit balance / history	/credits
Fresh chat for unrelated work	/clear; alias /new

`/context` shows categories such as user messages, agent responses, tools and loaded customizations. Older turns can be compacted automatically into checkpoint summaries. Use this view to inspect the active working context rather than as a total billing statement. [19]

`/effort` adjusts reasoning for supported models. In interactive use, `/effort low` can be tried for simple work; complex tasks may need greater effort. Headless parameters differ, so check the CLI documentation. [20]

`/credits` displays credit balance and consumption history. Do not equate credits with raw token counts. Record the balance report separately from the active context for an individual task. [21]

Simplify instructions during migration

The migration guide says existing `GEMINI.md` and `AGENTS.md` rules remain usable and identifies `.agents/skills/` as the workspace skill path. Remove duplicate instructions before migration; do not activate every legacy extension simply because you have moved. [22]

> Starting experiment: complete one small fix with one agent and only the tools required. Preserve acceptance criteria while changing either effort or tool count, one at a time. Do not assume a savings percentage that you have not measured in your current account.

Keep persistent instructions small

Moving instructions into a file reduces manual repetition; it does not eliminate token cost. All or part of the file may be loaded into sessions. A useful instruction file is a short, current working agreement that enables correct behavior.

Tool	Common file	Savings approach
Claude Code	CLAUDE.md; current AGENTS.md support	One authoritative rule set; conditional detail [04]
Codex	AGENTS.md	Short root file, local subdirectory rules [09]
Gemini CLI	GEMINI.md	Separate global and project instructions [31]
Antigravity CLI	GEMINI.md / AGENTS.md	Deduplicate during migration [22]

A short, adaptable example

```
# Project working rules
- Purpose: Payment UI and its service.
- Relevant paths: src/payments, tests/payments.
- Test: [command verified in this repository].
- Preserve the existing monetary rounding rule.
- Explain impact before changing the public API.
- Done: relevant tests and change review completed.
- Final reply: change, validation, remaining risk.
```

- **Keep:** actual test commands, important architecture constraints and specific rules preventing recurring mistakes.
- **Remove:** old session logs, repeated rules, motivational boilerplate and invalid file paths.
- **Load on demand:** rare deployment procedures, long style guides and specialized data migration steps.

Even a one-page file can be costly if its rules conflict. Resolve “always run every test” and “run only the relevant test” into one clear principle. Preserve security and data-integrity requirements rather than deleting them merely to save space.

> Recommendation: start with a small rule set and add items only for recurring real problems. Imports and links do not guarantee lazy loading in every product; inspect the active context.

Tools, MCP, logs and multiple agents

Tool definitions, call arguments and returned results can add to context. Some systems load tools on demand; others expose broader schemas up front. Avoid universal claims such as “one MCP server costs exactly this many tokens.” Measure active context and actual task usage.

A more efficient tool contract

- Limit searches by directory, date range, file type or result count.
- Request only required fields rather than complete records: ID, status, date and failure reason.
- Filter or aggregate large datasets locally, then send the small decision-relevant result.
- Keep tool results stable; avoid returning identical data in a different order.
- Consider disabling unused connections for the task, without removing a useful tool and creating repeated manual copying.

When do multiple agents make sense?

Parallel work can reduce elapsed time without reducing total tokens. Each agent may carry instructions and context, generate coordination messages and require result synthesis. Several agents are often an unnecessary starting point for a single-file fix.

Useful decomposition	Costly decomposition
Independent modules with separate acceptance criteria	Three agents reading the same files from scratch
A narrowly scoped specialist review	Copying the full chat to every agent
Reading large output and returning concise evidence	Sending all subagent logs back to the parent

> Recommendation: a delegated task should carry only its goal, relevant files, boundaries and result format. Request finding + file/line + evidence + uncertainty. Report time saved separately from total consumption.

Disabling necessary security checks or permission boundaries is not token optimization. The objective is to remove unnecessary steps and output while still completing the work correctly.

API: caching and usage reporting

This chapter applies to your own API integration. An API setting cannot be enabled by typing it into the same brand's consumer chat. Caching generally reduces cost or latency for repeated input; it does not remove the content's semantic length or context requirements.

