CONTEXT ENGINEERING

Context Engineering Manual

Optimizing AI Token Use, Subscription Capacity, and API Economics

Mike Hadaway

21 chapters in three parts, plus reference
About 45 minutes of reading
Last verified 4 September 2026

https://mikehadaway.info/context-engineering

Contents

REFERENCE

$200 Later

In the beginning, I was doing what many enthusiastic AI users do. I was exploring settings, connecting tools, and telling myself I was being financially responsible.

I had Claude Desktop and the roughly $20 monthly subscription. Then I found the usage credits setting.

Usage credits are the overflow valve. When a paid plan reaches its included usage limit, credits let you keep working by drawing down a prepaid balance at standard per-token rates instead of stopping (Manage usage credits for paid Claude plans). I turned it on, added a payment method, enabled automatic reloads, capped each reload at $10, and set an overall limit of $200.

Three numbers. It felt like three guardrails. I assumed the $200 would last for months.

It lasted about a week.

I forgot about the setting and went back to work. Claude did not forget. The reloads worked exactly as designed. That was the problem.

Look at what I had actually built. The $10 cap was not a brake. It was the size of each automatic purchase. The $200 was not a budget. It was how far the thing would travel before anyone had to steer. And the single setting that would have genuinely stopped it, the one that turns automatic reloads off, was the one I had gone out of my way to switch on.

The lesson was not that Claude had somehow stolen $200. The lesson was that I had configured a spending limit without designing a context strategy. I was paying for repeated context, unnecessary retries, oversized tasks, and the habit of reaching for maximum capability before deciding what the work actually required.

A limit tells you when to stop. It does not tell you what you were doing.

Something had to give.

This manual is what comes after that realization. It is about using AI deliberately. It is about choosing what the model should see, what it should ignore, what should persist, what should be retrieved, how much reasoning a task deserves, and how to measure whether the result justified the capacity consumed.

The Operator's Frame

The AI Operator works in four moves. This manual is organized around them, and every chapter is tagged with the one it serves.

Frame. Decide what result you need and what finished means, before you spend anything.

Ground. Decide what evidence the model sees and which source wins when two sources disagree.

Verify. Check the result against something other than its own fluency.

Govern. Set the boundaries, the review dates, and the spend, then keep them where someone else can find them.

Context engineering is the Ground move done carefully, with the other three holding it in place. A perfectly grounded answer to the wrong question is still waste.

How to Use This Manual

Nothing here is hidden. Every chapter is on the page at all times, and the reading level control at the top is a shortcut that jumps you to the right part rather than a filter that removes content.

The three parts are sorted by what you are paying for, not by how hard they are. That distinction matters, because the hardest idea in the manual sits in Part One and some of the easiest arithmetic sits in Part Three.

Foundations, chapters 1 through 8, is for anyone working inside a subscription.

The Discipline, chapters 9 through 14, is for anyone doing repeated work on the same material.

Systems, chapters 15 through 21, is for anyone whose usage is billed per token. That now includes usage credits, so it may be you even if you have never written a line of code.

Level labels elsewhere in the manual work the same way. When a recipe is marked Advanced or a checklist is marked for beginners, that is guidance about who will find it useful, not a lock. Read anything you want in any order.

Reference holds the material you come back to: the one-page card, the anti-patterns, the recipes, the checklists and worksheets, the decision trees, the glossary, and the sources.

Each chapter opens with a one-line summary, states a single rule, and closes with the way that rule usually fails. Every one of those failures is an instance of one of the six patterns in Chapter 11, which is the chapter to reach for when something has already gone wrong. If you read nothing else, read the rules, which are collected in one place at the back.

The Manual by Operator Phase

Every chapter serves one of the four moves. If you came here with a specific problem rather than a reading plan, start with the phase it belongs to.

PHASE	THE QUESTION IT ANSWERS	CHAPTERS
Frame	What result do I need, and what does finished mean?	4, 5, 13, 15, 19
Ground	What evidence should the model see, and which source wins?	2, 3, 7, 9, 10, 14, 18
Verify	How do I check this against something other than fluency?	8, 11, 21
Govern	What are the boundaries, the review dates, and the spend?	1, 6, 12, 16, 17, 20

Ground has the most chapters, and that is the honest shape of the discipline. Context engineering is mostly a grounding problem.

Currency of facts. The product behavior described here was checked on September 4, 2026. Model names, limits, prices, and plan entitlements change fast. The durable material is the reasoning. The date-sensitive material is flagged where it appears, and every vendor claim is linked to its official source in Source Notes. Recheck those links before making a purchase or a production decision.

FOUNDATIONS · 8 CHAPTERS · ABOUT 13 MINUTES

Part One: Foundations

CHAPTER 01 / 21 · GOVERN

Three Meters, Not One

> **In one line:** Two of the three meters bill by the token, and only one of them looks like it does.

This is where most people get the money wrong. It is the direct answer to the $200 Later story that opens this manual.

One: the plan allowance. A subscription buys access inside an application for a flat monthly fee. The vendor meters it as sessions, rolling windows, weekly caps, or some blend the product decides, and it resets on a schedule. You do not pay per token and you generally cannot see per-token accounting. When you exhaust it, you can wait for the reset, upgrade the plan, or buy usage credits (How do usage and length limits work?).

Two: usage credits. A prepaid balance that lets a paid plan keep working past its included limit, billed at standard per-token rates (Manage usage credits for paid Claude plans). This is the hybrid, and it is the one that surprises people. It lives inside the consumer app, so it feels like part of the subscription. It bills like the developer interface, so it behaves nothing like the subscription. Everything in Part Three about token accounting applies to your credits balance, even if you have never written a line of code.

Three: the application programming interface, usually shortened to API. Per-token billing from the first token, by category: new input, cached input, output, and sometimes reasoning, plus per-call charges for certain tools. No included allowance to exhaust.

The settings that govern meter two deserve a slow read, because their names are misleading in a specific way:

SETTING	WHAT PEOPLE ASSUME	WHAT IT DOES
Auto-reload	A convenience	The accelerator. With it off, spending stops at zero balance
Reload amount	A spending cap	The size of each automatic purchase
Monthly spend limit	A budget	The ceiling before someone has to intervene
Current balance	Money remaining	Money already spent, awaiting use

Only one of those four is a brake, and it is the one shaped like a convenience toggle.

Advice about one meter does not automatically transfer to another. Parts One and Two are mostly about the plan allowance. Part Three is mostly about per-token billing, which means it applies to meter two as much as meter three. Where a technique crosses over, it says so.

THE RULE

Know which meter is running before you optimize anything.

HOW THIS FAILS

Advice written for a flat monthly subscription gets applied to a per-token balance. The reader learns which meter they were on when the statement arrives.

CHAPTER 02 / 21 · GROUND

What Actually Fills the Window

In one line: Your typed question is a small part of what the model is actually reading.

What a token is

A model does not read words the way people do. It breaks text and other input into processing units called tokens. A token might be a whole word, part of a word, a punctuation mark, or another encoded unit. Images, audio, documents, tool descriptions, and system instructions you never see also consume capacity, even when the product reports their accounting differently.

For everyday work, exact counting matters less than knowing what is on the table. If you do need a real number rather than an estimate, providers expose token-counting endpoints for that purpose (Claude token counting). The working space holds:

- your current request
- earlier messages in the conversation
- uploaded files and images
- project instructions and project knowledge
- retrieved search results
- tool definitions and tool results
- the model's own answer
- internal reasoning, when the product exposes or meters it

Think of the context window as a worktable. A bigger worktable holds more material. Covering every inch of it with paper does not make the work better. The goal is to put the right evidence on the table at the right moment.

Four terms worth learning once

Input is everything the system receives, which is more than the sentence you just typed.

Output is what the model produces for you.

Context is the information available to the model for the current response.

Capacity is the practical amount of model access you have under your plan, your rate limits, your context window, and your tool allowances.

> THE RULE
>
> Give the model what it needs, not everything you have.

That does not mean prompts should be short. A complete 500-word instruction often beats a vague 20-word request that produces three failed drafts. The question is not how short you can make it. The question is what information raises the chance of a correct result.

> HOW THIS FAILS
>
> People optimize the visible part, meaning their own typed sentence, and ignore the invisible part, meaning the four uploaded files and the 200 turns of history sitting underneath it.

