

■ A GUIDE FOR AI FOUNDERS

The pricing economics of AI agents

How to price and bill an AI product so *every* customer stays profitable.



Contents

PART 1

Why agent economics are different

- 01 Same plan, different business
 - 02 Why the SaaS cost model breaks
 - 03 What one agent run costs
 - 04 Cheaper tokens, more tokens per task
 - 05 Lessons from companies that repriced in public
-

PART 2

Where margin leaks

- 06 Seven common margin leaks
 - 07 The heavy-user problem
-

PART 3

Choosing what to charge for

- 08 Pricing models and their margin trade-offs
 - 09 Choosing a unit you can cost
 - 10 Outcome pricing that stays profitable
 - 11 Credits that survive model changes
-

PART 4

Plan structures that protect margin

- 12 Combining prepaid and usage-based parts
 - 13 Guardrails that cap your downside
 - 14 A cost floor under value-based prices
 - 15 Worked plans, with the numbers
-

PART 5

Running pricing as a system

- 16 Attributing provider cost to customers
 - 17 The margin metrics to watch
 - 18 Accounting for prepaid credits
 - 19 What your billing system needs to do
 - 20 Repricing without losing trust
-

BACK MATTER

Summary and next steps

What to take away

About Clocknext

Why this ebook exists

Most pricing advice for software was written for products that cost almost nothing to serve one more time. AI agents work differently, and the pricing has to follow.

A seat added to a project management tool costs its vendor a few cents of hosting. An AI agent is different. Every task it runs calls a model, and every model call has a price. Some tasks call a model once. Others call it dozens of times, use a more capable model, or pull in paid tools along the way.

That changes the job of pricing. You still have to charge in a way your customers understand and accept. You also have to make sure that what each customer pays stays above what that customer costs you to serve, even when two customers on the same plan use your product in very different ways.

This ebook is about that second job. It is written for founders, and for the product, finance, and engineering leads who work with them, at companies building AI agents and AI-native products. It covers why agent economics behave the way they do, where margin leaks, how to choose what to charge for, which plan structures protect margin, and how to run pricing as a system that keeps working as models and customers change.

We drew on investor research, public reporting, provider price lists, and the published pricing changes of AI companies that have been through this in public. Where a number comes from a survey or a press report, we say so. Where an example is illustrative, we label it and show the arithmetic so you can rerun it with your own numbers.

Six things to take away

1. **Margin is a per-customer number.** A blended gross margin can look healthy while your heaviest accounts lose money on every task.

2. **AI gross margins run lower than SaaS margins.** Recent surveys put AI product margins around 50%, against 70 to 80% or more for traditional SaaS.
3. **Falling token prices do not guarantee falling costs.** Customers move to newer models and longer tasks, so cost per task falls far more slowly than price per token.
4. **The unit you charge for has to be one you can cost.** If you cannot say what a unit costs you to deliver, you cannot say whether it is profitable.
5. **Plan structure is where margin is protected.** Prepaid commitments, usage above an allowance, caps, and model-aware credit rates each close a specific leak.
6. **Pricing needs its own operating rhythm.** Provider prices, model mix, and customer behavior change every month, so pricing needs measurement and a change process of its own.

How to read it

Part 1 explains why agent economics behave differently. Part 2 shows where margin leaks. Part 3 covers what to charge for, and Part 4 covers how to structure plans around it. Part 5 is about running pricing once it is live. The back matter has a checklist and a runbook you can use with your team.

Each part stands on its own, so you can start wherever your most pressing problem is. If you are about to launch or change pricing, start with Part 3. If you already have customers and suspect some of them are unprofitable, start with Part 2.

A NOTE ON NUMBERS

Model prices in this ebook are standard list prices as of September 2026, in US dollars per million tokens. They change often. Illustrative examples are labeled as illustrative, and every one shows its inputs so you can substitute your own.

