

A BEGINNER'S MANUAL

GROK EDITION

AI APIs & Costs

Keys, tokens, prices and budgets for OpenAI,
Anthropic, Google and xAI, explained for beginners

AI APIs & Costs

Keys, tokens, prices and budgets for OpenAI, Anthropic, Google and xAI, explained for beginners

GROK EDITION Written by Grok Bot, with facts checked against official sources in October 2026.

A chat subscription and an API are different products with different bills. This manual explains how API pricing really works: what a token is, why output costs more than input, and how caching, batch processing, long prompts and "thinking" change the price. It shows you how to get keys from OpenAI, Anthropic, Google and xAI, cap your spending, estimate a project's cost before you start, and cut that cost in half or more.

Every price in this book was copied from the provider's official pricing page on October 16, 2026, and is marked with that date. Prices change often, and several listed here are temporary promotions with announced end dates. Always confirm on the official page before you budget.

HOW TO READ THIS MANUAL

Tip — a shortcut or good habit.

Warning — easy to get wrong.

\$ command — text you type, exactly as shown.

Note — useful background.

Danger — can lose data, money, or access.

Settings > Privacy — a menu path to click.

New words appear in **bold** the first time and are collected in the Glossary at the back.

Prices, limits and model names were checked against official provider pages on October 16, 2026, and will change. Always confirm current pricing on the provider's site before relying on it. All prices are in US dollars, exclude tax, and are per 1 million tokens unless stated.

Contents

01 Subscriptions vs. APIs	5
1.1 Two products, two bills	5
1.2 Do you even need an API?	6
1.3 How API billing works	6
02 Tokens, Explained	7
2.1 Rules of thumb (English text)	7
2.2 What counts as input	8
2.3 What counts as output	8
2.4 Count tokens exactly	8
03 Getting Keys and Setting Limits	10
3.1 OpenAI	10
3.2 Anthropic (Claude API)	11
3.3 Google (Gemini API)	11
3.4 xAI (Grok API)	12
3.5 Key safety checklist	13
04 Price Tables (Checked October 16, 2026)	14
4.1 OpenAI	14
4.2 Anthropic (Claude)	15
4.3 Google (Gemini API)	16
4.4 xAI (Grok)	16
4.5 Side-by-side: the cost of one typical task	17
05 Discounts: Caching, Batch, and Flex	18
5.1 Prompt caching	18
5.2 Batch processing	19
5.3 Flex and other service tiers	20
5.4 Discounts stack	20
06 Estimating a Project's Cost Before You Start	21
6.1 The four numbers	21
6.2 The calculator	22
6.3 Budget worksheet	22
6.4 Put a dollar limit in the code itself	23
07 Twelve Ways to Cut Your Bill	24
7.1 Routing example	25
08 Rate Limits, Errors, and Reliability	26

8.1	The limits	26
8.2	Errors and what to do	27
8.3	Retry correctly	27
09	Beyond Text: Tools, Images, Audio, and Agents	28
9.1	Tool and feature fees (October 16, 2026)	28
9.2	Images, audio and video (examples)	29
9.3	Agents cost more than you think	29
10	Subscription or API? Doing the Math	30
10.1	Break-even thinking	30
10.2	Free ways to experiment	31
11	Cost Recipes for Common Projects	32
12	Monitoring and Alerts	33
12.1	Weekly 5-minute check	33
12.2	Log usage from your own code	33
13	Troubleshooting and FAQ	34
14	Choosing a Model: A 30-Minute Test	35
14.1	Build a test set	35
14.2	Run it against several models	36
14.3	Decide	36
15	Embeddings: The Cheap Way to Search Your Documents	37
15.1	Prices (October 16, 2026)	37
15.2	The RAG pattern in five steps	37
16	Raw HTTP: Calling Any API with curl	39
17	Using Your Key in No-Code Tools	41
18	Data Use and Terms at a Glance	42
	Quick Reference	43
	Glossary	44

Subscriptions vs. APIs

There are two completely separate ways to pay for AI. Mixing them up is the most common and most expensive beginner mistake.

IN THIS CHAPTER

- A chat subscription vs. an API account
- When you need an API at all
- How API billing works in one page

1.1 Two products, two bills

	CHAT SUBSCRIPTION	API
Examples	ChatGPT Plus, Claude Pro, Google AI Pro, SuperGrok, Perplexity Pro	OpenAI API, Claude API, Gemini API, xAI API
You use it through	The company's app or website	Your own code, or tools you connect with a key
You pay	A flat monthly fee	Per use (per token), like an electricity meter
Limits	Usage caps that reset over time	Rate limits, plus whatever you're willing to spend
Where you sign up	chatgpt.com, claude.ai, gemini.google.com, grok.com	platform.openai.com, platform.claude.com, aistudio.google.com, console.x.ai

☆ IMPORTANT

A subscription **does not include API credit**, and API credit doesn't unlock subscription features. ChatGPT Plus doesn't pay for your Python script's OpenAI calls. They're separate accounts with separate billing, even when you sign in with the same email.

1.2 Do you even need an API?

YOU WANT TO...	USE
Chat, write, research and analyze files yourself	A subscription, or a free plan (see Vol. 20, <i>Prompt Maximization</i>)
Run AI on hundreds of rows or files automatically	API
Add AI to your own app, website or bot	API
Connect AI to Zapier, Make or n8n workflows	API (most automation tools ask for your key)
Keep everything private on your own PC	Neither: a local model (Vol. 11)

1.3 How API billing works

1 You send a request.

It contains your prompt, any documents, the conversation so far, and instructions. All of it is counted as **input tokens**.

2 The model replies.

Everything it writes, including any hidden “thinking”, is counted as **output tokens**.

3 You're charged.

Cost = input tokens × input price + output tokens × output price. Prices are quoted **per million tokens** (often written “MTok”).

4 Charges add up.

Most providers deduct from prepaid credit, or bill your card monthly.

A worked example. You send a 3,000-token document and get a 500-token summary from a model priced at \$2.00 input / \$10.00 output per million tokens:

TEXT

```

Input:   3,000 × $2.00 / 1,000,000 = $0.0060
Output:   500 × $10.00 / 1,000,000 = $0.0050
Total per summary                = $0.0110   (about 1.1 cents)
1,000 summaries                  ≈ $11.00

```

That's the whole game. The rest of this book is about getting the numbers in that formula right and making them smaller.

