Tokens, Context Windows and Cost

This guide is self-contained. You do not need the class video. Every term is explained the first time it appears, every number is worked out step by step, and you can check each calculation with a calculator. It answers four questions: what a token is, how much a request costs, how to keep the context window from filling up, and how to choose a model and settings that are fast, cheap and good enough.

> **Running example.** We use one food-seller app throughout. A seller in Mumbai types a dish name, category and price, and the app writes a short, appetising description. Each button click is one API call.

1. What a token is

An LLM does not read letters or words. It reads **tokens**: chunks of text that the model's tokenizer has learned to split out. A token can be a whole short word (`dish`), part of a long word (`Separ` + `ating`), a space plus a word, or punctuation. Rule of thumb for English: **one token is roughly four characters, or about three-quarters of a word.**

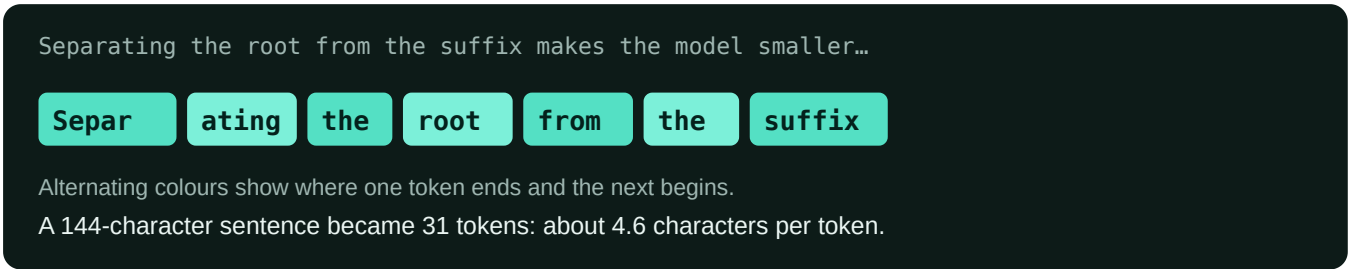


Figure 1. Tokens are not words. Common pieces are single tokens; rarer words split into several pieces. The example sentence (144 characters) is 31 tokens in one class tokenizer.

Why rare words cost more

The tokenizer was trained on a large text corpus. Frequent words such as *the* or *product* are single tokens. Rare words, names, code and non-English text break into more pieces, so they cost more tokens per idea. This is why an Indian-language or technical prompt can cost more than the same meaning in plain English. Always measure, never guess.

2. Input tokens and output tokens

Every API call has two sides, and each side is billed separately:

- **Input tokens** are everything you send: the system prompt (standing instructions), the user message, any examples, and any retrieved documents.
- **Output tokens** are everything the model writes back. Output tokens are usually priced higher than input tokens, and they are generated one by one, so they also take longer.

Total tokens per call = input tokens + output tokens. That total is what the usage report shows, and what the cost is based on.