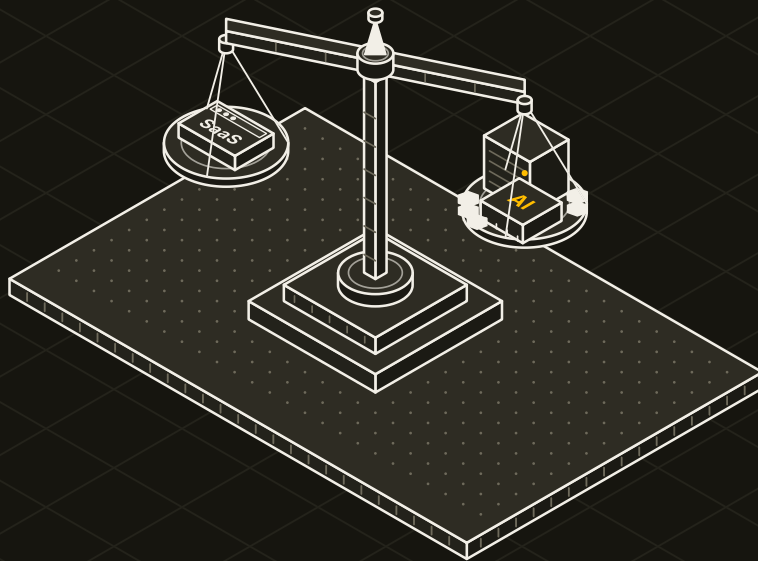
The vocabulary we use

Pricing conversations go wrong when people use the same word for different things. These are the terms used throughout this ebook.

TERM	MEANING
Signal	A metered event from your agent, such as a model call, a tool call, or a completed step. Each signal records which customer it belongs to and what it cost you.
Unit	Something countable you charge for, such as a seat, a document processed, a conversation, or a workflow run.
Credit	A prepaid allowance that customers spend down as they use your product, often at different rates for different models or features.
Outcome	A completed business result you price, such as a resolved support ticket or a qualified lead.
Wallet	A prepaid balance that a customer tops up and draws down.
Provider cost	What model providers and other third parties charge you to serve a customer.
Margin	Price minus provider cost. In this ebook, margin means gross margin on the usage you serve unless we say otherwise.
Plan	The combination of flat fees, units, credits, outcomes, and wallets a customer buys. Each part of a plan can be billed in advance or in arrears.

Why agent economics are different

AI products carry a cost that grows with every request. This part explains where that cost comes from, why it varies so much between customers, and what happened to companies that priced as if it did not.



- 01 Same plan, different business
- 02 Why the SaaS cost model breaks
- 03 What one agent run costs
- 04 Cheaper tokens, more tokens per task
- 05 Lessons from companies that repriced in public