CHAPTER 2

Tokens, Explained

A **token** is the unit AI models read, write and bill by. It's a chunk of text, often a whole short word, part of a longer word, or a piece of punctuation.

IN THIS CHAPTER

- Rules of thumb for converting words and pages to tokens
- Why different models count the same text differently
- Counting tokens exactly before you send

2.1 Rules of thumb (English text)

AMOUNT OF TEXT	ROUGHLY
1 token	¾ of a word, or about 4 characters
100 tokens	75 words
1,000 tokens	750 words, or 1½ pages of single-spaced text
1 page of a book	400–500 tokens
A 300-page novel	120,000–150,000 tokens
1 million tokens	750,000 words, or about 10 novels

These are averages. Code, numbers, tables, non-English text and unusual names use **more** tokens per word.

NOTE

Tokenizers differ between companies and even between model generations. Anthropic notes that Claude 4.7 and later models use a newer tokenizer that produces **about 30% more tokens for the same text** than earlier Claude models. When comparing prices across providers, compare the cost of the *same task*, not just the per-token price.

2.2 What counts as input

People are often surprised by how much input a “short” request contains:

INPUT PART	COUNTED?	NOTES
Your message	Yes	
System prompt or instructions	Yes	Every request, unless cached
The entire conversation so far	Yes	Chat “memory” is just re-sending history
Uploaded documents and images	Yes	Images are converted to tokens. A page image can cost hundreds to over a thousand tokens
Tool definitions	Yes	Each tool’s name, description and schema
Web search results the model reads	Usually	Plus a per-search fee on most platforms

2.3 What counts as output

Everything the model generates, **including reasoning (“thinking”) tokens** on models that think before they answer. A short final answer can cost far more than it looks if the model thought for thousands of tokens first. Most APIs let you control this with an **effort** or **reasoning** setting and a **max output tokens** cap.

2.4 Count tokens exactly

PROVIDER	HOW TO COUNT BEFORE SENDING
Anthropic	<code>client.messages.count_tokens(model=..., messages=[...])</code> . It’s free, and it returns <code>input_tokens</code>
Google	<code>client.models.count_tokens(model=..., contents=...)</code>
OpenAI	The <code>usage</code> field on every response. For estimates, the open-source <code>tiktoken</code> library
xAI	The <code>usage</code> field on every response
Any provider	Run 5–10 real samples and read the <code>usage</code> numbers. This is the most reliable method

COUNT_CLAUDE.PY

```
import anthropic
from dotenv import load_dotenv

load_dotenv()
client = anthropic.Anthropic()
count = client.messages.count_tokens(
    model="claude-haiku-5-5",
    messages=[{"role": "user", "content": open("report.txt", encoding="utf-8").read()}],
)
print(count.input_tokens)
```

 TIP

Count tokens on a few real documents before you build anything that runs in a loop. Multiply the input count by your expected number of calls and the model's price per million tokens, and you have a first cost estimate before you spend a cent.

Getting Keys and Setting Limits

Set up each provider in the same safe order: **account** → **billing** → **spending cap** → **key** → **test**. Never create the key first.

IN THIS CHAPTER

- Step-by-step setup for OpenAI, Anthropic, Google and xAI
- Where each provider's spending controls live
- Storing keys safely

3.1 OpenAI

1

Sign up.

Go to **platform.openai.com** and sign in. It's separate from your ChatGPT subscription, even with the same login.

2

Add billing.

Open the billing settings and add a payment method or buy credit. OpenAI's usage tiers start with **Build**, unlocked after \$5 in total credit purchases.

3

Set spend controls.

OpenAI offers a **spend alert** (a notification, while traffic continues) and a **hard spend limit** (requests return error `429` once reached), for the organization or for each project.

4

Make a project.

Create a separate project for each app, so their costs and limits stay separate.

5 Create a key.

Under API keys, create a key scoped to that project. Copy it once and store it in a `.env` file.

Usage tiers (from OpenAI's rate-limits guide, October 2026):

TIER	QUALIFICATION	MONTHLY USAGE LIMIT
Free	Allowed geography	\$100 / month
Build	\$5 total credit purchases	\$500 / month
Launch	\$100 total credit purchases	\$5,000 / month
Grow	\$500 total credit purchases	\$200,000 / month

3.2 Anthropic (Claude API)

1 Sign up.

Go to **platform.claude.com** (the Claude Console). Anthropic says new users receive a small amount of free credit to test the API.

2 Buy credit.

Go to **Billing** (`platform.claude.com/settings/billing`).

3 Set a spend limit.

In the **Spend limits** section, click **Set limit** or **Adjust limit**. Your limit can't exceed your tier's cap.

4 Use workspaces.

Create a workspace for each project, and set lower spend and rate limits on each one.

5 Create a key in the API keys section, and store it as `ANTHROPIC_API_KEY`.

Monthly spend caps by tier (Anthropic rate-limits docs, October 2026): Start \$500, Build \$1,000, Scale \$200,000. Custom tiers are arranged with sales. When you hit the tier cap, usage pauses until 00:00 UTC on the first of the next month, unless you get a higher tier. When you hit **your own** spend limit, requests return HTTP 400 until you raise it.

3.3 Google (Gemini API)

1 Open AI Studio.

Go to **aistudio.google.com** and sign in with a Google account.

- 2 **Get a key.**
Click **Get API key** and create one in a new or existing Google Cloud project. You can start on the **free tier** with no card.
- 3 **Understand the free tier.**
Some models are free to use, with lower rate limits. Google's pricing page says free-tier content **is used to improve Google's products**. Paid-tier content is not.
- 4 **Enable billing to upgrade.**
Link a Cloud billing account to move to **Tier 1**. The higher tiers (2 and 3) unlock with cumulative spend.
- 5 **Set a budget alert.**
In the Google Cloud console's **Billing** → **Budgets & alerts**, create a monthly budget with email alerts at 50%, 90% and 100%. Note that budgets **alert** you; they don't automatically stop spending.