Provider	Practical approach	Inspect
OpenAI	Put stable content first and the changing task last; follow the model's current cache rules. [23]	Input, cached input and output breakdown
Claude	Apply appropriate cache_control to stable repeated content; evaluate write and read charges separately. [28]	input_tokens + cache_creation_input_tokens + cache_read_input_tokens
Gemini	Use implicit caching opportunities; verify explicit-cache support for the model and API. [26]	Cache and other usage categories

Current Gemini documentation describes implicit caching for the Interactions API and Gemini 4; do not assume support for explicit cache objects there. Avoid copying older model or endpoint examples unchanged. [26]

Measure at the right layer

- Preflight counting: assess expected input size; this is not the final total bill.
- Response usage object: inspect actual request consumption; field names vary by API and version.
- Billing or usage dashboard: confirm monetary cost, including tools, cache and service conditions.

Claude's token-counting endpoint and Gemini's `count_tokens` method help assess large input before generation. A token count from one provider is not an exact measurement for another. [25][27]

> Do not pad prompts just to reach a cache minimum. Include initial writes, expiry, rebuilds and any storage charges in the calculation. A cache hit alone does not prove that the workflow is cheaper.

API budgets and a worked example

Prices can vary by model, service tier, endpoint, cache and context size. The following figures are entirely hypothetical and are not current prices from any provider. Use real rates and usage fields from your account.

```
C = (U × Pu + R × Pr + Wc × Pw + O × Po) / 1,000,000
    + tool fees + any storage / other charges
U: uncached input; R: cache reads; Wc: cache writes
O: total billable output under the provider's rules
```

Do not double-count categories. If reasoning is already a subset of total output, do not add it again. Check whether cache-write charges are reported separately or within input in your provider's documentation.

Hypothetical single request	Before	After
Uncached input U	40,000	12,000
Output O	4,000	1,500
Rate: input $2 / M, output $8 / M	Same	Same
Calculated cost	$0.112	$0.036
Total tokens in this simple example	44,000	13,500

Cost falls by approximately 67.9% in this scenario; tokens fall by approximately 69.3%. These are different percentages. The example excludes cache, additional tool charges and other hidden categories. It does not establish a universal savings rate for shorter answers.

Make budgets part of the workflow

- Set estimated token and cost budgets per request or task, with an explicit overrun policy.
- Allow enough output for completion; truncation followed by regeneration can increase cost.
- Use bounded retries for network failures and an identifier to prevent duplicated work.
- Consider supported Batch options for independent, nonurgent API jobs. This is a price/time tradeoff, not a reduction in token count. [29]

> If a budget is exceeded, save the partial result and remaining work instead of silently degrading quality. Do not convert real subscription quotas using this hypothetical dollar calculation.

PDFs, tables, images and long content

Uploading a large file does not mean every task processes its entire contents in the same way. Products can use different extraction, vision and retrieval methods. File size alone is not a token count; identify relevant pages or ranges whenever possible.

Content	More efficient approach	Preserve
Long PDF	Select sections/pages; separate repetitive appendices	Table titles, footnotes and sources
Excel / CSV	Select columns and dates; aggregate locally	Units, missing values, formula logic
Screenshot	Provide a readable view of the relevant region	Error text and necessary surroundings
Log	Send the first relevant error and limited surrounding lines	Version, time, correlation ID
Audio / video	Use a relevant interval or accurate transcript	Speaker, timestamps, necessary audiovisual evidence

Read in two stages

First map the source: “Which sections are needed to answer this decision question?” Then inspect the selected sections in detail. For a small document, the extra mapping turn may be wasteful; use this technique for genuinely large sources.

Clean text can replace a screenshot when it preserves the meaning. Keep the image when layout, a chart or visual state is part of the problem. Avoid the rule “lower resolution is always cheaper”; processing and tokenization depend on the model. Gemini’s counting guide explains that nontext input is tokenized as well. [25]

Example source boundary

```
Use only pages 12-18 and Table 4.
Tie each conclusion to a page or table name.
Flag missing data. If another part of the document is needed,
state which section is necessary and why before expanding scope.
```

> Exclusion is not a security mechanism. Remove unnecessary personal fields locally before sharing sensitive data. Do not distort results by dropping relevant footnotes or exceptional records merely to save tokens.