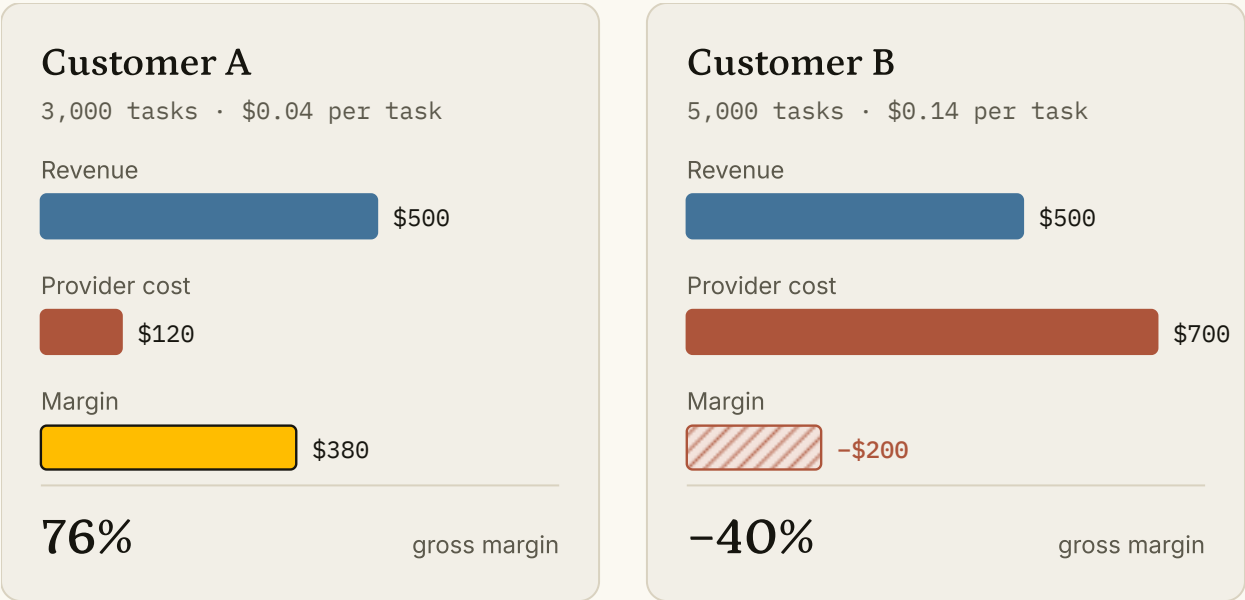
Same plan, different business

Two customers can pay you the same amount every month and leave you with opposite results. Revenue alone cannot tell them apart.

Picture two customers who signed up for the same plan in the same week. Each pays \$500 a month for up to 5,000 agent tasks. On your revenue dashboard they look identical.

Customer A runs 3,000 tasks a month. Most are short, and your router sends them to a small, inexpensive model. Each task costs you about \$0.04 in provider cost. Customer B runs all 5,000 tasks. Their tasks involve long documents, and they have asked for your most capable model. Each task costs you about \$0.14.

FIGURE 1.1
Same plan, two customers



Illustrative. Both customers pay \$500 a month for up to 5,000 tasks. Customer A costs \$120 to serve and Customer B costs \$700. The hatched bar is a loss.

Customer A earns you a 76% margin. Customer B costs you \$200 more each month than they pay. Your blended margin across the two is 18%, which looks thin but survivable. It hides the fact that one account is profitable and the other is a loss.

Now scale that to a book of 200 customers. A few dozen accounts like Customer B can absorb most of the margin the rest of the base produces. Because the heaviest users tend to be your most engaged customers, they are also the accounts your team works hardest to keep and expand.

Why revenue stopped being a proxy for profit

In traditional software, the cost of serving a customer barely changed with how much they used the product. Revenue was a reasonable proxy for profit, and pricing decisions could focus on value and willingness to pay.

In an AI product, cost moves with three things at once: how much a customer uses the product, which model sits behind each request, and the shape of the work itself. Long documents, multi-step tasks, and retries all raise the cost of a single request. None of them show up in your revenue data.

That is the central problem of agent pricing, and the reason this ebook starts with cost. Getting value capture right still matters. It only works when you can also see, customer by customer, what that value costs to deliver.

■ A QUESTION TO ASK YOUR TEAM

Can you name your five least profitable customers today? If the answer depends on a spreadsheet someone updates at month end, the chapters that follow will help you close that gap.

The rest of Part 1 explains why agent costs behave this way. Chapter 2 looks at the cost structure of AI products compared with SaaS. Chapter 3 breaks down what a single agent run costs. Chapter 4 looks at where model prices are heading, and Chapter 5 at what happened to companies that had to reprice in public.

Why the SaaS cost model breaks

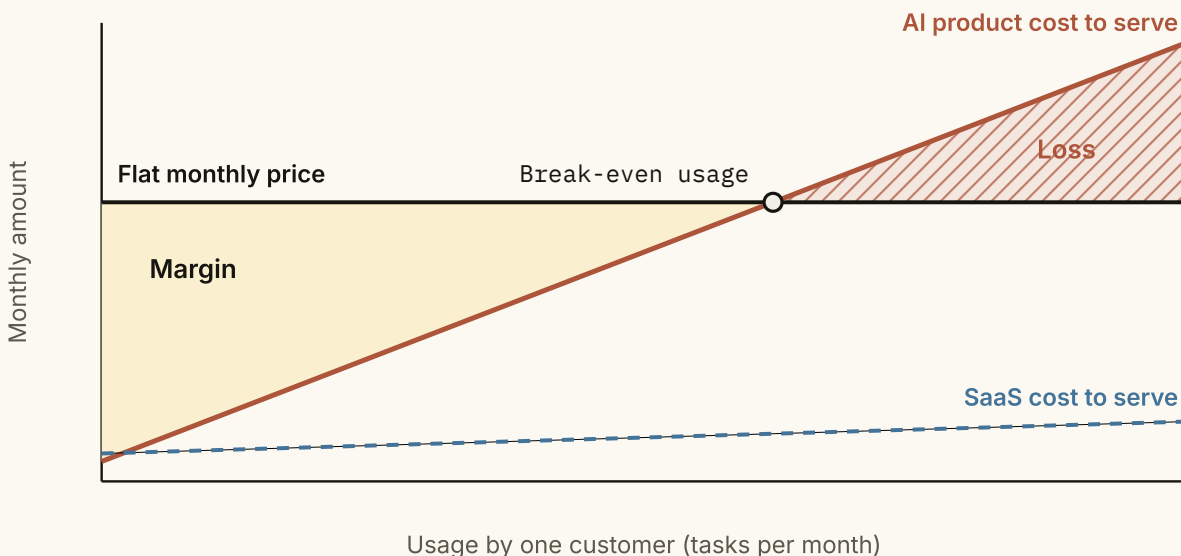
Software businesses were built on near-zero marginal cost. AI products carry a variable cost that grows with every request.