CHAPTER 03 / 21 · GROUND

The Attention Budget

> **In one line:** A larger context window is a ceiling, not a target, and the reason is measurable.

The symptom you have already seen

Anthropic's own help documentation notes that context window size depends on the model, with the newest models on paid plans supporting up to 1M tokens and others 500K or 200K (How do usage and length limits work?). A million-token window means the model can accept a very large working set. It does not mean any given task benefits from one. Long contexts dilute attention, hide conflicting instructions, raise latency, and make failures harder to diagnose. When a long conversation starts producing confidently wrong answers about things you told it forty turns ago, you are not imagining it.

Context rot

The measured version of this has a name. Chroma's technical report, Context Rot: How Increasing Input Tokens Impacts LLM Performance, evaluated 18 models including GPT-4.1, Claude 4, Gemini 2.5, and Qwen3, and found that models do not use their context uniformly. Reliability decreases as input length grows, and it does so even on simple retrieval and text-replication tasks (Context Rot).

Anthropic characterizes the effect as a performance gradient rather than a hard cliff. Models stay highly capable at long contexts but show reduced precision for retrieval and long-range reasoning compared with their performance on shorter ones (Effective context engineering for AI agents). The product documentation carries the same warning, stating plainly that more context is not automatically better (Claude context windows).

VERIFY BEFORE USE

Both sources are point-in-time measurements of specific model versions. The direction of the finding has held across every model family tested so far. The exact magnitude for the model in front of you today has not been measured by me.

This last section is the underlying reason. You can skip it and lose nothing practical.

Why this happens, if you want the mechanism

Anthropic describes context as a finite resource with diminishing returns, and describes models as having an attention budget that every additional token draws down. The reason is structural. These models are built on the transformer architecture, in which every token can attend to every other token, producing n squared pairwise relationships for n tokens. As the context gets longer, the model's ability to hold all those relationships gets stretched thinner. Models are also trained on distributions where short sequences are far more common than long ones, so they have less practice with long-range dependencies (Effective context engineering for AI agents).

THE RULE

Find the smallest set of high-signal material that gets the outcome you need. Capacity is a ceiling, not a target.

HOW THIS FAILS

Someone reads that a model has a million-token window, concludes that context management is solved, uploads the folder, and spends the next hour arguing with a very patient assistant about which version of the budget is current.

CHAPTER 04 / 21 · FRAME

Useful Work Is the Unit That Matters

> **In one line:** A cheap answer you cannot use costs more than an expensive answer you can.

Token minimization is a bad goal on its own. If a cheap call fails and has to be repeated four times, it costs more than one well-designed call to a stronger model, and that is before you count your own time.

Use this as the mental formula:

FORMULA

```
Efficiency = useful accepted work / AI capacity consumed
```

Accepted work means output that clears the real quality threshold. Not output that looks finished. Output you actually used.

The ten beginner rules

Seven that apply to almost any task:

1. Keep one conversation focused on one body of work.
2. State the outcome, the audience, the constraints, and the format you want.
3. Upload only files that can change the answer.
4. Ask for the amount of output you will actually use.
5. Use deeper reasoning only when the task warrants it.
6. Correct the defective instruction instead of asking for another full rewrite.
7. Verify factual output before you rely on it.

Three more that pay off once the work gets longer:

1. Store reusable project information in a project or notebook when the product supports one.
2. Start a new conversation when the subject, the audience, or the governing instructions materially change.
3. Before leaving a long conversation, capture decisions, current state, unresolved issues, and next actions.

THE RULE

Measure yourself on accepted work per unit of capacity, not on how little you typed.

HOW THIS FAILS

Efficiency gets measured by how short the prompt was, which is the one number that has almost nothing to do with whether the work got done.

CHAPTER 05 / 21 · FRAME

Choosing the Lightest Capable Path

> **In one line:** Match the tool to the difficulty before you start, not after the third retry.

Classify the task first

TASK	SENSIBLE STARTING POINT
Rewrite a paragraph	Fast or standard model, short answer
Summarize one known file	Standard model, file attached once
Compare several sources	Stronger model, explicit comparison criteria
Research a changing fact	Web-enabled research with citations
Make a high-stakes recommendation	Strong reasoning, primary sources, human review
Repeated work on one project	Project or notebook with curated knowledge

Do not spend frontier capability on routine formatting. Do not force a small model through work it keeps failing. Move up or down based on observed quality, not on which model sounds most serious.

Control the response length

Output consumes capacity too. Ask for what you will use:

PROMPT

```
Give me a five-bullet decision brief. Each bullet should be one sentence.
```

That beats asking for a detailed analysis when what you need is a meeting opener.

Avoid the rewrite loop

WEAK	Try again. No, rewrite it again. Still not right. Do the whole thing over.
BETTER	Keep the structure. The problem is the recommendation section: it is too generic and lacks a decision criterion. Replace only that section with three options ranked by cost, risk, and time.

Targeted correction preserves the work that was already good and removes the ambiguity that caused the failure.

What to do when you are near a limit

1. Stop exploratory prompting and name the deliverable.
2. Ask for a compact state summary if the conversation is long.
3. Move routine edits to a faster or lower-capacity model if one is available.

4. Remove files and background that cannot change the answer.
5. Request shorter outputs.
6. Finish one coherent task before opening another.
7. Wait for the plan's reset instead of buying capacity reflexively.
8. If you turn on usage credits to finish something, treat it as a decision with a number attached, not a setting. Check the auto-reload state, not just the spend limit.
9. Upgrade only when an efficient workflow still hits a real constraint.

Anthropic's own usage guidance says message length, attachment size, conversation length, tool use, model choice, and artifact activity all affect plan usage (Claude usage-limit best practices). The exact allowance varies by plan and demand, so treat the product's current usage indicator as the operational source of truth, not any number in this manual.

> THE RULE
>
> **Use the least intensive route that reliably clears the quality bar.**

> HOW THIS FAILS
>
> The strongest model becomes the default for everything, which is not a strategy. It is the absence of one.

CHAPTER 06 / 21 · GOVERN

Conversations, Projects, and Clean Restarts

> **In one line:** Continuity is an asset until it turns into interference.

Continue the current conversation when

- the goal is unchanged
- earlier decisions still govern the work
- the same files are still relevant
- the next request is a direct continuation
- the thread is still coherent

Start a new conversation when

- the goal or the audience changes
- old instructions conflict with the new task
- the transcript is mostly abandoned exploration
- the model keeps confusing earlier requirements with current ones
- you want a clean comparison or an independent review

- a milestone is finished and a compact handoff can carry the state forward

Hand off state, do not dump transcript

PROMPT

```
Create a continuation brief containing:
1. objective
2. approved requirements
3. decisions and rationale
4. source files and controlling versions
5. completed work
6. unresolved issues
7. next action
8. actions that require approval
Do not include brainstorming that was rejected.
```

State preservation matters more than transcript preservation. The transcript is the argument. The brief is the conclusion.

Projects are not magic memory

A project earns its keep when work shares standing instructions, reference files, and recurring goals. It is not a reason to upload everything you own. ChatGPT Projects group chats, files, and project instructions (ChatGPT Projects). Claude Projects provide a knowledge base and shift to retrieval automatically when project knowledge gets large (Claude Projects).

Use separate projects when the governing instructions, the confidentiality boundary, the audience, or the source library differs. One project per body of work, not one project per person.

THE RULE

Keep a conversation while it is still cheaper than rebuilding it, and not one turn longer.

HOW THIS FAILS

The immortal chat. It knows everything, which is precisely why it can no longer tell you which thing is current.

CHAPTER 07 / 21 · GROUND

Files, Sources, and Who Wins a Disagreement

> **In one line:** An unlabeled pile of files is not evidence. It is furniture.

Upload with a reason

For every file, finish this sentence:

> The model needs this file because it contains _____ that could change the answer.

If you cannot fill in the blank, do not upload it yet.

Tell the model how to use each source

PROMPT

```
Use policy.pdf as the controlling source for requirements.
Use proposal.docx as the draft to revise.
Use voice-samples.md only for tone.
Do not treat the archived notes as current policy.
```

Without source roles, a model treats signed policy, a stale draft, and somebody's brainstorm as equally valid. It has no way to know which one you would defend in a meeting.