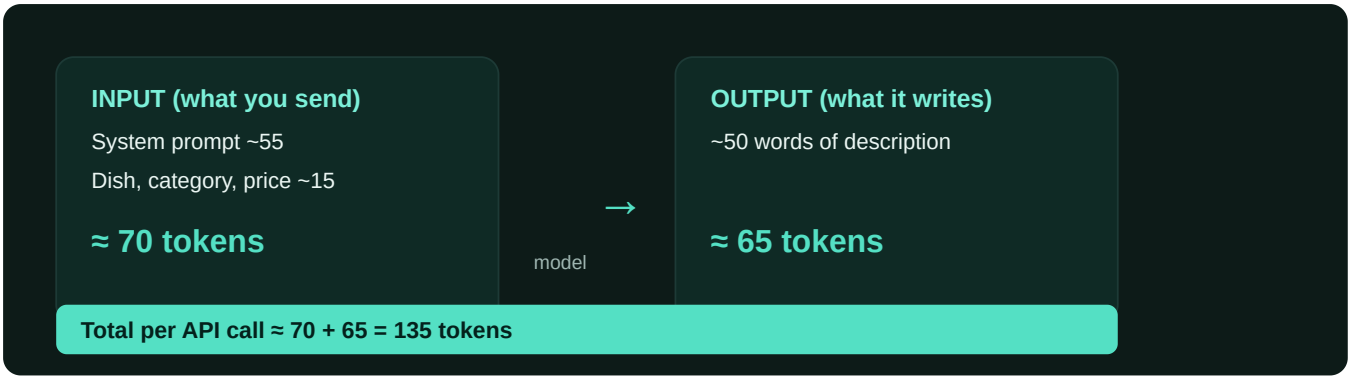


Figure 2. The seller-app call. Input is about 70 tokens, output about 65, total about 135. The system prompt is the part that repeats on every click.

3. Counting tokens before you ship

Measure first. Never estimate. Three ways to count:

1. **Count the prompt in the provider's console or token-counting endpoint.** This gives the exact input count before you pay for a call.
2. **Read the usage block in each response.** Every API response reports input and output tokens. Log them.
3. **In Claude Code or Claude Desktop, check the context usage panel** (see section 5). It shows how much of the window your session is using.

> **Common mistake:** counting only the words you typed. The system prompt, earlier turns in the conversation, examples and tool definitions are all input, and they are resent on every call.

4. Pricing, with a worked cost per call

Prices change, so the numbers below use the class example: **Claude Haiku at $0.80 per million input tokens and $4 per million output tokens.** Before you publish any cost, check the current rate on Anthropic's pricing page: anthropic.com/pricing. Use the formula, and replace the rates with today's rates.

Formula: cost per call = (input tokens ÷ 1,000,000 × input rate) + (output tokens ÷ 1,000,000 × output rate)

Worked example: one seller-app call

Step	Calculation	Result
Input cost	70 ÷ 1,000,000 × $0.80	$0.000056
Output cost	65 ÷ 1,000,000 × $4.00	$0.000260
Total per call	$0.000056 + $0.000260	**$0.000316**

So one generation costs about **$0.0003**, which is roughly 3 hundredths of a cent. The cost looks trivial, but it multiplies. Output is about 82% of the cost here, even though it is under half the tokens. That is why output length is the first thing to control.

From one call to a monthly bill

Multiply the cost per call by the number of calls. Use three scenarios so the business survives the bad case, not only the average one:

Scenario	Generations per dish	Example: 200 sellers × 15 dishes	Monthly cost (USD)
Conservative	1.2	3,000 dishes × 1.2 = 3,600 calls	3,600 × $0.000316 ≈ $1.14
Realistic	1.8	3,000 × 1.8 = 5,400 calls	5,400 × $0.000316 ≈ $1.71
Aggressive	2.8 (cap 3)	3,000 × 2.8 = 8,400 calls	8,400 × $0.000316 ≈ $2.65

The table shows the arithmetic, not a forecast. Use your own seller count and rates. Add 2 new dishes per seller per month to the dish count each month. The rule to take away: **a cost is only meaningful once it is multiplied by volume, and volume grows over time.** Decide whether to charge for generations, give a free allowance, or absorb the cost, using the conservative case as your floor.

Worked example: a multi-call feature ("Find my matches")

One click on a job-matching product can run 25 model calls. Price each call type separately, because they differ in model and size. The token counts below are **assumptions for teaching**; measure your own before you publish.