The economics of traditional software are simple to state. Engineering and sales are expensive, and serving one more customer is close to free. Hosting, support, and payment fees make up most of the cost of goods sold. That is why mature SaaS companies report gross margins of 70 to 80% or more, and why investors learned to expect them.

AI products add a cost that SaaS never had. Every request an agent handles runs at least one model call, and many run dozens. That cost belongs in cost of goods sold, and it grows with usage. ICONIQ describes the pattern in its 2026 State of AI report: as AI products scale, the share of cost going to people falls and the share going to inference rises.

FIGURE 2.1

Under a flat price, heavy use turns margin into loss



Conceptual. With a flat monthly price, SaaS cost to serve stays close to flat as usage grows. AI cost to serve rises with every task, so the same plan moves from margin to loss past a break-even level of usage.

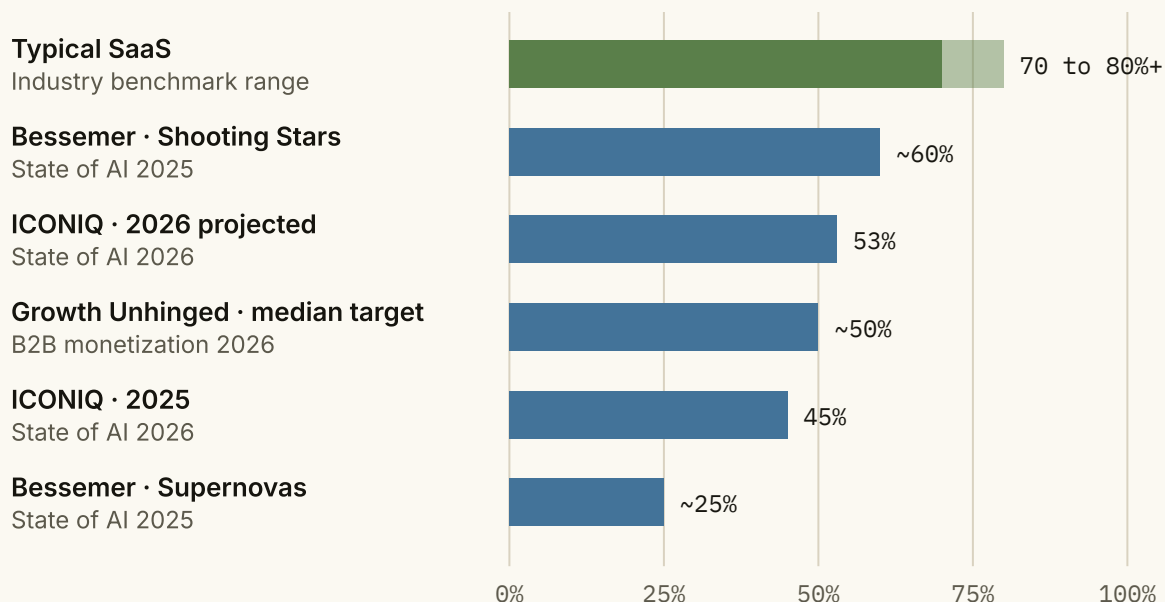
Figure 2.1 shows why this matters for pricing. A flat price works well when cost to serve is flat. When cost rises with usage, a flat price creates a break-even point. Customers below it are profitable. Customers above it are subsidized by everyone else.