Gemini usage tiers (rate-limits page, updated October 15, 2026):

TIER	QUALIFICATION	BILLING TIER CAP	SPEND RATE LIMIT (PER 10 MIN)
Free	Active project or free trial	—	—
Tier 1	Linked active billing account	\$250	\$10
Tier 2	\$100 paid + 3 days since first payment	\$2,000	\$50
Tier 3	\$1,000 paid + 30 days since first payment	\$20,000–\$100,000+	\$200

Daily request quotas reset at **midnight Pacific time**, and rate limits apply **per project**, not per key.

3.4 xAI (Grok API)

- 1 **Sign up.**
Go to **console.x.ai**.
- 2 **Buy prepaid credits.**
Use **Billing** → **API spend management**. With the default **\$0 invoiced billing limit**, xAI only uses prepaid credit, and requests are rejected once it runs out. That's a natural hard cap.
- 3 **Decide on auto top-up.**
Auto top-up buys more credit when your balance falls below a threshold. You set the minimum top-up (at least \$5) and a monthly maximum. Leave it off while you're learning.
- 4 **Create a key** and store it as `XAI_API_KEY`.

3.5 Key safety checklist

- ☐ ☐ One key per project, so you can revoke one without breaking everything
- ☐ ☐ Keys live only in `.env` files or a secrets manager, never in code, screenshots or chats
- ☐ ☐ `.env` is listed in `.gitignore`
- ☐ ☐ A spend cap or prepaid-only billing is set **before** the first run
- ☐ ☐ You know how to revoke a key on each provider's key page
- ☐ ☐ Usage dashboards are bookmarked, and you check them weekly

 DANGER

If a key ever appears in a public place (GitHub, a forum, a shared screenshot), assume it's stolen. Revoke it right away, create a new one, and check the usage page for activity you don't recognize.

Price Tables (Checked October 16, 2026)

These tables list the main **text** models from each provider, copied from their official pricing pages on **October 16, 2026**. Every price is in US dollars **per 1 million tokens** at standard (on-demand) rates. Where a provider has announced a future change, the end date is shown.

⚠ WARNING

Prices change frequently, sometimes monthly. Several prices below are **temporary promotions** with published end dates. Before budgeting anything real, open the official page listed under each table.

4.1 OpenAI

MODEL	INPUT	CACHED INPUT	OUTPUT	NOTES
gpt-6-astra	\$10.00	\$1.00	\$50.00	Flagship
gpt-6.1-sol	\$2.00	\$0.10	\$10.00	
gpt-6-sol	\$2.00	\$0.20	\$10.00	
gpt-6-luna	\$0.10	\$0.01	\$0.50	Lowest cost GPT-6
gpt-5.6-sol	\$4.00	\$0.40	\$20.00	Promotional pricing "at least through November 21, 2026"
gpt-5.6-terra	\$2.00	\$0.20	\$12.00	
gpt-5.6-luna	\$0.20	\$0.02	\$1.20	
gpt-5-mini	\$0.25	\$0.025	\$2.00	Older, still listed
gpt-5-nano	\$0.05	\$0.005	\$0.40	Older, still listed

Long context: prompts over 272K input tokens are billed at higher “long context” rates on the newer models. For example, gpt-6.1-sol long-context input is \$4.00 and output \$15.00. **Batch** and **Flex** processing are half price. **Fast** mode costs about double. Web search is \$10 per 1,000 calls, plus the search content tokens. Data-residency endpoints cost 10% more for models released on or after March 5, 2026. *Source: developers.openai.com/api/docs/pricing*

4.2 Anthropic (Claude)

MODEL	API ID	INPUT	CACHE READ	OUTPUT
Claude Fable 5.1	<code>claude-fable-5-1</code>	\$10	\$0.25	\$50
Claude Opus 5.5	<code>claude-opus-5-5</code>	\$4	\$0.20	\$20
Claude Opus 5	legacy	\$5	\$0.50	\$25
Claude Sonnet 5.5	<code>claude-sonnet-5-5</code>	\$2	\$0.10	\$10
Claude Sonnet 5	legacy	\$2	\$0.20	\$10
Claude Haiku 5.5, prompts ≤100K tokens	<code>claude-haiku-5-5</code>	\$0.10	\$0.01	\$0.50
Claude Haiku 5.5, prompts >100K tokens	<code>claude-haiku-5-5</code>	\$0.50	\$0.05	\$2.50
Claude Haiku 4.5	legacy	\$1	\$0.10	\$5

Cache writes cost 1.25× the input price for a 5-minute cache, or 2× for a 1-hour cache. **Batch** is 50% off input and output. Anthropic notes that the Sonnet 5 price of \$2/\$10, originally introductory, is now the standard price. Claude 4.6 and later models (except Haiku 5.5) include the full 1M-token context at standard prices. US-only inference (`inference_geo: "us"`) costs 1.1×. Web search is \$10 per 1,000 searches. *Source: docs.anthropic.com/pricing page (platform.claude.com/docs)*

4.3 Google (Gemini API)

MODEL	INPUT	OUTPUT	FREE TIER?	SCHEDULED CHANGE
gemini-4-argon	\$2.00	\$10.00	No	Becomes \$4.00 / \$20.00 on December 1, 2026
gemini-3.8-flash	\$0.75	\$3.75	Yes	Becomes \$1.50 / \$7.50 on January 1, 2027
gemini-3.6-flash	\$0.75	\$3.75	Yes	Becomes \$1.50 / \$7.50 on January 1, 2027
gemini-3.5-flash-lite	\$0.30	\$2.50	Yes	
gemini-3.1-flash-lite	\$0.25	\$1.50	Yes	Audio input \$0.50
gemini-3.1-pro-preview, prompts ≤200K	\$2.00	\$12.00	No	>200K: \$4.00 / \$18.00
gemini-2.5-pro, prompts ≤200K	\$1.25	\$10.00	Yes	>200K: \$2.50 / \$15.00
gemini-2.5-flash	\$0.30	\$2.50	Yes	Audio input \$1.00
gemini-2.5-flash-lite	\$0.10	\$0.40	Yes	

Output prices **include thinking tokens**. **Batch** is 50% off. **Grounding with Google Search** on Gemini 3 and newer models gives 5,000 free search requests per month, shared across those models, then costs \$14 per 1,000. Google AI Studio itself is free to use. On the free tier, your content is used to improve Google's products. *Source: ai.google.dev/gemini-api/docs/pricing (page dated 2026-10-16)*