Split, summarize, or keep whole

Split a file when only one section is relevant and it divides cleanly. Summarize when the decisions matter but the wording does not. Keep the full source when exact language, tables, citations, or cross-section relationships matter.

Retrieval versus stuffing

Retrieval means finding the relevant fragments at the moment of the request instead of loading an entire library into every prompt. It suits large or changing collections. It also adds a failure mode: the system can retrieve the wrong passage or miss the right one entirely. Whenever the answer depends on retrieved evidence, ask for citations or source locations so you can check.

Point at things instead of pasting them

This is the consumer version of a technique that has become standard in agent design. Rather than pre-loading everything, keep lightweight identifiers such as file paths, links, or saved queries, and load the actual content only when the task calls for it. Anthropic describes this as a just-in-time approach and notes that the metadata of the reference itself carries signal: a file named `test_utils.py` in a `tests` folder implies something different from the same filename in `src/core_logic/`. Folder hierarchies, naming conventions, and timestamps help both people and models decide when something is worth opening (Effective context engineering for AI agents).

You already do this. You do not memorize your filing cabinet. You remember where the drawer is.

A worked example

WEAK	`Read these six files and tell me what you think.`

BETTER	`Review proposal.docx for the executive sponsor.`
	`Use policy.pdf as the controlling requirement source.`
	`Use pricing.xlsx only to verify cost claims.`
	`Identify the five defects most likely to block approval.`
	`For each, cite the source location and propose the smallest correction.`
	`Do not rewrite the proposal.`

The difference is not length. It is that the second version tells the model which file governs, which file is being changed, what counts as a defect, and what not to do.

THE RULE

Name the controlling source before you name the task.

HOW THIS FAILS

The everything upload. A whole folder, just in case, complete with three versions of the same budget and no indication of which one survived the meeting.

CHAPTER 08 / 21 · VERIFY

Myths That Cost You Capacity

> **In one line:** Most expensive AI habits come from a belief that sounded reasonable.

Myth: Shorter prompts are always better. Complete prompts reduce retries. Remove irrelevant detail, not necessary detail. Anthropic makes the same point about system prompts: minimal does not mean short, it means the smallest set of information that fully describes the expected behavior (Effective context engineering for AI agents).

Myth: The biggest context window is the best model. Context size is one capability among quality, latency, tools, cost, and reliability. See Chapter 3.

Myth: Always use the smartest model. Use the least intensive model that reliably clears the quality threshold.

Myth: Maximum reasoning means maximum quality. More reasoning adds cost and delay without improving simple work, and it encourages over-analysis when the success criteria are weak.

Myth: Never start a new chat. Continuity is useful right up until stale context becomes interference.

Myth: Starting a new chat always saves capacity. Rebuilding all the relevant context can cost more than continuing a clean conversation. Both directions have a wrong answer.

Myth: Projects remember everything perfectly. Project behavior depends on the product, the plan, the settings, the retrieval, and the quality of what you put in. A project is a shelf. It is not a memory.

Myth: A big context window means the model read all of it carefully. It accepted all of it. Recall degrades as input grows, measurably (Context Rot).

Myth: Free plans cannot do serious work. Bounded tasks with good instructions produce real value on free tiers. Paid tiers mainly buy capacity, tools, speed, and access. Identify the constraint before you buy the cure.

Myth: If the answer is wrong, ask for a complete rewrite. Diagnose the failure instead. It is usually a missing source, an unclear criterion, the wrong audience, or a defective output contract.

THE RULE

When a habit feels obviously right, check whether anyone has measured it.

HOW THIS FAILS

Advice that was true about one product in one month gets carried into a different product a year later, and nobody notices because it still sounds sensible.

If you stop here

That is the foundation. Three things to do tomorrow:

1. Before your next big task, write the five-line inventory from Chapter 10: goal, controlling sources, constraints, known decisions, required output.
2. Delete one file from a project that cannot change any answer.
3. The next time a result is wrong, name the failure before you retype the prompt.

Part Two is for when the same work comes back a second time. Part Three is for when you are billed per token, which includes usage credits. Neither is a prerequisite for the other.

THE DISCIPLINE · 6 CHAPTERS · ABOUT 9 MINUTES

Part Two: The Discipline

CHAPTER 09 / 21 · GROUND

The Context Engineering Stack

> **In one line:** Five layers, each with a different lifespan.

Context engineering is the deliberate design of what the model sees while doing a task. Anthropic frames it as the natural progression of prompt engineering: prompt engineering asks what words to use, context engineering asks what configuration of context is most likely to produce the behavior you want (Effective context engineering for AI agents).

The stack:

DIAGRAM

Persistent context. Rules that apply broadly: communication preferences, safety boundaries, default units, organizational policy, stable role expectations. Keep this layer short. A permanent instruction that matters twice a year should not tax every task you run.

Project context. What one body of work shares: objectives, audience, glossary, controlling sources, voice examples, decision rights, current constraints.

Retrieved context. Evidence selected from a larger collection at the moment it becomes relevant. Narrow enough to focus the model. Traceable enough to verify.

Task context. The current assignment: what changed, what must be produced, which inputs apply, what is out of scope, what needs approval.

Output contract. The acceptance conditions: format, length, structure, audience, factual standard, citation method, definition of done.

The prompt you actually type should usually be the delta, meaning the new information this task requires. Stable material belongs in stable layers. If you are retyping your job title every morning, one of these layers is empty.

The right altitude

Anthropic describes a Goldilocks zone for instructions. At one extreme, engineers hardcode brittle if-else logic to force exact behavior, which becomes fragile and expensive to maintain. At the other, they give vague guidance that assumes shared context the model does not have. The useful altitude is specific enough to guide behavior and flexible enough to leave the model good heuristics. Their advice on getting there: start with a minimal instruction on the best model available, then add instructions and examples based on the failure modes you actually observe, rather than preemptively (Effective context engineering for AI agents).

That last part is the discipline. Most bloated instructions are a scar from a failure that happened once.

One task, all five layers

A district analyst has to turn a quarterly enrollment file into a board memo. Here is where each piece goes.

LAYER	WHAT LANDS HERE	WHY
Persistent	Write for a non-technical board. Never include student-level data.	True of every memo this person will ever write
Project	The board's memo template, last quarter's approved memo, the definition of "enrolled" the district uses	True for this body of work, not for their other work
Retrieved	The two prior quarters, pulled only when a trend claim needs support	Large, occasionally relevant, wrong to paste in full every time
Task	This quarter's file, the three questions the board asked last meeting, the fact that one school consolidated in March	New this time and only this time
Contract	600 words, recommendation first, every number traceable to a row, flag anything the file cannot answer	What finished looks like

The prompt they type is the Task row and the Contract row. Everything above those was decided once. If they are retyping the definition of "enrolled" every quarter, it is in the wrong layer.

THE RULE

Put every piece of context in the layer that matches its lifespan.

HOW THIS FAILS

Everything lands in the prompt, so everything has the lifespan of one message and gets retyped, slightly differently, forever.

CHAPTER 10 / 21 · GROUND

Context Budgeting

> **In one line:** Decide what goes in by deciding what it is for.

Classify before including

CONTEXT CLASS	TREATMENT
Required now	Include directly
Reusable across tasks	Store in persistent or project context
Occasionally relevant	Retrieve when needed
Historical but important	Summarize as state
Superseded	Label clearly or remove
Irrelevant	Exclude

A practical context inventory

Before a large task, write five lines:

PROMPT

```
Goal:
Controlling sources:
Constraints:
Known decisions:
Required output:
```

Then list the candidate files and remove anything that cannot change the result. This takes two minutes and routinely removes half the pile.

Examples are compressed specifications

One good example communicates tone and structure more efficiently than three paragraphs of description. Anthropic advises curating a diverse set of canonical examples rather than stuffing in a laundry list of edge cases, and describes examples as the pictures worth a thousand words for a model (Effective context engineering for AI agents).

Use examples to establish a pattern. Do not use them to smuggle last quarter's facts into this quarter's work.

Context decay

Information ages at different rates. Brand voice may hold for years. Model names, plan limits, prices, and regulations may not hold for a quarter. Put a verification date next to volatile material and set a review interval proportional to the risk.

THE RULE

Every item in context should have a reason and a review date.

HOW THIS FAILS

The project knowledge base becomes an archive. Nothing is ever removed, because removing something requires knowing whether it still matters, and nobody wrote that down.