Copyable prompt library

Replace bracketed fields with your own information. These are task instructions, not product commands. Adapt them to any of the three providers' chats or a suitable coding agent.

1. A short answer bounded by sources

```
Goal: [decision question].
Sources: [file/section/data range].
Answer only from these sources; label inference separately.
Output: [3 bullets / small table], at most [150] words.
Do not invent missing facts; state only critical gaps.
```

2. Prevent unnecessary regeneration

```
Change only [section] of the previous output.
Correction: [specific instruction].
Preserve figures, sources and other decisions.
Return only the changed part, not the entire artifact.
```

3. A narrowly scoped bug fix

```
Failure: [input, expected, actual].
Starting files: [paths].
Accept: [observable behavior + relevant test].
Verify relevant files and the cause first.
Make the smallest coherent fix; run relevant tests.
Output: change, test evidence, remaining risk.
If two different approaches fail for the same reason,
stop the loop and report the blocker and evidence needed.
```

4. Bounded research

```
Decision: [choice]. Geography / period: [boundary].
Start with at most [5] relevant primary sources.
Expand if decision-changing evidence is missing.
Output: 3 findings, uncertainties, recommendation and links.
Do not keep summarizing pages that repeat the same information.
```

> Fixed thresholds such as five sources or two attempts are starting preferences. Explicitly raise them when complexity or failure risk demands it. Combine these templates with the handoff request on page 6.

Measure actual savings

Test the assumption “I wrote less, so I used less” with a controlled comparison. Use the same task, data and acceptance criteria. Do not derive a universal percentage from one unusually easy result.

A small A/B experiment

- Select 5-10 representative tasks: short writing, a document question, a small bug and a multi-file change.
- Use the existing workflow in A and one improvement in B, such as narrower source scope.
- For code, use the same starting revision and a clean working state; avoid letting an earlier solution make the next run easier.
- Keep model, effort and enabled tools unchanged. Record cold versus warm cache conditions separately.
- Define success in advance, then compare total consumption, time, retries and quality.

Field	Example / explanation
Product, client, date and model	CLI version; normal / high effort
Task and acceptance outcome	Pass / partial / fail
Input / cache / output / reasoning	Only trustworthy reported fields
Tool calls and retries	Include failed attempts
Elapsed time / real cost	API dashboard; visible subscription quota change
Quality and defects	Tests, citation accuracy, required corrections

```
Savings (%) = 100 × (baseline usage - new usage) / baseline usage
Usage per success = usage across all attempts / accepted tasks
Cache share = cache reads / total input [same API and definition]
```

If eight tasks produce six accepted results, do not count only the six successful calls; failed attempts belong in total usage too. If the subscription display is rounded or token details are unavailable, label the result as an approximate allowance change.

> Raw token counts across providers do not independently rank quality or price. Tokenizers, models, caches, tool chains and accepted outcomes can differ. Compare the same completed job.

Troubleshooting and common myths

Symptom / claim	Check and correction
“My prompt was short; usage was high.”	Inspect history, files, reasoning and tool calls.
“A new chat resets my quota.”	It starts fresh context; prior consumption is not refunded.
“Cached content has no tokens.”	Caching may discount repeated input; context still exists.
“English is always cheaper.”	Tokenizers and tasks differ; measure meaning lost in translation.
“Lowest effort is always best.”	Compare failures and total retry consumption.
“Multiple agents save tokens.”	Elapsed time may fall while total usage rises.
“Ctrl+L deletes history.”	In Gemini CLI it clears the display; /clear is separate. [16]
“Read the whole repo first.”	Discover files and symbols, then read relevant sources.
“AGENTS.md makes it free.”	Loaded instructions occupy context; keep them short and current.
“One command works everywhere.”	Verify product, CLI/IDE surface and installed version.

Diagnose an unexpected increase in this order

- Did authentication or billing change, perhaps from subscription to API?
- Did model, reasoning effort or long-context behavior change?
- Was a new extension, MCP server, global rule or automated task added?
- Did a large file or test log accidentally enter context?
- Did the cache expire, or did a stable prefix change and require rebuilding?
- Did the agent enter a tool-and-fix loop around the same failure?