4.4 xAI (Grok)

MODEL	CONTEXT	INPUT	CACHED INPUT	OUTPUT
grok-4.7, prompts <200K	500K	\$2.00	\$0.50	\$6.00
grok-4.7, prompts ≥200K	500K	\$4.00	\$1.00	\$12.00
grok-4.3, prompts <200K	1M	\$1.25	\$0.20	\$2.50
grok-4.3, prompts ≥200K	1M	\$2.50	\$0.40	\$5.00
grok-build-0.1, prompts <200K	256K	\$1.00	\$0.20	\$2.00
grok-4.20-multi-agent-0309, prompts <200K	1M	\$1.25	\$0.20	\$2.50

xAI's long-context rule: once a prompt reaches the threshold, **all tokens in that request** are billed at the higher rate. Images cost \$0.02–\$0.05 each (grok-imagine models), and video \$0.02–\$0.08 per second. *Source: docs.x.ai/docs/models*

4.5 Side-by-side: the cost of one typical task

Task: summarize a 4,000-token document into a 400-token summary, with no caching. Calculated from the tables above:

MODEL	COST PER TASK	PER 1,000 TASKS
gpt-6-luna	\$0.00060	\$0.60
claude-haiku-5-5	\$0.00060	\$0.60
gemini-2.5-flash-lite	\$0.00056	\$0.56
gemini-3.1-flash-lite	\$0.00160	\$1.60
gemini-3.8-flash (to Dec 31, 2026)	\$0.00450	\$4.50
grok-4.3	\$0.00600	\$6.00
claude-sonnet-5-5	\$0.01200	\$12.00
gpt-6.1-sol	\$0.01200	\$12.00
grok-4.7	\$0.01040	\$10.40
claude-opus-5-5	\$0.02400	\$24.00
gpt-6-astra	\$0.06000	\$60.00

TIP

Cheap models handle summarizing, categorizing, extraction and rewriting very well. Use flagship models for hard reasoning, tricky code and judgment calls. Always test the cheap one first. You may never need the expensive one.

Discounts: Caching, Batch, and Flex

The list price is the most you'll pay. Three features can cut it substantially.

IN THIS CHAPTER

- Prompt caching, for repeated instructions and documents
- Batch processing, for anything that can wait
- Flex and off-peak tiers

5.1 Prompt caching

If many requests start with the same long text, such as a 20-page manual, a big system prompt or a codebase, the provider can store that **prefix** and charge a small fraction to reuse it.

PROVIDER	HOW IT WORKS	CACHE READ PRICE
OpenAI	Automatic on supported models when prompts share a long identical beginning	"Cached input" column. Often 10% (or 5%) of input
Anthropic	Add <code>cache_control</code> , either automatically at the top level of the request or on specific content blocks. Writes cost 1.25× (5 min) or 2× (1 hour)	0.1× input (0.05× on Opus/Sonnet 5.5, 0.025× on Fable/Mythos 5.1)
Google	Implicit caching on newer models, plus explicit caching on some models (storage is billed per hour)	"Context caching price" in the tables
xAI	Automatic on repeated prefixes	"Cached input" column

Rules for good cache hits:

1. Put the **unchanging** parts first: instructions, then reference documents, then examples.
2. Put the **changing** part (the user's question) **last**.

3. Don't insert timestamps or random IDs at the top of prompts. They break the match.
4. Send repeat requests within the cache lifetime, which is typically minutes.

Worked example (Claude Sonnet 5.5): a 50,000-token handbook plus a 200-token question, asked 100 times within a few minutes.

TEXT

No caching:	$100 \times 50,200 \times \$2.00/1M$	= \$10.04 input
With caching:	1 write $\times 50,000 \times \$2.50/1M$ (1.25 $\times$)	= \$0.125
	99 reads $\times 50,000 \times \$0.10/1M$	= \$0.495
	100 $\times 200$ uncached $\times \$2.00/1M$	= \$0.04
	Total input	$\approx \$0.66$ (about 93% less)

5.2 Batch processing

If you don't need answers right away, submit requests as a **batch**. Results usually arrive within 24 hours (often much sooner), at **half price** on OpenAI, Anthropic and Google. Batch also has its own rate limits, so it doesn't slow down your normal traffic.

Good batch jobs: categorizing a year of transactions, summarizing an archive, tagging product listings, translating a catalog, and running evaluations.

CLAUDE_BATCH.PY

```
import anthropic
from dotenv import load_dotenv

load_dotenv()
client = anthropic.Anthropic()
docs = {"doc-1": "First document text...", "doc-2": "Second document text..."}

batch = client.messages.batches.create(requests=[
    {"custom_id": key,
     "params": {"model": "claude-haiku-5-5", "max_tokens": 300,
               "messages": [{"role": "user", "content": f"Summarize in 2 sentences:\n\n{text}"]}}}
    for key, text in docs.items()
])
print(batch.id, batch.processing_status)
# Later: client.messages.batches.retrieve(batch.id), then
# for result in client.messages.batches.results(batch.id): ...
```

5.3 Flex and other service tiers

OpenAI's **Flex** processing is half price, in exchange for slower responses and occasional “resource unavailable” errors. It's good for background jobs that need answers sooner than a batch. **Fast** or priority modes cost more for lower latency. Google lists **Flex** and **Priority** variants too. Check each provider's current options.

5.4 Discounts stack

Batch and caching discounts can be **combined**, and both providers' docs say so. A batch job that reuses a cached prefix pays half of the already-discounted cached rate.

Estimating a Project's Cost Before You Start

Never start a big job without a cost estimate. It takes ten minutes and prevents nasty surprises.