Call type	Model	Calls	Input tokens / call	Output tokens / call	Why this model
Query generation	Haiku	1	160	60	Short, structured task
Web search (external API)	Search API	4	0 tokens	0	Billed per search, not per token
Listing extraction	Sonnet	20	1,200	150	Reading messy web pages needs accuracy
Deduplication	Haiku	1	900	80	Simple comparison
Scoring and ranking	Sonnet	1	1,500	200	Judgement across the candidate's profile
Final synthesis	Sonnet	1	1,800	400	Writes the explanations users read

Cost per click, using Haiku at $0.80 / $4 and Sonnet at an *example* $3 / $15 per million (check current rates):

Call type	Input cost	Output cost	Subtotal per call × count
Query generation (Haiku, 1)	160 × $0.80/M = $0.000128	60 × $4/M = $0.000240	$0.000368
Listing extraction (Sonnet, 20)	1,200 × $3/M = $0.003600	150 × $15/M = $0.002250	$0.005850 × 20 = $0.117000
Deduplication (Haiku, 1)	900 × $0.80/M = $0.000720	80 × $4/M = $0.000320	$0.001040
Scoring (Sonnet, 1)	1,500 × $3/M = $0.004500	200 × $15/M = $0.003000	$0.007500
Synthesis (Sonnet, 1)	1,800 × $3/M = $0.005400	400 × $15/M = $0.006000	$0.011400
Total per click			**≈ $0.137 (plus search API fees)**

What this teaches: 20 extraction calls on Sonnet are about 85% of the token cost. The product lever is obvious: move extraction to a cheaper model, or trim the listing text before it reaches the model (see section 7). Notice that the search API is billed separately, so cost per click is tokens *plus* external fees.

5. The context window: your working memory

The **context window** is the total number of tokens the model can hold in one session at once: your messages, its replies, attached files, tool results and the standing instructions. Think of it as the model's desk. When the desk is full, older items must be removed or summarised.

Sizes vary by model and plan. Check the current figure in the model documentation for the model you use. Your own Claude Code usage screen in the class showed a window of about **967k tokens** for that model, so do not assume one fixed number.

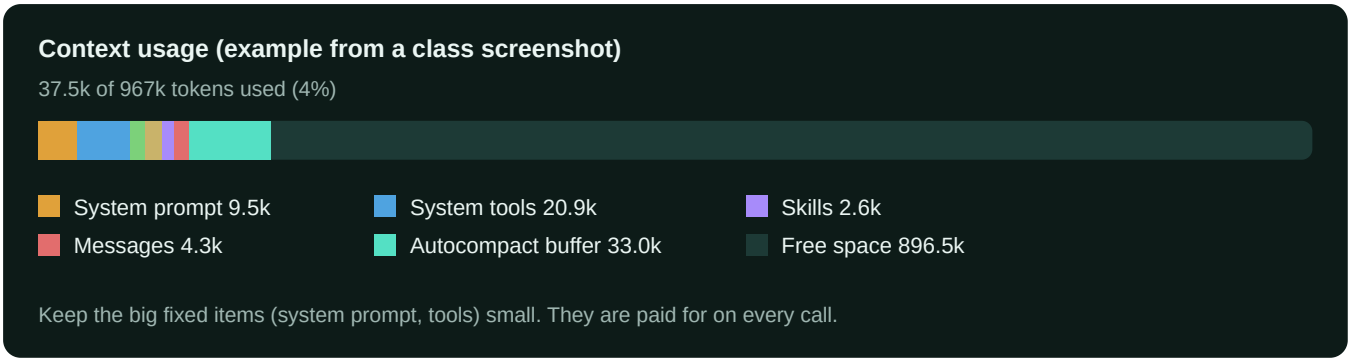


Figure 3. What fills a context window. The autocompact buffer (teal) is space kept back so the tool can summarise older messages before the window overflows.

How to check usage

- **In Claude Code:** type `/context` to see the breakdown: system prompt, tools, memory files, skills, messages and free space.
- **In an API response:** read the usage fields for input and output tokens on every call.
- **In a long chat:** start a fresh session when the topic changes. A new session starts empty, so it costs less per message.

Why replies cost more than prompts. Each reply is generated one token at a time, and it becomes part of the history that is resent with the next message. Output tokens are priced higher, so a long answer costs more than a long question. Ask for the shape you need ("five bullets, under 80 words") to keep replies short.