> Requesting a full chain of thought is not a savings technique. Ask for what you need: a short rationale, verifiable evidence, key assumptions and the result. Do not infer hidden reasoning volume from the length of the visible explanation.

A seven-day rollout and final checklist

Day	Action
1	Measure the current workflow; choose 5-10 tasks and acceptance criteria.
2	Add goal, scope and acceptance criteria to initial prompts.
3	Narrow files, pages and logs; compare quality.
4	Simplify persistent instructions; remove duplicates and stale rules.
5	Try model and effort options on the same tasks.
6	Measure tool chains and API cache behavior where applicable.
7	Compare usage per accepted result; retain what worked.

Before every task

- Is there one deliverable with observable acceptance criteria?
- Is the necessary context accessible and current?
- Are product, account type, model and effort known?
- Are output format, source scope and stopping conditions stated?

After every task

- Is the result correct, with required tests or source verification completed?
- Did the measurement include all attempts and tool consumption?
- Would a short handoff be sufficient for the next task?
- Does this work reveal a real need for a new persistent rule?

Short glossary

Context: working content used by the model for a request. **Tokenizer:** a system that splits content into token units. **Compaction:** converting older history into a smaller representation or summary. **Cache:** reuse of repeated content for computational or pricing benefits. **MCP:** a protocol for connecting models to tools and resources. **Effort:** a supported model setting that influences reasoning depth.

> The durable objective is less irrelevant context, fewer failed attempts and lower consumption per accepted result. Review rules monthly or after product changes rather than expecting a fixed savings percentage.

Official sources / 1

Numbers match the links in the text. Click a title to open the official page. All links were checked on 17 November 2026.

[01] OpenAI | Understanding and counting tokens
help.openai.com

[02] Anthropic | Usage limit best practices
support.claude.com

[03] Claude Code | Manage costs effectively
code.claude.com

[04] Claude Code | How Claude remembers your project
code.claude.com

[05] Claude Code | Extend Claude with skills
code.claude.com

[06] OpenAI | Codex / Work best practices
learn.chatgpt.com

[07] OpenAI | Developer commands (CLI)
learn.chatgpt.com

[08] OpenAI | Codex / Work pricing and usage
learn.chatgpt.com

[09] OpenAI | Custom instructions with AGENTS.md
learn.chatgpt.com

[10] OpenAI | Projects in ChatGPT
help.openai.com

[11] OpenAI | Prompt engineering best practices for ChatGPT
help.openai.com

[12] Google | Gemini Apps limits and feature availability
support.google.com

[13] Google | Tips for creating custom Gems
support.google.com

[14] Google | Gemini Code Assist consumer account deprecation
developers.google.com

[15] Google Cloud | Code Assist Standard and Enterprise sunset
docs.cloud.google.com

[16] Gemini CLI | CLI commands
geminicli.com

Official sources / 2

Numbers match the links in the text. Click a title to open the official page. All links were checked on 17 November 2026.

[17] Gemini CLI | Ignoring files (.geminiignore)
geminicli.com

[18] Gemini CLI | Token caching and cost optimization
geminicli.com

[19] Antigravity CLI | Context command
antigravity.google

[20] Antigravity CLI | Effort command
antigravity.google

[21] Antigravity CLI | AI credits command
antigravity.google

[22] Antigravity CLI | Migration from Gemini CLI
antigravity.google

[23] OpenAI API | Prompt caching
developers.openai.com

[24] OpenAI API | Reasoning models
developers.openai.com

[25] Gemini API | Understand and count tokens
ai.google.dev

[26] Gemini API | Context caching
ai.google.dev

[27] Claude API | Token counting
platform.claude.com

[28] Claude API | Prompt caching
platform.claude.com

[29] OpenAI API | Batch API
developers.openai.com

[30] Google Cloud | Code with Gemini Code Assist
docs.cloud.google.com

[31] Gemini CLI | Provide context with GEMINI.md
geminicli.com

> Product limits and commands can change. For Google's transition, [14]-[15] establish current account status; [16]-[18] describe eligible legacy CLI setups. When dates or versions differ, prioritize the current official migration notice and your installed version's help.