CHAPTER 11 / 21 · VERIFY

The Six Ways Context Fails

> **In one line:** Most bad answers are context failures, not knowledge failures, and they fail in recognizable ways.

These six names are mine. The underlying behaviors are not controversial, but do not go looking for this exact taxonomy in a vendor document.

1. Starvation. The model lacked something it needed. Symptom: a confident, generic answer that would fit any organization. Fix: add the specific missing source.

2. Dilution. Too much material, so the signal thinned out. Symptom: it gets details wrong that are demonstrably present in the context. This is the measured effect from Chapter 3 (Context Rot). Fix: remove, summarize, or retrieve instead of stuffing.

3. Clash. Two sources disagree and nothing said which one controls. Symptom: the answer splits the difference, or picks one silently. Fix: state authority explicitly.

PROMPT

```
If the sources conflict, use the signed policy as controlling.
Treat the slide deck as explanatory, not authoritative.
List unresolved conflicts instead of choosing silently.
```

4. Contamination. A wrong fact entered early, was never corrected, and is now being reused as established. Symptom: an error that keeps reappearing after you fixed it. Fix: correct it at the source and restart the thread. Correcting it in turn 40 does not remove it from turns 1 through 39.

5. Drift. The context was accurate when it was written and no longer is. Symptom: last year's price, last reorganization's approval chain. Fix: verification dates, per Chapter 10.

6. Overload of options. Too many tools, connectors, or capabilities with overlapping purposes. Symptom: the model picks a plausible wrong tool. Anthropic names bloated tool sets as one of the most common failure modes and offers a clean test: if a human engineer cannot definitively say which tool should be used in a given situation, an agent cannot be expected to do better (Effective context engineering for AI agents). Fix: turn things off.

The diagnostic order

Work down the list. Starvation and clash account for most of what people blame on the model.

THE RULE

Name the failure before you change anything, because the fixes are opposites. Starvation wants more context. Dilution wants less.

HOW THIS FAILS

Every bad answer gets treated as dilution, so people keep cutting context until the model is starving, and then conclude that AI does not work on real problems.

CHAPTER 12 / 21 · GOVERN

Conversation Architecture and State

> **In one line:** Long work needs a shape, not just a longer thread.

Work in phases

A strong long-running workflow separates:

1. discovery
2. requirements
3. plan
4. execution
5. verification
6. handoff

At each milestone, compress. Preserve decisions and evidence. Discard the ideas you rejected, which is most of what a long transcript actually contains.

The continuation brief

- objective and definition of done
- controlling instructions and sources

- decisions already approved
- completed work and verification status
- unresolved questions
- risks and assumptions
- the exact next step
- actions that require authorization

Branch for independent thinking

Use a separate conversation for a critique, a competing strategy, or a fresh review. A reviewer that watched the entire original debate inherits the original assumptions. Give it the deliverable, the acceptance criteria, and the sources, then ask it to find defects. Do not give it the argument you already had.

Durable state belongs outside the chat

For consequential work, the current specification, decision log, checklist, or source file lives in a repository, a document, or a task record. The conversation holds the working state. It should not be the only place a project's controlling decisions exist.

This is the same boundary that agent designers now build deliberately. A conversation is working memory. A file is the record.

THE RULE

The chat holds the current state. Something durable holds the decisions.

HOW THIS FAILS

The project's only record of why a decision was made is turn 74 of a thread that got archived when the laptop was replaced.

CHAPTER 13 / 21 · FRAME

Output Contracts

> **In one line:** Many expensive failures happen because nobody defined finished.

A complete output contract

PROMPT

```
Deliverable: one decision memo
Audience: executive team
Length: 700 to 900 words
Structure: recommendation, evidence, risks, next action
Sources: use only the attached policy and data table
Citations: cite page or row near every factual claim
Quality bar: recommendation must identify owner, deadline, and tradeoff
Boundary: assess and report; do not send, publish, or modify systems
```

The boundary line matters more as tools get more capable. Say what the model may do, not only what it should produce.

Ask for uncertainty

Require the model to distinguish:

- verified fact
- reasonable inference
- assumption
- missing information

This is the cheapest quality control available. It converts a polished guess into a labeled guess, which is a different thing entirely.

Contract failures look like knowledge failures

When output is wrong in a way that feels stubborn, check the contract before you blame the model. A recommendation with no owner and no deadline is not a model that failed to reason. It is a specification that never asked.

THE RULE

Define done in the request, not in the third round of feedback.

HOW THIS FAILS

The acceptance criteria live in your head, get discovered one at a time during review, and each discovery costs another full generation.

CHAPTER 14 / 21 · GROUND

Platform Playbooks

> **In one line:** The context problem is the same everywhere. The controls are not.

The side-by-side version of this chapter is the Platform Optimization Cheat Sheet in Reference. Read this for the reasoning, use the table for the comparison.

> VERIFY BEFORE USE
>
> Everything in this chapter is product behavior as of September 4, 2026, and product behavior changes. The reasoning is durable. The specifics need rechecking.

ChatGPT

Use a regular chat for a bounded task. Use a Project when chats, files, and instructions belong to one continuing body of work (ChatGPT Projects). Keep project instructions stable and specific. Open a fresh chat inside the project for each new deliverable rather than stretching one thread across unrelated outputs.

Use stronger reasoning for decisions, hard synthesis, and ambiguous constraints. Use standard modes for drafting, summarizing, and routine transformation. When you need current facts, use web research and insist on links near the claims.

Claude

Claude is well suited to long-form writing, close reading, and sustained work over organized source material. Projects are valuable for content reused across conversations. Anthropic states that reused project content can benefit from caching, and that large project knowledge on paid plans can shift to retrieval (Claude usage-limit best practices, Claude Projects).

Do not keep one Claude conversation forever merely because it has history. Capture state at milestones. Start fresh when old exploration begins to compete with current instructions.

Gemini

Gemini is useful when very large files or genuinely multimodal material belong in one analysis. A large context window is capacity, not an invitation. Use saved instructions or Gems for reusable behavior, and keep current-task directions in the current request.

Google notes that more advanced models and higher thinking levels consume more of a user's allowance. Start at standard thinking for ordinary work and move up when the task needs it (Gemini app limits).

Microsoft Copilot Free

Appropriate for web-grounded questions, brainstorming, rewriting, summarizing pages, and general assistance. Signing in adds history and other capabilities. It does not turn general web chat into a curated business workspace. Microsoft describes the free experience as web-grounded chat at no cost, with additional features when signed in (Microsoft Copilot experiences).

Keep requests bounded. If the task depends on private organizational material, use the approved work environment and confirm what grounding and protections actually apply.

Microsoft 365 Copilot Chat

Copilot Chat changes the context problem, because account type, licensing, organizational configuration, and data access all affect what it can see. Name the source boundary you want: the web, a specific file, or work content available to you. Never assume it searched every email, site, and file. Microsoft states that Copilot Chat is included with eligible Microsoft 365 licenses and is web-grounded, and that broader work-grounded behavior depends on licensing and configuration (Microsoft Copilot experiences).

Microsoft 365 Copilot Notebooks

Use a Notebook when a task should be grounded in a curated set of references. Add only the files, pages, chats, and notes that define the assignment. Microsoft states that Copilot Notebooks uses the references added to the notebook rather than automatically reaching the user's entire OneDrive, email, Teams history, or the web (How Copilot Notebooks works).

That makes a Notebook a clean example of context engineering. The user defines the evidence boundary on purpose.

Comparative truth

These products are not the same. Plan allowances, project behavior, retrieval, reasoning controls, file limits, and data boundaries all differ. Make direct comparisons only after checking current official documentation. Durable advice belongs to context roles and acceptance criteria, not to a permanent winner.

THE RULE

Choose the grounding boundary before you choose the prompt.

HOW THIS FAILS

A technique that worked in one product gets carried to another where the same words mean something different, and the failure is silent.

SYSTEMS · 7 CHAPTERS · ABOUT 12 MINUTES

Part Three: Systems

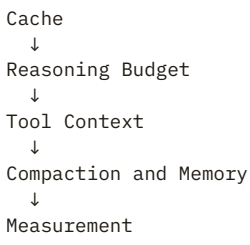
This part is mostly about the programming interface, where you pay per token and can see the accounting. Some of it applies to subscription work, and those places are marked.