6. How the model chooses each word: attention and sampling

Attention: not every token gets equal weight

When the model processes the word *some* in "Riya poured tea into the cup and added some...", it does not treat every earlier word equally. It compares the current token with every earlier token, gives each a relevance score, turns the scores into weights that sum to 1, and mixes the earlier words according to those weights. In that sentence, *tea* and *cup* carry strong weight. The output

is a blend of the most relevant earlier tokens, so the same word *some* ends up with a different internal meaning depending on its context.

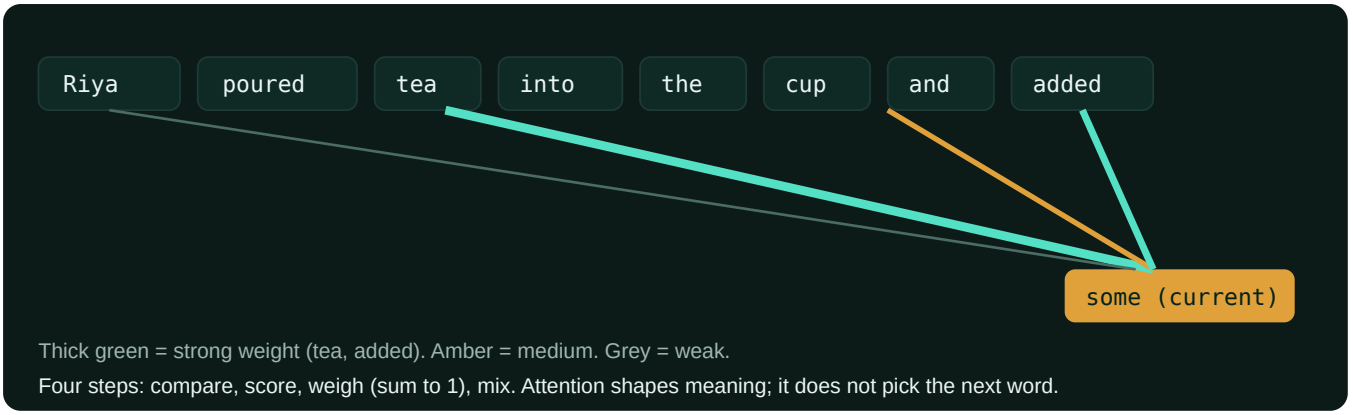


Figure 4. Attention as a weighted mix over earlier tokens. The weights are learned, not hand-written.

Sampling: how the next token is picked

After the model scores every possible next token, a sampling setting decides how the choice is made. These are the four controls you will see in most APIs:

Control	What it does	Use it when	Cost effect
Temperature	Lower = more predictable; higher = more varied. 0 is near-deterministic.	Structured output such as JSON: 0 to 0.3. Creative copy: 0.7 to 1.0.	None directly, but very high values can make output longer and less useful.
Top-k	Only the k most likely next tokens are candidates.	Stop rare, odd words from slipping in.	None directly.
Top-p (nucleus)	Keep the smallest set of tokens whose probabilities add up to p (for example 0.9).	A moderate way to balance variety and reliability.	None directly.
Max tokens	A hard limit on how many output tokens are generated.	Always set it. It caps output cost.	**Direct:** sets the largest possible output bill per call.

> **Rule of thumb for the seller app:** temperature 0.7 for the description, max tokens about 120 (enough for 50 words), and temperature 0 for any JSON the app must parse. Tune on real examples, then lock the values.

7. Using fewer tokens and getting more from each one

1. **Shorten the standing prompt, but keep the structure.** Say what the job is, the audience, the length and the format in one short block. Remove filler such as "you are a very helpful

assistant".