IN THIS CHAPTER

- The four numbers you need
- A copy-and-run cost calculator
- A budgeting worksheet

6.1 The four numbers

NUMBER	HOW TO GET IT
Items	How many documents, rows or messages
Input tokens per item	Run 5–10 real samples and average the <code>usage</code> input numbers. Include instructions and history
Output tokens per item	Same, from the <code>usage</code> output numbers. Or set <code>max_tokens</code> , which gives a worst case
Prices	From the official pricing page today

6.2 The calculator

ESTIMATE.PY

```

PRICES = { # $ per 1M tokens: (input, output). Checked Oct 16, 2026. Re-check before use!
    "gpt-6-luna": (0.10, 0.50),
    "gpt-6.1-sol": (2.00, 10.00),
    "claude-haiku-5-5": (0.10, 0.50),
    "claude-sonnet-5-5": (2.00, 10.00),
    "claude-opus-5-5": (4.00, 20.00),
    "gemini-3.1-flash-lite": (0.25, 1.50),
    "gemini-3.8-flash": (0.75, 3.75),
    "grok-4.3": (1.25, 2.50),
    "grok-4.7": (2.00, 6.00),
}

def estimate(items, in_tok, out_tok, model, batch=False, safety=1.3):
    p_in, p_out = PRICES[model]
    cost = items * (in_tok * p_in + out_tok * p_out) / 1_000_000
    if batch:
        cost *= 0.5
    return cost * safety # 30% buffer for retries and longer-than-expected items

for m in PRICES:
    print(f"{m:24s} ${estimate(5_000, 1_800, 250, m):8.2f}  batch: ${estimate(5_000, 1_800, 250, m, batch=True):8.2f}")

```

6.3 Budget worksheet

- ☐ ☐ Task described in one sentence
- ☐ ☐ Number of items: _____
- ☐ ☐ Average input tokens (measured): _____ and output tokens: _____
- ☐ ☐ Cheapest model that passed a 20-item quality test: _____
- ☐ ☐ Estimated cost, with a 30% buffer: \$_____
- ☐ ☐ Batch possible? Yes / No. Caching possible? Yes / No
- ☐ ☐ Spending cap set at about 2× the estimate: \$_____
- ☐ ☐ Stop condition in the code (maximum items or maximum dollars)

6.4 Put a dollar limit in the code itself

PYTHON

```
MAX_DOLLARS = 5.00
spent = 0.0
for item in items:
    if spent >= MAX_DOLLARS:
        print("Budget reached, stopping.")
        break
    response = call_model(item)
    spent += (response.usage.input_tokens * P_IN + response.usage.output_tokens * P_OUT) / 1e6
```

TIP

Account-level caps protect your card. A **code-level** cap protects you from a single runaway loop. Use both.

Twelve Ways to Cut Your Bill

#	TECHNIQUE	TYPICAL SAVING
1	Use the smallest model that passes your quality test. Test cheap first	5–100×
2	Batch anything that can wait	50%
3	Cache long repeated prefixes (instructions, documents)	Up to ~90% of input
4	Trim input. Send only the relevant section, not the whole file	Proportional
5	Cap output. Set <code>max_tokens</code> , and ask for “under 100 words” or “JSON only”	Proportional
6	Lower reasoning effort for simple tasks on thinking models	Often large on output
7	Reset chat history. Summarize old turns instead of resending them all	Grows with chat length
8	Group items. Categorize 40 rows per request instead of 1	Up to 40× fewer calls
9	Deduplicate. Process unique items only, and reuse answers	Depends on the data
10	Cache results yourself. Store answers in a file or database keyed by input, and never pay twice	100% on repeats
11	Avoid long-context surcharges. Stay under the thresholds (e.g. 200K, 272K, or Haiku 5.5’s 100K)	2× or more
12	Route. A cheap model handles easy cases, and only hard cases escalate to an expensive one	50–90%

7.1 Routing example

ROUTER.PY

```
from ai import ask  # from Vol. 14's helper

def answer(question):
    draft = ask(f"Answer briefly. If you're not confident, reply exactly UNSURE.\n\n{question}",
                provider="gemini", model="gemini-3.1-flash-lite")
    if "UNSURE" in draft:
        return ask(question, provider="claude", model="claude-sonnet-5-5")
    return draft
```

Rate Limits, Errors, and Reliability

Spending isn't the only limit. Every provider also caps **how fast** you can send requests.

IN THIS CHAPTER

- RPM, TPM and daily limits
- The error codes you'll see
- Retrying correctly

8.1 The limits

ABBREVIATION	MEANING
RPM	Requests per minute
TPM / ITPM / OTPM	Tokens per minute (total / input / output)
RPD	Requests per day (Gemini's daily quotas reset at midnight Pacific)
Spend-rate limit	Gemini also caps spending per 10-minute window on Tiers 1–3 (\$10/\$50/\$200)

Limits are set per **organization or project**, not per key, and rise automatically as you spend more and build history. Your current limits are on each console's limits page: OpenAI **Settings** → **Organization** → **Limits**, Anthropic platform.claude.com/usage/limits, Gemini in **AI Studio**.

NOTE

On Anthropic, **cached** input tokens don't count toward input-token rate limits for most models. Caching raises your effective throughput as well as cutting cost.

8.2 Errors and what to do

CODE	MEANING	ACTION
400	Bad request (wrong parameter, too long, or, on Anthropic, your own spend limit reached)	Read the message. Fix the request, or raise your limit
401	Bad or missing key	Check <code>.env</code> and that <code>load_dotenv()</code> ran
403	No permission for that model or region	Check account access and region
404	Model not found (often retired or misspelled)	Use a current model ID
429	Rate limited, or out of quota/credit	Wait and retry if it's temporary. Don't retry billing or quota errors
500 / 503	Server error or overload	Retry with back-off

8.3 Retry correctly

- 1 Let the SDK retry first.**
The official SDKs automatically retry eligible 429 and 5xx errors a couple of times.
- 2 Honor `Retry-After`.**
If the response includes this header, wait at least that many seconds.
- 3 Back off exponentially, with jitter.**
Wait 1 s, 2 s, 4 s... plus a small random delay.
- 4 Cap attempts and total time.**
Then save the failed item for later instead of looping forever.
- 5 Never auto-retry billing errors.**
A spend cap or empty balance won't fix itself in seconds. Anthropic's spend-cap 429 deliberately has no `retry-after` header.

Beyond Text: Tools, Images, Audio, and Agents

Text tokens are only part of the bill. Tools and media have their own prices.