CHAPTER 15 / 21 · FRAME

The Advanced Controls

> **In one line:** Five levers turn prompt craft into workload architecture.

DIAGRAM



Cache reuses stable input instead of paying to reprocess it. **Reasoning budget** matches thinking effort to difficulty. **Tool context** exposes only the capabilities and results that help. **Compaction and memory** preserve state while shedding transcript weight. **Measurement** tells you whether any of it worked.

The first four are things you build. The fifth is the one that keeps you honest, and it is the one most often skipped.

THE RULE

Do not adopt a lever you cannot measure.

HOW THIS FAILS

A team implements caching, reports a cost reduction, and never notices that the retry rate went up enough to erase it.

CHAPTER 16 / 21 · GOVERN

The Economics of an AI Request

> **In one line:** Cost per call is the wrong denominator. Cost per accepted task is the right one.

General cost formula

For one request:

FORMULA

```
Request cost
= (new input tokens × new input rate)
+ (cache-write tokens × cache-write rate)
+ (cache-read tokens × cache-read rate)
+ (output tokens × output rate)
+ tool or retrieval charges
+ storage or cache-duration charges
```

Some vendors fold reasoning tokens into output billing. Some expose them in a separate usage field. Some tools carry per-call charges. Map the provider's current invoice categories before you use this formula, because the categories are where the surprises live.

For a workload:

FORMULA

```
Workload cost
= successful request costs
+ failed request costs
+ retry costs
+ validation costs
+ fallback costs
+ human correction cost
```

That last line is usually the largest and is almost never in the vendor dashboard.

Worked example: architecture beats brevity

These rates are hypothetical, chosen for arithmetic. They are not current vendor pricing and should not be used for budgeting.

- new input: $2 per million tokens
- cached input: $0.20 per million tokens
- output: $8 per million tokens
- each task uses 40,000 repeated input tokens, 2,000 new input tokens, and 3,000 output tokens
- 100 tasks run

Without caching:

FORMULA

```
Input = 100 × 42,000 = 4,200,000 tokens
Input cost = 4.2 × $2 = $8.40
Output = 100 × 3,000 = 300,000 tokens
Output cost = 0.3 × $8 = $2.40
Total = $10.80
```

Ignoring cache-write charges for this simplified illustration, with the 40,000-token prefix cached after the first request:

FORMULA

```
New input = 100 × 2,000 + 40,000 = 240,000 tokens
New input cost = 0.24 × $2 = $0.48
Cache reads = 99 × 40,000 = 3,960,000 tokens
Cache-read cost = 3.96 × $0.20 = $0.792
Output cost = $2.40
Approximate total = $3.672 plus cache-write charges
```

The architecture created the saving. A shorter user question would not have come close.

Worked example: the cheaper call that costs more

Workflow A costs $0.04 per attempt and succeeds 55 percent of the time.

FORMULA

```
Expected model cost per success = $0.04 / 0.55 = $0.073
```

Workflow B costs $0.07 per attempt and succeeds 92 percent of the time.

FORMULA

```
Expected model cost per success = $0.07 / 0.92 = $0.076
```

On model cost alone they are nearly identical. Now add a person. If each failed attempt needs five minutes of review at $60 an hour, Workflow A adds $5 in labor per hour of running, and Workflow B adds well under half that. Cost per call would have hidden the entire decision.

THE RULE

Divide by accepted tasks, and count the human minutes.

> HOW THIS FAILS
>
> Procurement compares per-million-token rates across vendors, picks the cheapest, and discovers the retry rate six weeks later.

CHAPTER 17 / 21 · GOVERN

Prompt Caching

> **In one line:** Put the stable material first, and then verify that the cache is actually being hit.

The stable-prefix pattern

PROMPT

```
STABLE PREFIX
System instructions
Policies
Schemas
Tool definitions
Large repeated references
Examples

DYNAMIC SUFFIX
Current record
Current question
Time-sensitive data
Requested output
```

Caching reduces the price or the latency of reprocessing eligible repeated input. It does not remove that input from the model's context window. A cached 40,000-token prefix still occupies 40,000 tokens of attention budget, with everything Chapter 3 says about that.

That distinction is worth repeating because it is the most common misunderstanding in this whole chapter. Caching is a billing optimization. It is not a context optimization.

The vendors all have it, and they are all different

OpenAI, Anthropic, and Google each document prompt or context caching, but the controls, eligibility thresholds, retention, usage fields, and prices differ (OpenAI prompt caching, Claude prompt caching, Gemini context caching).

OpenAI's documentation describes caching as reusing work when requests share a prompt prefix, with reads billed at a reduced cached-input rate, and notes that cache reuse requires the entire rendered prefix to match (OpenAI prompt caching). VERIFY CURRENT Discount levels, minimum sizes, and retention windows are exactly the parameters most likely to have changed since this was checked.

Google documents implicit caching for current Gemini families and recommends structuring repeated requests with a similar prefix, keeping large common content at the front (Gemini context caching). The design advice converges across all three vendors even where the billing does not: common content first, variable content last.

Cache design rules

1. Put the most stable content first.
2. Put timestamps, user records, and current questions last.
3. Avoid tiny changes to the prefix. A single edited word at the top can invalidate everything after it.
4. Measure actual cache reads. Do not assume a hit.
5. Compare cache-write and storage costs against expected reuse before committing.
6. Do not cache sensitive material without reviewing retention and data controls.
7. Remember that a cached bad instruction is still a bad instruction, delivered faster and cheaper.

Cache break-even

Let:

- `W` = one-time cache-write cost
- `R` = cost of one cache read
- `U` = cost of sending the same tokens uncached
- `N` = number of later reuses

Caching is economically favorable when:

```
PROMPT

W + (N × R) < N × U
```

Therefore:

```
PROMPT

N > W / (U - R)
```

Use current vendor rates and retention rules. Do not copy a break-even count from a different provider, or from this manual.

> THE RULE
>
> **Stable first, dynamic last, then check the usage field.**

> HOW THIS FAILS
>
> The cache illusion. Everyone believes the prefix is cached. Nobody has looked at the cached-token count, and a timestamp at the top of the system prompt has been quietly invalidating it since launch.

CHAPTER 18 / 21 · GROUND

Retrieval, Tools, and Structured Output

> **In one line:** Every tool you expose is context you are spending.

Retrieval economics

Retrieval replaces a large repeated corpus with a small set of relevant passages. Its cost includes indexing, storage, search, the retrieved tokens themselves, and the risk of missing evidence.

Measure it on two dimensions:

- **precision:** how much of what was retrieved was actually relevant
- **recall:** how much of the necessary evidence was retrieved

High precision with poor recall gives focused, incomplete answers. High recall with poor precision recreates context stuffing with extra steps.

Just-in-time versus pre-loaded

The field has shifted. Many applications still do embedding-based retrieval before inference. Increasingly, systems augment that with just-in-time strategies, where the agent holds lightweight references and pulls data at runtime through tools. Anthropic gives the example of Claude Code writing targeted queries and using commands like `head` and `tail` to work across large data without ever loading the full objects into context (Effective context engineering for AI agents).

The tradeoff is real. Runtime exploration is slower than retrieving something precomputed, and it takes deliberate engineering to give the model the right tools and heuristics for navigating. Without that, an agent burns context chasing dead ends. Anthropic notes that the most effective systems are often hybrids: retrieve some things up front for speed, allow autonomous exploration for the rest, and that a hybrid suits less dynamic domains such as legal or financial work (Effective context engineering for AI agents).

Tool context

Every tool definition consumes space. Expose only tools the task might need. Write descriptions that are clear about purpose, inputs, side effects, and error conditions. Anthropic's guidance is that tools should be self-contained, robust to error, and unambiguous about intended use, and that bloated, overlapping tool sets are among the most common failure modes (Effective context engineering for AI agents).

This applies to consumer work too. If you have fourteen connectors switched on and you are asking about a spreadsheet, thirteen of them are overhead and at least one is a distraction.

Limit large tool outputs before they reach the model. Filter fields, narrow date ranges, cap rows.

Structured outputs

Use a schema when downstream software needs predictable fields. Structured output reduces parsing failures and retries. An oversized schema also consumes context and can make simple work harder. Define only required fields, name them clearly, and validate programmatically rather than by reading.