What the benchmarks show

The effect shows up clearly in published research on AI company margins.

FIGURE 2.2

Gross margin benchmarks, AI companies and SaaS



Sources: Bessemer Venture Partners, State of AI 2025; ICONIQ, State of AI 2026; Growth Unhinged, The state of B2B monetization in 2026. The Growth Unhinged figure is a target margin, and the ICONIQ 2026 figure is a projection.

Bessemer Venture Partners' State of AI 2025 report studied 20 fast-growing AI startups. The fastest growers, which Bessemer calls Supernovas, averaged about 25% gross margin, often accepting lower margins in exchange for faster distribution. A second group of steadier growers, which it calls Shooting Stars, averaged about 60%.

ICONIQ's 2026 report, based on a survey of more than 300 software executives, puts AI product gross margins at 45% in 2025, with a projection of 53% in 2026 and 59% in 2027. In Growth Unhinged's 2026 survey of 230 B2B software and AI companies, the median company targets a gross margin of about 50% on its AI products, and only 12% target 80% or more.

Two points stand out. Margins are improving: ICONIQ reports that two thirds of builders saw better unit economics per query, helped by managing inference costs and better model routing. And the improvement comes from work. The companies improving their margins measure cost closely and design their products and pricing around it.

Your provider's margin is your cost

Model providers have improved their own margins sharply. SaaStr, citing The Information, reported that OpenAI's compute margin rose from about 35% in January 2024 to about 70% in October 2025, and that Anthropic expected to move from a steeply negative gross margin in 2024 to about 50%.

Those gains belong to the providers. When you build on their models, their list price is your cost. A healthier margin at the model layer does not by itself lower your bill, and it gives you no information about which of your customers are profitable.

Investors read low margins in different ways

Some investors see low AI margins as a stage. Partners at Andreessen Horowitz have argued that low gross margins at a single point in time say little about whether a business model will work over the long run. Others are more cautious. Mayfield's Navin Chaddha has said that the AI companies that win will be the ones that build margin control into the product from the start.

Both views lead to the same practical conclusion. Whether your margin today is a phase or a structural problem depends on facts you can only see at the level of individual customers, features, and models. That is the level the rest of this ebook works at.

■ IN PRACTICE

Classify provider cost correctly from the start. A useful rule from The SaaS Academy: if a customer's action triggered the model call, it belongs in cost of goods sold. Your own team's internal AI use belongs in operating expenses.

SOURCES FOR THIS CHAPTER

1. Bessemer Venture Partners, *The State of AI 2025*, bvp.com/atlas/the-state-of-ai-2025
2. ICONIQ, *State of AI: The Builder's Economy* (2026), iconiq.com/growth/reports/state-of-ai-2026
3. Kyle Poyar, *The state of B2B monetization in 2026*, Growth Unhinged, growthunhinged.com
4. SaaStr, *Have AI gross margins really turned the corner?*, citing The Information, saastr.com
5. PitchBook, *How long can AI agents like Cursor keep burning cash? Investors are divided*, pitchbook.com

6. The SaaS Academy, *How AI changes SaaS gross margin*, thesaasacademy.com

What one agent run costs

To price an agent well, you need a clear picture of what a single run costs you, and of what makes that cost move.

The cost of an agent run is built from a handful of layers. Most teams know the model is the largest one. Fewer have measured how much the other layers add, or how widely the total varies from one run to the next.