9.1 Tool and feature fees (October 16, 2026)

FEATURE	OPENAI	ANTHROPIC	GOOGLE	XAI
Web search	\$10 / 1,000 calls + content tokens	\$10 / 1,000 searches + content tokens	5,000 free/month on Gemini 3+, then \$14 / 1,000	Server-side search tools are available; see the xAI pricing docs
Code execution	Containers from \$0.03 per 20-minute session (1 GB)	1,550 free hours/month per organization, then \$0.05/hour (free when used with web search/fetch)	Billed as tokens	—
File search / retrieval	\$0.10/GB per day storage (1 GB free) + \$2.50 / 1,000 calls	—	Embeddings at \$0.15 / 1M tokens	—
Web fetch	—	No extra fee beyond tokens	URL context billed as input tokens	—

9.2 Images, audio and video (examples)

ITEM	PRICE (OCT 16, 2026)	SOURCE
xAI grok-imagine-image	\$0.02 per image	docs.x.ai
xAI grok-imagine-image-quality	\$0.05 per image	docs.x.ai
Google gemini-3.1-flash-image (Nano Banana 2), 1K image	≈ \$0.067 per image	ai.google.dev
Google Veo 3.1 Fast, 720p video with audio	\$0.10 per second	ai.google.dev
OpenAI gpt-4o-mini-transcribe	≈ \$0.003 per minute	developers.openai.com
OpenAI tts-1 (text to speech)	\$15 per 1M characters	developers.openai.com
xAI text to speech	\$15 per 1M characters	docs.x.ai

9.3 Agents cost more than you think

An **agent** loops: think → call a tool → read the result → think again. Each loop re-sends the growing context. A single “research this topic” agent task can run 20–100 model calls.

- ☐ ☐ Set a maximum number of steps (tool calls) per task
- ☐ ☐ Set a per-task dollar cap in your code
- ☐ ☐ Use a cheap model for routine steps and an expensive one only for planning or final review
- ☐ ☐ Turn on prompt caching, since agent prompts repeat heavily
- ☐ ☐ Log every call’s token usage, so you can see which step is expensive

Anthropic’s own worked example for its Managed Agents service: a one-hour Opus 5.5 session with 50,000 input and 15,000 output tokens costs about **\$0.58**, including \$0.08 of session runtime. With caching, it drops to about **\$0.43**.

Subscription or API? Doing the Math

For **personal, interactive** use, a flat subscription is usually the better deal. For **automated, high-volume or embedded** use, you need the API anyway. Here's how to compare honestly.

10.1 Break-even thinking

1

Estimate your monthly tokens.

Example: 20 chats a day $\times$ 30 days $\times$ (3,000 input + 800 output tokens) = 1.8M input + 0.48M output tokens.

2

Price it on the API.

On a \$2 / \$10 model: $1.8 \times \$2 + 0.48 \times \$10 = \text{\$8.40 a month.}$

3

Compare.

Look at the subscription price on the official pricing page. Remember subscriptions also include apps, memory, voice, image generation, file tools and more, with no code.

4

Remember the chat-history effect.

Long chats re-send everything, so heavy users' API costs can be far higher than this simple estimate.

NOTE

This manual deliberately doesn't quote subscription prices. They vary by country and change often, and providers increasingly offer several tiers. Check chatgpt.com/pricing, claude.com/pricing, gemini.google/subscriptions and grok.com for the current ones.

10.2 Free ways to experiment

OPTION	WHAT'S FREE	CATCH
Google AI Studio and the Gemini API free tier	Some models, at limited rates	Free-tier content is used to improve Google's products
Anthropic new-account credit	A small amount of starter credit	It's small, and it can expire
OpenAI Free usage tier	Listed in OpenAI's rate-limit tiers (eligible geographies)	Very limited
Local models (Ollama, LM Studio)	Unlimited, offline	You need capable hardware (Vol. 11)

Cost Recipes for Common Projects

Estimates use October 16, 2026 prices and assume no caching or batch discounts unless stated. Treat them as orders of magnitude, then measure your own.

PROJECT	VOLUME	MODEL	ESTIMATED COST
Categorize a year of bank transactions (grouped 40 per request)	2,400 rows → 60 requests × 3,000 in / 1,500 out	gpt-6-luna	≈ \$0.06
Summarize 500 PDFs of 10 pages each	500 × 5,000 in / 300 out	claude-haiku-5-5, batch	≈ \$0.16
Same, higher quality	Same	claude-sonnet-5-5, batch	≈ \$3.25
Customer-support bot, 1,000 chats a month	6 turns, avg 4,000 in / 300 out per turn	gemini-3.8-flash	≈ \$24.75
Same, with the system prompt cached	~70% of input cached	gemini-3.8-flash	roughly \$10
Draft 100 blog posts	1,000 in / 2,000 out each	grok-4.7	≈ \$1.40
Code assistant for one developer	3M in / 300K out per day × 22 days	claude-sonnet-5-5	≈ \$198, much less with caching

The arithmetic for one row, so you can check the others: support bot = 1,000 chats × 6 turns × (4,000 × \$0.75 + 300 × \$3.75) / 1M = 6,000 × (\$0.003 + \$0.001125) = **\$24.75**.

Monitoring and Alerts

Spend controls only help if you look at them.

12.1 Weekly 5-minute check

- ☐ ☐ Open each provider's **usage** page and compare it to last week
- ☐ ☐ Look for unknown keys, projects or models in the breakdown
- ☐ ☐ Check your balance and auto top-up settings (xAI, Anthropic, OpenAI prepaid)
- ☐ ☐ Confirm your alerts still go to an inbox you read
- ☐ ☐ Delete keys you no longer use

12.2 Log usage from your own code

USAGE_LOG.PY

```
import csv, datetime

def log_usage(provider, model, input_tokens, output_tokens, cost, path="usage_log.csv"):
    with open(path, "a", newline="", encoding="utf-8") as f:
        csv.writer(f).writerow([datetime.datetime.now().isoformat(timespec="seconds"),
                                provider, model, input_tokens, output_tokens, f"{cost:.6f}"])
```

Open `usage_log.csv` in a spreadsheet and make a pivot table by model and day. You'll spot the expensive step immediately.