Batch and asynchronous work

When immediate interaction is not required, batch or lower-priority processing may cost less. The tradeoff is price against latency, operational complexity, and retry handling. Verify provider terms before designing around it.

THE RULE

A tool the model cannot confidently choose between is a tool you should turn off.

HOW THIS FAILS

The connector shelf. Everything is enabled because enabling was easy, and now the model has three plausible ways to look up a customer and picks a different one each time.

CHAPTER 19 / 21 · FRAME

Reasoning Budgets and Model Routing

> **In one line:** More thinking is a cost, not a quality setting.

Reasoning is a budget

Use low or standard reasoning for extraction, classification, formatting, and routine summaries. Increase it for ambiguous decisions, multi-source reconciliation, difficult mathematics, architecture, and high-risk review.

OpenAI's guidance says higher reasoning effort is not automatically better and recommends raising it only when evaluation shows a measurable quality gain (OpenAI model guidance). Google similarly warns that more advanced models and higher thinking levels consume more usage (Gemini app limits).

Routing rule

PROMPT

```
Choose the least expensive and least intensive route
that reliably passes the task's quality threshold.
```

Build a small evaluation set of representative tasks. Test candidates on accuracy, completeness, latency, token use, and retry rate. Route by task class, not by preference. Twenty labeled examples will settle arguments that have been running for months.

Escalation pattern

1. Start with the proven default for that task class.
2. Validate the output.
3. Escalate only on a defined failure.
4. Pass the failure evidence and the relevant state, not the entire history.
5. Record whether escalation actually improved the result.

Step five is the one everyone skips, and it is the only one that tells you whether the escalation rule is worth keeping.

THE RULE

Escalate on evidence, and carry the evidence, not the transcript.

HOW THIS FAILS

The maximum-everything default. Biggest model, deepest reasoning, every tool on, longest output. That is not routing. It is the absence of routing with a larger invoice.

CHAPTER 20 / 21 · GOVERN

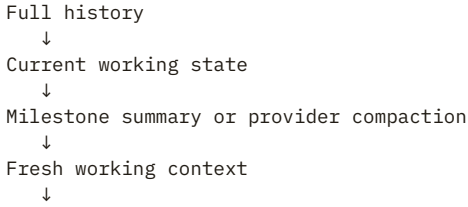
Long-Horizon Work

> **In one line:** When the work outlasts the window, you need a strategy for what survives.

Long-horizon tasks are the ones where total token count exceeds the context window: a large migration, a multi-week research project, anything measured in hours of continuous work. Waiting for bigger windows does not solve it, because windows of every size remain subject to the attention problem in Chapter 3. Anthropic describes three techniques that address this directly: compaction, structured note-taking, and sub-agent architectures (Effective context engineering for AI agents).

Compaction

DIAGRAM



Compaction takes a conversation approaching the window limit, summarizes it, and restarts with the summary. In Claude Code, Anthropic implements this by passing the message history back to the model to compress, preserving architectural decisions, unresolved bugs, and implementation details while discarding redundant tool outputs, then continuing with that compressed context plus the most recently accessed files (Effective context engineering for AI agents).

The art is in what you keep. Over-aggressive compaction loses the subtle detail whose importance only becomes obvious later. Anthropic's tuning advice is to maximize recall first, making sure the compaction prompt captures everything relevant, then improve precision by cutting the superfluous (Effective context engineering for AI agents).

Both major platforms now expose this. OpenAI documents server-side compaction on the Responses API, enabled by setting a compaction threshold, plus a standalone compaction endpoint (OpenAI compaction). Anthropic documents server-side compaction that summarizes the conversation as it approaches the window limit (Claude compaction).

Two design points are worth knowing before you build on either. First, how you carry conversation forward changes what you are allowed to prune: OpenAI's guidance is that with response-identifier chaining you should not prune manually, while with stateless input chaining you can drop items that came before the most recent compaction item (OpenAI conversation state, OpenAI context management). Second, compaction is not the same as truncation. Truncation drops the oldest material. Compaction tries to carry its meaning forward. Only one of those preserves the decision you made in turn three.

Context editing, or clearing what is already spent

The lightest-touch form of compaction is clearing old tool results. Once a tool has been called deep in the history, the raw result is rarely needed again. Anthropic's context editing feature clears specific tool results on the client, distinct from compaction, which summarizes the whole conversation server-side (Context editing).

Anthropic reports measured results on an internal agentic search evaluation: the memory tool combined with context editing improved performance 39 percent over baseline, and context editing alone improved it 29 percent. In a 100-turn web search evaluation, context editing let agents finish workflows that would otherwise have failed from context exhaustion, while reducing token consumption 84 percent (Managing context on the Claude Developer Platform).

VERIFY BEFORE USE

Those are vendor-reported numbers from an internal evaluation set, not an independent benchmark, and they describe a specific model and configuration. Treat them as evidence that the technique matters, not as a number you will reproduce.

Structured note-taking

The agent writes notes to persistent storage outside the context window and reads them back later. Anthropic describes this as agentic memory and gives the plain version: a to-do list, or a NOTES.md file that the agent maintains as it goes. The pattern lets an agent track progress across a long task and keep dependencies that would otherwise vanish across dozens of tool calls (Effective context engineering for AI agents).

Anthropic's memory tool implements this as a file directory the model can create, read, update, and delete, persisting between sessions. Their documentation recommends pairing memory with compaction for long-running agents: compaction keeps the active context small, and memory preserves the information that has to survive the summary. They also recommend setting up memory files deliberately for projects that span sessions, rather than writing them ad hoc as work proceeds (Memory tool).

That last recommendation is the one that transfers to everyone, including people who will never touch an API. A file you designed on purpose survives. A file the assistant improvised at turn 60 does not.

Sub-agents

Rather than one agent holding state for a whole project, specialized sub-agents handle focused tasks with clean context windows. The coordinating agent holds the plan. Each sub-agent may explore extensively, using tens of thousands of tokens, and returns only a condensed summary, often 1,000 to 2,000 tokens (Effective context engineering for AI agents).

The detailed search context stays isolated in the sub-agent. The lead agent gets the conclusion. This is the same reason you send someone to read the whole report and come back with a page.

Choosing among the three

Anthropic's own guidance on when to use which:

TECHNIQUE	BEST FOR
Compaction	Tasks with extensive back-and-forth that need conversational continuity
Structured note-taking	Iterative development with clear milestones
Sub-agents	Complex research and analysis where parallel exploration pays off

Source: Effective context engineering for AI agents.

The subscription version of all of this

You do not need an API to use these ideas.

- Compaction is the continuation brief in Chapter 12.
- Structured note-taking is a decision log in a document you own.
- Sub-agents are a separate chat given only the deliverable and the acceptance criteria, asked to find defects.

The mechanisms differ. The discipline is identical.

Where this manual stops

Once AI begins operating tools, repositories, terminals, applications, and other agents, context optimization becomes part of agent architecture. Permissions, tool selection, state recovery, verification, and rollback deserve their own treatment. That is a different manual, and this one stops at the boundary.

THE RULE

Decide in advance what survives the summary, and write it somewhere the summary cannot reach.

HOW THIS FAILS

The summary is generated automatically, looks reasonable, and quietly drops the one constraint that made the whole plan legal.

CHAPTER 21 / 21 · VERIFY

Measurement

> **In one line:** If you cannot say what improved, you did not optimize. You rearranged.

The core metrics

FORMULA

```
Tokens per successful task
= total workload tokens / accepted tasks

Cost per successful task
= total workload cost / accepted tasks

First-pass success rate
= tasks accepted without retry / total tasks

Retry rate
= repeated attempts caused by defects / total attempts
```

Cache effectiveness

FORMULA

```
Cache hit rate
= requests with an eligible cache hit / eligible requests

Cached-token share
= cache-read tokens / total eligible input tokens
```

Both come from provider usage fields, not from your assumptions.

Context relevance ratio

This one is a sampled audit, not a billing field:

FORMULA

```
Context relevance ratio
= input tokens judged necessary or materially useful
  / total reviewed input tokens
```

Do not pretend this number is objective. Sample both failed and successful tasks. Label each piece of context as required, helpful, neutral, distracting, or conflicting. The labels are more useful than the ratio.

Quality-adjusted efficiency

FORMULA

```
Quality-adjusted efficiency
= accepted quality score × successful tasks
  / total cost
```

Define the quality rubric before you compare systems. Otherwise the metric improves every time someone lowers the bar, which is a thing that happens.