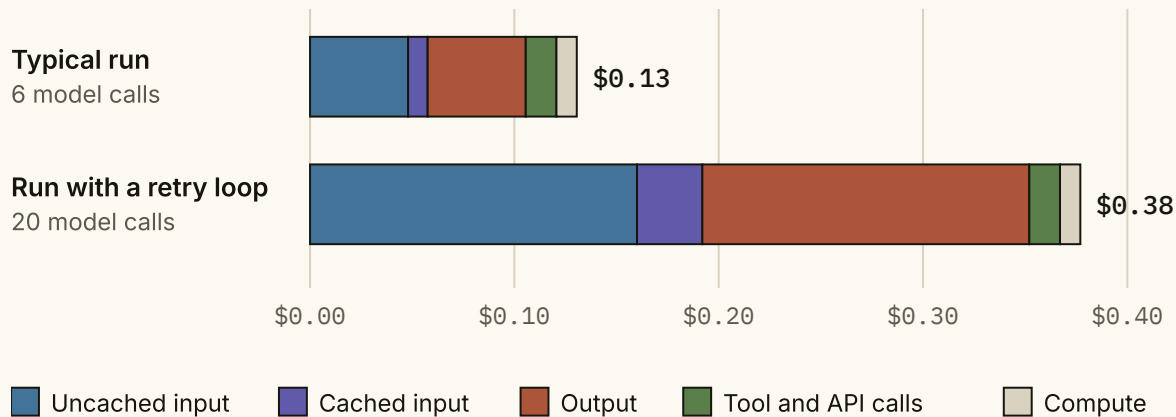
LAYER	WHAT DRIVES IT
Input tokens	Prompt length, system instructions, retrieved documents, conversation history, and tool definitions
Output tokens	Response length and, for reasoning models, the thinking tokens the model generates, which are billed as output
Cached input	Repeated context served from a provider cache, usually at a large discount to standard input
Tool and API calls	Search, data enrichment, code execution, and other paid services the agent calls
Compute and hosting	Sandboxes, vector databases, queues, and runtime that providers bill by time
Retries and loops	Failed attempts, self-correction, and multi-step plans that call the model again

A worked example

Consider a support agent running on Claude Sonnet 5.5, which lists at \$2 per million input tokens and \$10 per million output tokens, with cached input at \$0.20 per million. A typical run makes six model calls. Each call sends 12,000 input tokens, 8,000 of which come from the cache, and generates 800 output tokens. The agent also calls three third-party APIs at an assumed \$0.005 each and uses about a cent of compute.

FIGURE 3.1

The cost of one agent run



Illustrative, using Claude Sonnet 5.5 list prices as of September 2026. Typical run: 24,000 uncached input tokens (\$0.048), 48,000 cached tokens (\$0.0096), 4,800 output tokens (\$0.048), tools (\$0.015), compute (\$0.01). The retry loop repeats the same calls 20 times.

The typical run costs about \$0.13. When the same task goes wrong and the agent retries and corrects itself across 20 model calls, it costs about \$0.38. The customer sees one answer either way. You pay nearly three times as much for the second one.

Output tokens make up more than a third of the typical run's cost. In the retry loop, repeated input and output together account for more than 90%. Loops, more than any single price, are what make agent costs hard to predict.

The drivers that move cost the most

Output is the expensive direction

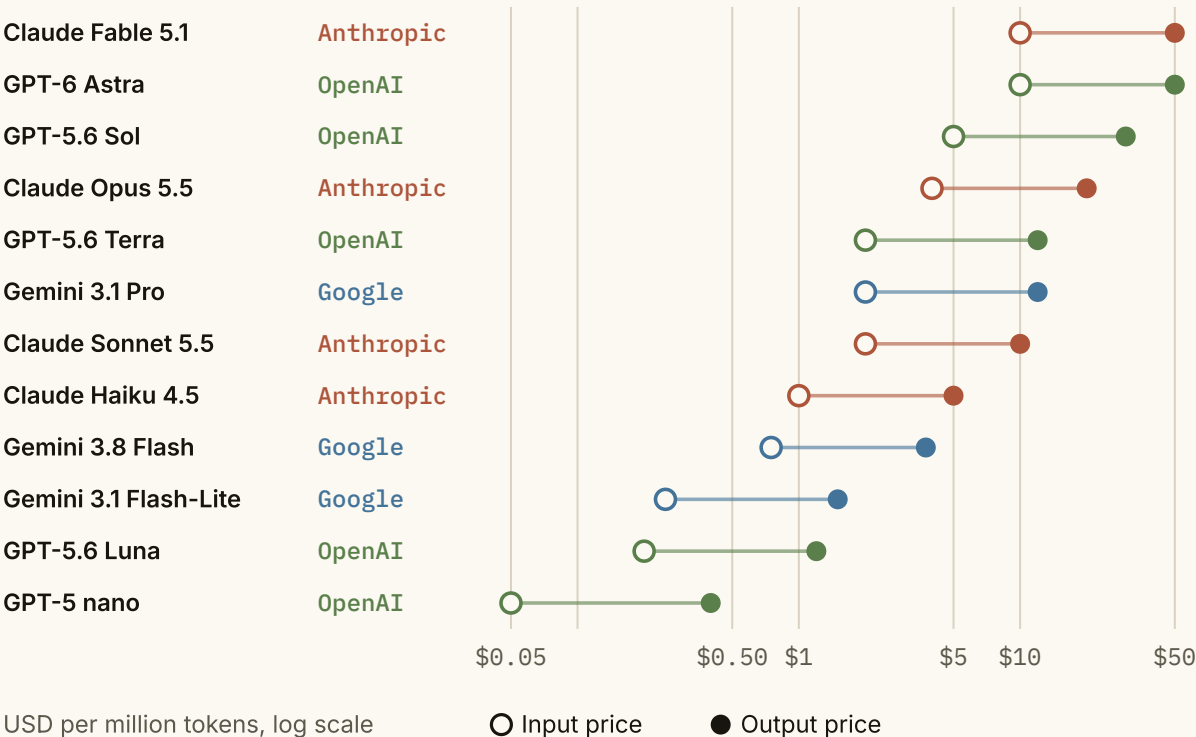
Across the major providers' current price lists, output tokens cost five to six times as much as input tokens. Google's Gemini pricing states that output prices include thinking tokens, so a reasoning model that works through a problem at length is billed for all of that work at the output rate.