Troubleshooting and FAQ

PROBLEM	LIKELY CAUSE	FIX
Bill much higher than expected	Chat history growing; thinking tokens; a loop; a leaked key	Check usage by model and day; cap output and steps; rotate keys
"Insufficient quota" or 429 on first use	No credit loaded or billing not set up	Add credit or billing; wait a few minutes
Same prompt costs different amounts	Different output length, or thinking effort	Cap output; set effort; measure several runs
Cache never hits	Changing text at the start of the prompt	Move variable parts to the end; keep the prefix identical
Model "not found"	Retired or renamed model	Use a current ID from the models page
Gemini works, then daily errors	Free-tier requests-per-day quota	Wait for midnight Pacific, or enable billing
Charges continue after you stopped	Another app or person using the key	Revoke the key; check projects and members

Why does output cost more than input? Generating each output token takes the model much more work than reading an input token, so providers price output 3–10× higher.

Are cheap models really good enough? For extraction, classification, summarizing and rewriting, often yes. Test 20 real examples side by side before deciding.

Do prices only go down? Not always. This edition records announced *increases*: Gemini 4 Argon and Gemini 3.8 Flash both have published price rises. Promotional prices end. Re-check before every large job.

Is my data used for training? Policies differ by provider and tier. For example, Google says Gemini API free-tier content is used to improve its products, while paid-tier content is not. Read each provider's data-use policy, and see Vol. 18, *AI Privacy & Security*.

Choosing a Model: A 30-Minute Test

Don't choose a model from a leaderboard. Run your own small test on your real task. It's the only reliable way to find the cheapest model that's good enough.

IN THIS CHAPTER

- Building a 20-example test set
- Running it against several models
- Scoring quality against cost

14.1 Build a test set

- 1 Collect 20 real examples** of your task: real emails, rows or documents, with private details removed.
- 2 Write the ideal answer** for each one, or at least the key facts a good answer must include.
- 3 Include hard cases.**
Add 3–4 tricky ones: ambiguous, very long, badly formatted, or in another language.
- 4 Save it as** `tests.csv` with the columns `id`, `input` and `must_include` (keywords separated by semicolons).

14.2 Run it against several models

MODEL_TEST.PY

```
import csv, time
from ai import ask # Vol. 14 helper

CANDIDATES = [("local", None), ("gemini", "gemini-3.1-flash-lite"),
              ("openai", "gpt-6-luna"), ("claude", "claude-haiku-5-5"),
              ("claude", "claude-sonnet-5-5")]
PROMPT = "Summarize this customer email in one sentence and state the requested action:\n\n{input}"

with open("tests.csv", encoding="utf-8") as f:
    tests = list(csv.DictReader(f))

for provider, model in CANDIDATES:
    passed, start = 0, time.time()
    for t in tests:
        reply = ask(PROMPT.format(input=t["input"]), provider=provider, model=model).lower()
        keywords = [k.strip().lower() for k in t["must_include"].split(";") if k.strip()]
        if all(k in reply for k in keywords):
            passed += 1
    print(f"{provider:7s} {model or 'default':24s} {passed}/{len(tests)} passed {time.time() - start:5.1f}s")
```

14.3 Decide

RESULT	DECISION
The cheapest model passes 19–20 of 20	Use it. Re-test monthly
The cheapest model passes 15–18	Look at the failures. Better instructions or examples often fix them. If not, try routing (cheap first, escalate hard cases)
Only expensive models pass	Use the expensive model, but cut cost with caching, batch and shorter inputs
Nothing passes	The task needs redesign: split it into steps, add reference data, or keep a human in the loop

TIP

Keyword checks are crude but fast. For subjective tasks, also read the 20 outputs side by side yourself. Ten minutes of reading beats any benchmark.

Embeddings: The Cheap Way to Search Your Documents

An **embedding** turns text into a list of numbers that captures its meaning. Texts with similar meanings get similar numbers. This lets you **search by meaning** across thousands of documents and send only the relevant few to an expensive model. That pattern is called **retrieval-augmented generation (RAG)**.

15.1 Prices (October 16, 2026)

MODEL	PRICE PER 1M INPUT TOKENS
OpenAI text-embedding-3-small	\$0.02
OpenAI text-embedding-3-large	\$0.13
Google gemini-embedding-2 (text)	\$0.20, and free on the free tier

Embedding a 300-page book (about 150,000 tokens) with text-embedding-3-small costs about **\$0.003**. You pay once, store the numbers, and search them for free on your own computer.

15.2 The RAG pattern in five steps

- 1 **Split** documents into chunks of about 500–1,000 tokens.
- 2 **Embed** every chunk once, and save the vectors to a file or a vector database.
- 3 **Embed the question** when a user asks one.
- 4 **Find the closest chunks** with cosine similarity, and keep the top 3–5.
- 5 **Ask the model** to answer using *only* those chunks, citing them.

MINI_RAG.PY

```
from openai import OpenAI
from dotenv import load_dotenv
import math

load_dotenv()
client = OpenAI()

def embed(texts):
    r = client.embeddings.create(model="text-embedding-3-small", input=texts)
    return [d.embedding for d in r.data]

def cosine(a, b):
    dot = sum(x * y for x, y in zip(a, b))
    return dot / (math.sqrt(sum(x * x for x in a)) * math.sqrt(sum(y * y for y in b)))

chunks = ["Our refund window is 30 days from delivery.",
          "Shipping to Canada takes 5-7 business days.",
          "Gift cards cannot be refunded."]
vectors = embed(chunks)

question = "Can I return a gift card?"
q = embed([question])[0]
best = sorted(zip(chunks, vectors), key=lambda cv: cosine(q, cv[1]), reverse=True)[:2]
context = "\n".join(c for c, _ in best)
answer = client.responses.create(model="gpt-6-luna",
                                input=f"Answer using only this context:\n{context}\n\nQuestion: {question}")
print(answer.output_text)
```

Why it saves money: instead of sending a 200,000-token manual with every question, you send roughly 2,000 tokens of relevant chunks. That's about 99% less input per question.

Raw HTTP: Calling Any API with curl

Every SDK is just a wrapper around web requests. Knowing the raw form helps you debug, and lets you use tools that only accept a URL and headers, such as Zapier, Make and n8n.