Minimum measurement set

Track, per task:

- task type
- model or mode
- input, cached input, reasoning, and output usage where available
- total cost
- latency
- accepted or rejected
- retry reason
- human correction minutes
- source or retrieval failure

Nine fields. A spreadsheet handles it. The reason to write them down is that memory is generous about which experiments worked.

THE RULE

Baseline first, change one thing, measure the same way.

HOW THIS FAILS

The optimization is declared successful in the same meeting where it was proposed, on the strength of a single impressive-looking example.

What Is Durable and What Is Not

Three things in this manual should outlive the products described in it.

Context is a finite resource with diminishing returns. That follows from how the models are built, not from any vendor's roadmap.

Cost per accepted task is the only denominator that reflects the actual work. Everything else is a proxy.

The failure is usually in the framing, the grounding, or the verification, and only rarely in the model.

Everything else here, meaning the plan names, the caching parameters, the feature names, and the numbers, is a snapshot. It was accurate on September 4, 2026. Check the links before you spend anything on the strength of it.

14 SECTIONS · LOOKUP AND PRACTICE

Reference

REFERENCE

The One-Page Card

Before every important request

FORMULA

```
Outcome
+ necessary context
+ source roles
+ constraints
+ output contract
+ verification method
```

When a result fails, in diagnostic order

The first six are the failure modes from Chapter 11, in the order worth checking them.

PROMPT

```
1. Starvation      Was necessary context missing?
2. Clash           Did two sources disagree with no stated authority?
3. Frame           Was the goal or the definition of done unclear?
4. Dilution        Was there so much context that the signal thinned?
5. Contamination   Did a wrong fact enter earlier and get reused?
6. Drift           Was the context accurate once and not now?
7. Routing         Was the wrong model or reasoning level used?
8. Retrieval       Did the search miss the evidence?
9. Tools           Did the model lack a tool, or have too many?
10. Currency       Did I verify the provider's current behavior?
```

The central rule

> Context capacity is a ceiling, not a target. Spend enough capacity to produce accepted work, then remove what does not improve the outcome.

PRACTICE

Anti-Patterns

> **In one line:** Nine habits, each of which felt efficient at the time.

The everything upload. A whole folder, just in case. Introduces duplication, stale versions, and conflicting authority in one action. BEGINNER

The immortal chat. Continuing forever because the thread knows everything, until nothing in it can be located. BEGINNER

The maximum-everything default. Biggest model, deepest reasoning, all tools, longest output. See Chapter 19. BEGINNER

The invisible source hierarchy. Current policy, an old draft, and informal notes handed over together with no indication of which one controls. BEGINNER

The full-rewrite correction. Throwing away good work because one section failed. BEGINNER

The upgrade-before-diagnosis habit. Buying a higher tier before fixing file sprawl, prompt ambiguity, repeated output, or model selection. The tier is the cheapest thing to change and the least likely to be the problem. BEGINNER

The cache illusion. Assuming repeated content is cached without checking usage fields, prefix stability, minimums, and retention. ADVANCED

The price-only model comparison. Ignoring retry rate, latency, human correction time, and quality. ADVANCED

The connector shelf. Every integration enabled because enabling was easy, leaving the model an ambiguous action space. See Chapter 18. INTERMEDIATE

PRACTICE

Recipes

> **In one line:** Seven patterns you can copy.

Recipe 1: Review a proposal BEGINNER

The full worked example lives in Chapter 7, where source roles are introduced. See Chapter 7.

Recipe 2: Continue a long project BEGINNER

PROMPT

```
Before we continue, produce a state brief with the objective,
approved requirements, controlling sources, completed work,
verification evidence, unresolved issues, and next action.
Exclude rejected brainstorming. Then wait.
```

The final instruction matters. Without it you get the brief and an unprompted attempt at the next three steps.

Recipe 3: Get an independent review INTERMEDIATE

Open a new conversation. Give it three things and nothing else:

PROMPT

```
Here is the deliverable.
Here are the acceptance criteria.
Here are the controlling sources.
Find the defects most likely to cause this to be rejected.
Do not rewrite it. List defects with severity and location.
```

Do not include the original discussion. A reviewer who watched you reason your way to the answer will agree with the answer.

Recipe 4: Build a curated notebook or project BEGINNER

1. Define one outcome for it.
2. Add only current, relevant references.
3. Remove or label obsolete versions.
4. Write instructions for audience, format, and source priority.
5. Ask a test question whose answer you already know.
6. Check whether the cited source boundary is correct.
7. Expand only when a real evidence gap appears.

Step five is the whole recipe. Everything else is setup.

Recipe 5: Reduce API repetition ADVANCED

1. Move stable instructions, schemas, tools, and large references into a common prefix.
2. Put the current record and question at the end.
3. Keep the prefix stable according to the provider's matching rules.
4. Inspect the usage fields for cache writes and cache reads.

5. Compare total workload cost before and after, not per-call price.

Recipe 6: Route routine and hard tasks ADVANCED

DIAGRAM

```
Routine extraction → fast model → schema validation
If validation fails → retry once with defect feedback
If still failing → stronger model with the source and the failure record
```

Log which branch each task took. After a hundred tasks you will know whether the fast model belongs in that slot.

Recipe 7: Decide whether to upgrade BEGINNER

Upgrade when all four are true:

1. the workflow is already reasonably efficient
2. the limit repeatedly blocks valuable work
3. the higher tier removes that specific constraint
4. the time or value recovered exceeds the added price

Do not upgrade because a pricing page makes the largest plan feel like the serious-user plan. That is a design decision, not a diagnosis.

REFERENCE

Beginner Subscription Efficiency Checklist

CHECKLIST

- [ ] I know the deliverable I need.
- [ ] I chose a model appropriate to the difficulty.
- [ ] I included necessary background and removed irrelevant detail.
- [ ] Every uploaded file can change the answer.
- [ ] I stated the audience, format, and length.
- [ ] I used deeper reasoning only when the task needs it.
- [ ] I asked for a targeted correction instead of a full rewrite.
- [ ] I verified factual claims.
- [ ] I captured state before leaving a long conversation.
- [ ] I diagnosed my actual limit before considering an upgrade.

REFERENCE

Project Context Checklist

FOR BEGINNER AND INTERMEDIATE

CHECKLIST

- [ ] One clear project purpose
- [ ] Defined audience
- [ ] Current controlling sources
- [ ] Obsolete sources removed or labeled
- [ ] Stable instructions separated from current tasks
- [ ] Source hierarchy stated
- [ ] Sensitive-data boundary reviewed
- [ ] Test question used to verify grounding
- [ ] Review date assigned to volatile facts
- [ ] Separate chats used for separate deliverables

REFERENCE

Long-Horizon Work Checklist

FOR INTERMEDIATE AND ADVANCED

CHECKLIST

- [ ] Phases are named and the current phase is known
- [ ] Decisions live in a durable file, not only in the chat
- [ ] A continuation brief exists and is current
- [ ] What must survive summarization is written down separately
- [ ] Tool results that are no longer needed have been cleared or are being cleared
- [ ] Independent review happens in a separate context
- [ ] Approval boundaries are stated in the working instructions

REFERENCE

Context Engineering Worksheet

FOR INTERMEDIATE

Outcome. What usable result must exist?

Quality threshold. What makes the result acceptable?

Persistent context. Which stable rules apply broadly?

Project context. Which goals, terms, sources, and decisions apply to this body of work?

Retrieved context. What evidence should be found only when needed?

Task context. What is new or different now?

Output contract. What format, length, structure, citations, and boundaries apply?

Excluded context. What must not influence the result?

Verification. How will accuracy and completeness be checked?

REFERENCE

API Token Economics Worksheet

FOR ADVANCED

PROMPT

```
Task class:

Requests per period:

First-pass success rate:

Average new input tokens:

Average cache-write tokens:

Average cache-read tokens:

Average output tokens:

Average reasoning tokens, if reported:

Tool, retrieval, and storage charges:

Retry rate:

Human correction minutes:

Latency requirement:

Quality threshold:

Cost per accepted task:

Candidate optimization:

Measured result after change:
```

REFERENCE

Decision Tree: Should I Start a New Chat?