Model choice spans a wide range

Current list prices run from \$0.40 per million output tokens for OpenAI's smallest model to \$50 for the most capable models from OpenAI and Anthropic. A task routed to a frontier model can cost dozens of times what the same task costs on a small one.

FIGURE 3.2

Model list prices, input and output



Standard-tier list prices in US dollars per million tokens, as of September 2026, from provider pricing pages and published summaries. Gemini 3.8 Flash is shown at its promotional price, which applies until 31 December 2026. Batch processing is roughly half price at all three providers.

Context and caching change cost by large factors

Anthropic's documentation describes a case where pasting a 1,862-row spreadsheet into a prompt used about 91,000 input tokens per request and answered 6 of 25 questions correctly. Uploading the same file and letting the model analyze it with code answered all 25, at about a twelfth of the cost. Reads from the cache on current Claude models cost 10% of the standard input price.

Tools add overhead before they do anything

Declaring tools costs tokens on every request, whether or not the agent uses them. Anthropic documents that its computer-use toolset adds about 4,500 input tokens to each request that includes it.

Long context can carry a surcharge

On Gemini 3.1 Pro, the input price doubles from \$2 to \$4 per million tokens once a prompt passes 200,000 tokens. OpenAI applies a similar premium to its longest requests.

Loops are the hardest cost to predict

ICONIQ's 2026 report describes a workflow planned at \$0.10 per run whose cost rose above \$1.50 once agents began retrying and correcting their own work. Retries rarely show up in a demo, and they are common in production.

Runtime is becoming a cost line of its own

Some providers now bill agent infrastructure by time. Anthropic's managed agents, for example, bill session runtime measured to the millisecond. Compute time now sits alongside tokens as a provider cost you need to measure.

■ WHAT TO RECORD ON EVERY SIGNAL

Tokens in and out, cached tokens, the model used, tool calls, and retries, each tagged with the customer, the agent, and the feature that triggered it. You will need all of them to attribute cost in Part 5.

SOURCES FOR THIS CHAPTER

1. Anthropic, Claude Platform documentation: pricing, and optimizing for cost and intelligence, platform.claude.com/docs
2. Google, Gemini Developer API pricing, ai.google.dev/gemini-api/docs/pricing
3. OpenAI, API pricing, developers.openai.com/api/docs/pricing; Morpheus, *OpenAI API pricing (2026)*, morphllm.com
4. BenchLM and CloudZero, 2026 model price summaries, benchlm.ai and cloudzero.com
5. ICONIQ, *State of AI: The Builder's Economy (2026)*, iconiq.com

Cheaper tokens, more tokens per task

Model prices are falling quickly. The cost of the work your customers ask for is falling much more slowly.