2. **Ask for structured output.** A JSON array of 4 search queries (about 60 output tokens) is cheaper and easier for code to use than three paragraphs of explanation.
3. **Trim large inputs before they reach the model.** Send the fields that matter, not the whole web page. Extract the job title, skills and location first, then pass only those.
4. **Use XML or clear tags to separate parts.** For example `<role>`, `<instructions>`, `<input>`. Tags help the model find each section, which reduces retries, and each retry costs tokens.
5. **Route each task to the smallest model that passes.** Use Haiku for short, structured steps and Sonnet only where judgement matters.
6. **Cap output.** Set max tokens and ask for a length ("under 50 words").
7. **Reuse, do not repeat.** Put standing instructions in a skill or project instructions, so you are not retyping them and the tool loads them efficiently (see the skills guide in this series).
8. **Cache what does not change.** Repeated prompts that share the same long system prompt can be cheaper when the provider supports prompt caching. Check your provider's current caching rules before you rely on it.
9. **Start a fresh session** when the topic changes, instead of carrying a long history into an unrelated question.

8. Choosing a model: fast, cheap or creative

Model family	Best for	Speed	Cost	Quality on hard reasoning
Haiku (small)	Classification, short copy, extraction of simple fields, query generation	Fastest	Lowest	Good for simple, well-defined steps
Sonnet (medium)	Reading messy inputs, scoring, most product features	Medium	Medium	Strong general quality
Opus (large)	Hard analysis, complex planning, high-stakes reasoning	Slowest	Highest	Strongest

Check the current list and prices in the provider documentation, since names and rates change. The decision rule: **start with the smallest model, measure quality on 20 real examples, and move up only where the smaller model fails.** Do not pay for a large model to do a simple job.

9. Limits and guardrails

- **Context limit:** the request fails or gets truncated when the window is full. Keep inputs under the limit with room for the answer.
- **Rate limit:** providers cap requests per minute. Retry with a short wait; do not retry in a tight loop, because each retry is billed.

- **Budget limit:** set a spending cap in the provider console, and log cost per feature so one bug cannot drain the budget.
- **Output limit:** max tokens stops generation early. Make sure the answer is still complete at that limit.

10. Put it into practice

> **Exercise (20 minutes).** Take any prompt you already use. (1) Count its input tokens with your provider's counter. (2) Run it once and read the output count. (3) Work out the cost with the formula in section 4 using current rates. (4) Shorten the system prompt, cap max tokens, and run it again. (5) Record the new totals and the difference in cost. Write one sentence on what you cut and whether quality held.

Quick check (answers are in the next section)

1. A reply is 65 output tokens and the prompt is 70 input tokens. Which costs more per token at the example rates?
2. Why does the system prompt matter even though you type it only once?
3. Which sampling setting caps the cost of a single call?
4. Your JSON output breaks sometimes. Which control do you lower first?

11. Answers and glossary

1. Output, at $4 per million tokens versus $0.80 for input, so output is 5 times the rate. **2.** It is resent on every call, so it is paid for on every click. **3.** Max tokens. **4.** Temperature, down towards 0.

Term	Plain meaning
Token	A small piece of text the model reads and writes, roughly four characters.
Input tokens	Everything sent to the model in one call.
Output tokens	Everything the model writes back.
Context window	The total tokens the model can hold in one session.
Attention	The model's way of weighing earlier tokens by relevance.
Temperature	How predictable or varied the chosen words are.
Max tokens	The hard limit on output length per call.
Autocompact buffer	Space kept free so older messages can be summarised before the window overflows.

12. Trusted sources to read next

- Anthropic pricing: anthropic.com/pricing (check current rates before any cost estimate).
- Anthropic documentation on models, context windows and token counting: docs.anthropic.com.
- Anthropic prompt engineering guide: docs.anthropic.com (search "prompt engineering").

Prices and model names in this guide were checked against the class material and should be re-checked on the date you publish. Token counts marked "assumption" are for teaching; measure your own before relying on them.