DIAGRAM

```
Is the goal unchanged?
├─ No → Start a new chat.
└─ Yes
   Are the same instructions and sources controlling?
   ├─ No → Start a new chat with a handoff.
   └─ Yes
      Is old context causing confusion or repetition?
      ├─ Yes → Summarize state, then start fresh.
      └─ No → Continue the current chat.
```

REFERENCE

Decision Tree: Where Does This Context Belong?

DIAGRAM

```
Will it apply to almost everything I do?
├─ Yes → Persistent context. Keep it short.
└─ No
   Will it apply to this whole body of work?
   ├─ Yes → Project context.
   └─ No
      Will it be needed occasionally, from a larger set?
      ├─ Yes → Retrieve it when needed. Store a pointer, not the content.
      └─ No
         Is it new or specific to right now?
         ├─ Yes → Task context. Put it in the prompt.
         └─ No → It probably does not belong anywhere. Leave it out.
```

REFERENCE

Platform Optimization Cheat Sheet

VERIFY BEFORE USE

Product behavior as of September 4, 2026.

PLATFORM	USE DURABLE CONTEXT FOR	USE A FRESH CHAT FOR	WATCH CLOSELY
ChatGPT	Project instructions, project files, recurring work	New deliverables or independent reviews	Project memory settings, file limits, tool usage
Claude	Project knowledge, recurring source sets, long-form work	Milestones where old exploration interferes	Conversation length, attachments, model and tool usage
Gemini	Reusable Gems or instructions, genuinely large multimodal work	A changed objective or a clean comparison	Thinking level, context size, plan limits
Copilot Free	General web-grounded assistance	Unrelated topics or clean restarts	Capacity limits and source verification
Copilot Chat	Governed work or web chat under the signed-in account	A different source boundary or audience	License, account, grounding, organizational configuration
Copilot Notebooks	Curated Microsoft 365 reference sets	A different project or evidence boundary	Reference quality, obsolete files, rollout and license availability

REFERENCE

The Rules

> **In one line:** Every chapter rule in one place.

#	CHAPTER	THE RULE
1	Three Meters, Not One	Know which meter is running before you optimize anything.
2	What Actually Fills the Window	Give the model what it needs, not everything you have.
3	The Attention Budget	Find the smallest set of high-signal material that gets the outcome you need. Capacity is a ceiling, not a target.
4	Useful Work Is the Unit That Matters	Measure yourself on accepted work per unit of capacity, not on how little you typed.
5	Choosing the Lightest Capable Path	Use the least intensive route that reliably clears the quality bar.
6	Conversations, Projects, and Clean Restarts	Keep a conversation while it is still cheaper than rebuilding it, and not one turn longer.
7	Files, Sources, and Who Wins a Disagreement	Name the controlling source before you name the task.
8	Myths That Cost You Capacity	When a habit feels obviously right, check whether anyone has measured it.
9	The Context Engineering Stack	Put every piece of context in the layer that matches its lifespan.
10	Context Budgeting	Every item in context should have a reason and a review date.
11	The Six Ways Context Fails	Name the failure before you change anything, because the fixes are opposites. Starvation wants more context. Dilution wants less.
12	Conversation Architecture and State	The chat holds the current state. Something durable holds the decisions.
13	Output Contracts	Define done in the request, not in the third round of feedback.
14	Platform Playbooks	Choose the grounding boundary before you choose the prompt.
15	The Advanced Controls	Do not adopt a lever you cannot measure.

#	CHAPTER	THE RULE
16	The Economics of an AI Request	Divide by accepted tasks, and count the human minutes.
17	Prompt Caching	Stable first, dynamic last, then check the usage field.
18	Retrieval, Tools, and Structured Output	A tool the model cannot confidently choose between is a tool you should turn off.
19	Reasoning Budgets and Model Routing	Escalate on evidence, and carry the evidence, not the transcript.
20	Long-Horizon Work	Decide in advance what survives the summary, and write it somewhere the summary cannot reach.
21	Measurement	Baseline first, change one thing, measure the same way.

REFERENCE

Glossary

API	Application programming interface. A software interface that lets an application send work to a model and receive results, billed per token rather than by subscription.
Attention budget	The practical limit on how well a model can relate every part of its context to every other part. Every additional token draws it down.
Cache hit	Reuse of eligible previously processed prompt content.
Cache write	Initial storage or preparation of prompt content for later cached use.
Compaction	A provider or application process that represents older context in a smaller continuation state.
Context	The information available to the model for the current response.
Context editing	Removing specific material, typically old tool results, from the conversation before it is sent again.
Context engineering	Deliberate selection, organization, delivery, and maintenance of the information available to a model for a task.
Context rot	The measured decline in a model's reliability as input length grows.

Context window	The maximum working space a model can use for input and output under a particular request configuration.
Grounding	Connecting an answer to identified sources such as the web, files, organizational data, or notebook references.
Just-in-time context	Holding lightweight references and loading the underlying content only when the task requires it.
Latency	Time from request to usable result.
Model routing	Choosing a model or mode based on task requirements.
Output contract	Explicit acceptance conditions for the response.
Prompt caching	Reusing a stable portion of repeated input to reduce eligible processing cost or latency. A billing optimization, not a context optimization.
Reasoning budget	The amount of additional computational effort assigned to a task.
Retrieval	Selecting relevant evidence from a larger collection at request time.
Usage credits	A prepaid balance on a paid consumer plan that allows work to continue past the plan's included limit, billed at standard per-token rates. Not the same as the subscription, and not the same as a direct API account.
Structured note-taking	An agent writing notes to persistent storage outside the context window and reading them back later.
Structured output	A response constrained to a machine-readable schema.
Sub-agent	A separate agent with its own clean context window that performs a focused task and returns a condensed result.
Token	A processing unit used by a model for text or encoded input and output.
Tool context	Tool definitions, tool choices, calls, and results available to the model.

REFERENCE

Source Notes

Product behavior and research findings were checked against these sources on September 4, 2026. The links are maintenance points, not a claim that behavior will stay the same.

Principles and research

- Effective context engineering for AI agents <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>, Anthropic

- Context Rot: How Increasing Input Tokens Impacts LLM Performance <https://research.trychroma.com/context-rot>, Chroma

Anthropic product and platform

- Claude usage-limit best practices <https://support.claude.com/en/articles/9797557-usage-limit-best-practices>
- Claude Projects <https://support.claude.com/en/articles/9517075-what-are-projects>
- How do usage and length limits work? <https://support.claude.com/en/articles/11647753-how-do-usage-and-length-limits-work>
- Manage usage credits for paid Claude plans <https://support.claude.com/en/articles/12429409-manage-usage-credits-for-paid-claude-plans>
- Claude token counting <https://platform.claude.com/docs/en/build-with-claude/token-counting>
- Claude context windows <https://platform.claude.com/docs/en/build-with-claude/context-windows>
- Claude prompt caching <https://platform.claude.com/docs/en/build-with-claude/prompt-caching>
- Claude compaction <https://platform.claude.com/docs/en/build-with-claude/compaction>
- Claude context editing <https://platform.claude.com/docs/en/build-with-claude/context-editing>
- Claude memory tool <https://platform.claude.com/docs/en/agents-and-tools/tool-use/memory-tool>
- Managing context on the Claude Developer Platform <https://claude.com/blog/context-management>

OpenAI

- OpenAI prompt caching <https://developers.openai.com/api/docs/guides/prompt-caching>
- OpenAI compaction <https://developers.openai.com/api/docs/guides/compaction>
- OpenAI context management <https://developers.openai.com/api/docs/guides/context-management>
- OpenAI conversation state <https://developers.openai.com/api/docs/guides/conversation-state>
- OpenAI model guidance <https://developers.openai.com/api/docs/guides/latest-model>
- ChatGPT Projects <https://help.openai.com/en/articles/10169521>

Google

- Gemini context caching <https://ai.google.dev/gemini-api/docs/caching>
- Gemini app limits <https://support.google.com/gemini/answer/16275805>

Microsoft

- Microsoft Copilot experiences <https://support.microsoft.com/en-US/Microsoft-365-Copilot/what-s-the-difference-between-microsoft-copilot-free-and-copilot-in-microsoft-365>
- How Microsoft 365 Copilot Notebooks works <https://support.microsoft.com/en-us/microsoft-365-copilot/how-microsoft-365-copilot-notebooks-works>