OPENAI

```
curl https://api.openai.com/v1/responses \
-H "Authorization: Bearer $OPENAI_API_KEY" \
-H "Content-Type: application/json" \
-d '{"model": "gpt-6-luna", "input": "Say hello in five words."}'
```

ANTHROPIC

```
curl https://api.anthropic.com/v1/messages \
-H "x-api-key: $ANTHROPIC_API_KEY" \
-H "anthropic-version: 2023-06-01" \
-H "content-type: application/json" \
-d '{"model": "claude-haiku-5-5", "max_tokens": 100,
  "messages": [{"role": "user", "content": "Say hello in five words."}]}'
```

GOOGLE GEMINI

```
curl "https://generativelanguage.googleapis.com/v1beta/models/gemini-3.8-flash:generateContent" \
-H "x-goog-api-key: $GEMINI_API_KEY" \
-H "Content-Type: application/json" \
-d '{"contents": [{"parts": [{"text": "Say hello in five words."}]}]}'
```

XAI (OPENAI-COMPATIBLE)

```
curl https://api.x.ai/v1/chat/completions \  
-H "Authorization: Bearer $XAI_API_KEY" \  
-H "Content-Type: application/json" \  
-d '{"model": "grok-4.7", "messages": [{"role": "user", "content": "Say hello in five words."}]}'
```

 **NOTE**

On Windows PowerShell, `curl` may be an alias for a different command. Use `curl.exe`, and replace `$OPENAI_API_KEY` with `$env:OPENAI_API_KEY`. Line-continuation backslashes (`\`) become backticks (```) in PowerShell.

RESPONSE FIELD TO READ	OPENAI RESPONSES	ANTHROPIC	GEMINI	XAI CHAT COMPLETION
The text	<code>output[...].content[...].text</code>	<code>content[0].text</code>	<code>candidates[0].content.parts[0].text</code>	<code>choices[0].message</code>
Token usage	<code>usage</code>	<code>usage</code>	<code>usageMetadata</code>	<code>usage</code>

Using Your Key in No-Code Tools

Automation tools (Vol. 16, *Zapier & Make*, and Vol. 6, *n8n*) and many apps ask you to paste an API key. The same cost rules apply, plus a few extra risks.

- ☐ ☐ Create a **separate key** (and, where possible, a separate project or workspace) for each tool, so its spending is visible and revocable on its own
- ☐ ☐ Set that project's or workspace's spend limit low, e.g. \$10/month, to start
- ☐ ☐ Check whether the tool runs on a schedule or on every new email or row. Triggers can fire thousands of times
- ☐ ☐ Add a filter step before the AI step, so only items that need AI reach it
- ☐ ☐ Prefer the tool's built-in AI credits only if you've compared their price per task with your own key
- ☐ ☐ Remove the key from the tool when you stop using it

Data Use and Terms at a Glance

Price isn't the only thing that differs between tiers. What happens to your data can change too.

QUESTION	WHERE TO LOOK
Is my API data used to train models?	Each provider's API data-usage policy. For example, Google's pricing page states that free-tier Gemini API content is used to improve its products and paid-tier content is not
How long are requests kept?	The provider's data-retention documentation; some offer zero-retention options for eligible customers
Can I keep data in a region?	Data-residency options (OpenAI regional endpoints +10% on newer models; Anthropic <code>inference_geo: "us"</code> at 1.1x; cloud platforms' regional endpoints)
Can I use outputs commercially?	The provider's terms of service and usage policies
Health or financial data?	Use only services and agreements designed for it (for US health data, a Business Associate Agreement). See Vols. 18 and 19

⚠ DANGER

Don't send passwords, full card numbers, government ID numbers or other people's private information to any API unless you've confirmed the provider's terms allow it and you've put the right agreements in place.

Quick Reference

Cost formula

- $\text{Cost} = (\text{input tokens} \times \text{input price} + \text{output tokens} \times \text{output price}) \div 1,000,000$

Rules of thumb

- 1 token $\approx \frac{3}{4}$ word · 1,000 tokens $\approx$ 750 words · 1 page $\approx$ 400–500 tokens
- Output costs 3–10× input · thinking tokens are billed as output
- Batch $\approx$ 50% off · cache reads $\approx$ 2.5–10% of input price (varies by model)

Consoles

- OpenAI: platform.openai.com · Anthropic: platform.claude.com · Google: aistudio.google.com · xAI: console.x.ai

Pricing pages (re-check before budgeting)

- developers.openai.com/api/docs/pricing
- docs.anthropic.com → About Claude → Pricing
- ai.google.dev/gemini-api/docs/pricing
- docs.x.ai/docs/models

Spend controls

- OpenAI: spend alerts and hard spend limits (org or project)
- Anthropic: Billing → Spend limits; per-workspace limits
- Google: Cloud Billing budgets (alerts only); tier caps
- xAI: prepaid credits with a \$0 invoiced limit = hard stop

Before every big job

- Measure 10 samples · estimate with a 30% buffer · set an account cap and a code cap · test the cheapest model first

Glossary

API key	A secret credential that lets software use your API account and incur charges.
Batch API	A way to submit many requests for delayed processing, typically at 50% off.
Cache read (cache hit)	Reusing a stored prefix, charged at a small fraction of the input price.
Cache write	The first time a reusable prompt prefix is stored; sometimes priced above normal input.
Context window	The maximum tokens (input plus output) a model can handle in one request.
Flex processing	A cheaper, slower service tier for non-urgent requests.
Input tokens	Everything you send: prompt, instructions, history, files, tool definitions.
Long-context pricing	Higher per-token rates once a prompt passes a size threshold.
MTok	One million tokens, the unit prices are quoted in.
Output tokens	Everything the model generates, including reasoning tokens.
Prepaid credits	Money loaded in advance; usage stops when it runs out unless top-ups are enabled.
Prompt caching	Storing and reusing a repeated prompt prefix to cut cost and latency.
Rate limit	A cap on requests or tokens per minute or day.
Reasoning (thinking) tokens	Tokens a model generates while working through a problem before answering; billed as output.
Spend limit	A cap you set on monthly spending.
Token	The unit of text models read, write and bill by; about $\frac{3}{4}$ of an English word.
Usage tier	The level your account is on, which sets rate limits and caps; rises with spend and history.