13. Extension: step-by-step exercises for tokens and cost

Exercise 1: count a real prompt (15 minutes)

1. Write the system prompt for the seller app on one screen, then copy it into your provider's token counter (or a text-length estimate of about 4 characters per token, to start).
2. Add one sample dish (name, category, price) and count again.
3. **You should see:** a total near the 70-token input example from section 2, or your own number.
4. Run the call once. Note the input and output counts in the usage data.

Exercise 2: cut the cost by a third

1. Shorten the system prompt by removing filler words. Keep the job, audience, length and tone.
2. Set max tokens to a value that fits your 50-word target (about 80–120 tokens).
3. Run the same three dishes before and after. Compare output tokens and read the descriptions for quality.
4. **Pass when:** output tokens fall and every description still meets the brief.

Exercise 3: choose a model

Task	Smallest model to try first	Escalate only if
Classify a dish as Breakfast, Lunch, Dinner or Dessert	Small (Haiku-class)	Accuracy is under your target on 20 examples
Write a 50-word description	Small (Haiku-class)	Tone or fluency fails the review checklist
Compare two competing descriptions and pick the better one	Medium (Sonnet-class)	Judgement is inconsistent on the test set

Current model names, limits and prices come from the provider's documentation on the day you decide. The class figures ($0.80 and $4 per million tokens for Haiku) are a historical example only.

14. Diagram: where the tokens go

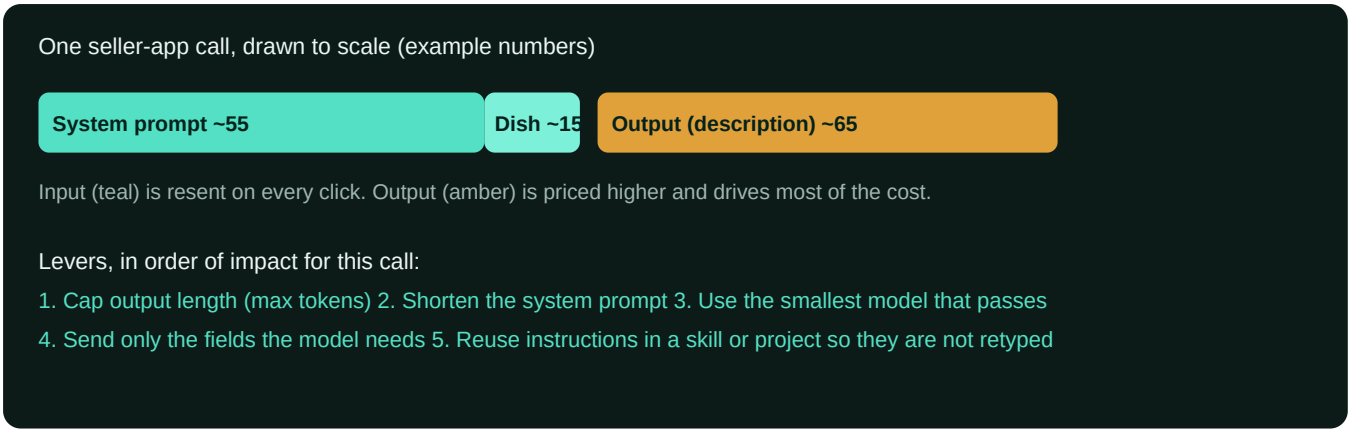


Figure 4. Token budget per call, with the levers ranked by impact.

15. Advanced: prompt caching and batching (concepts)

- **Caching:** if many requests share the same long instructions, some providers can reuse that part at a lower rate. Check the provider's current rules before you design around it.
- **Batching:** sending many non-urgent requests together, where the provider offers it, can lower cost. It adds delay, so use it only where waiting is acceptable.
- **Structured output:** asking for JSON with a fixed schema shortens output and makes the next step reliable.

16. Extended answer key and checklist

Item	Done
Counted input and output for one real call	[ ]
Set max tokens on every call	[ ]
Chose the smallest model that passes a test set of 20	[ ]
Worked the cost per call with current rates	[ ]
Logged usage for every feature	[ ]