If model prices keep falling, the cost problem might seem likely to solve itself. Prices are falling fast. Andreessen Horowitz estimated in 2024 that, for a model of equivalent performance, cost falls about tenfold each year. Its example was a thousandfold drop in three years, from \$60 to \$0.06 per million tokens for the same benchmark score.

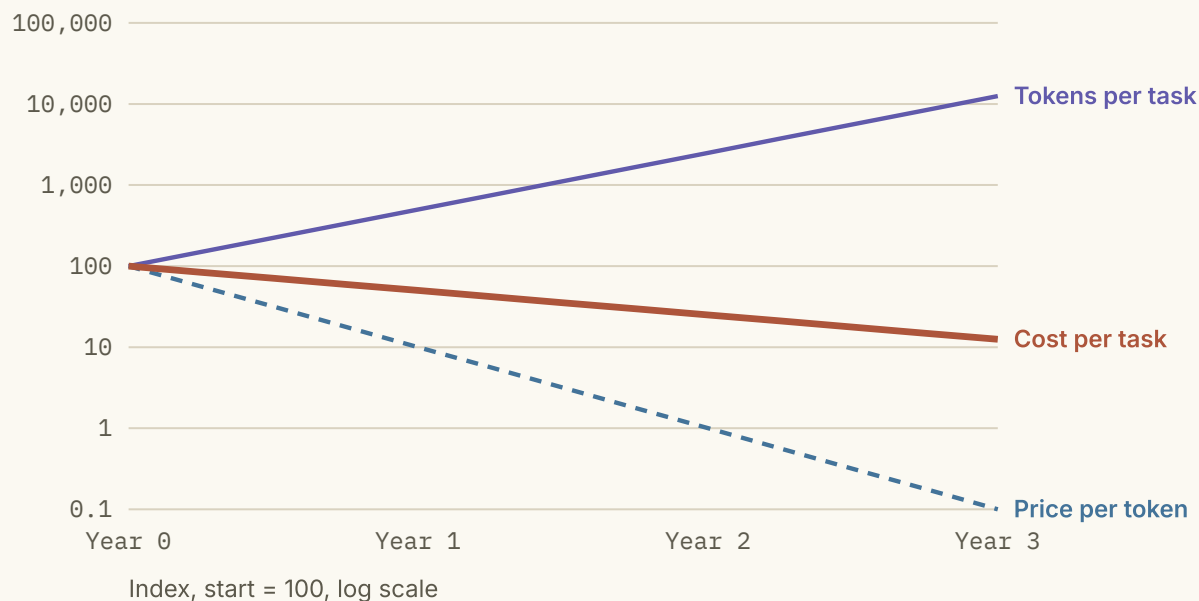
Epoch AI's analysis from September 2026 reaches a similar conclusion. It estimates that the cost of reaching a given level of performance has fallen about 47% per quarter since 2023, or about 13 times per year. The decline is steepest when a capability first becomes state of the art, at about 75 times per year, and slows to under 5 times per year two years later.

The catch is in equivalent performance

Your customers rarely want last year's performance. They want the newest model, longer context, and agents that work through multi-step tasks on their own. Each of those raises the number of tokens a task consumes. SaaStr has described the result as a treadmill: the price of each token keeps dropping while the number of tokens per task keeps climbing.

FIGURE 4.1

Price per token and cost per task move at different speeds



Illustrative. Price per token falls tenfold a year, in line with the Andreessen Horowitz estimate. Tokens per task rise fivefold a year, an assumption chosen for illustration. Cost per task still falls, by half each year, far more slowly than the price of a token.

The exact rates vary by product. SaaStr reports that token consumption per task rose between 10 and 100 times from December 2023 as agentic workflows spread, a range that shows how much it depends on what your agent does. The pattern is consistent: your cost per task falls much more slowly than the headline price per token, and it can rise when your customers adopt a new model.

Provider prices move in both directions

Prices do not only fall. OpenAI's GPT-5.5, released in April 2026, doubled per-token prices compared with GPT-5.4, from \$2.50 and \$15 per million input and output tokens to \$5 and \$30. In July 2026, OpenAI cut prices on two of its GPT-5.6 models by 20% and 80%. Over a single year, your largest cost line moved sharply in both directions.

A model upgrade can raise your costs without any change to the price list. According to an analysis by Finout, the tokenizer introduced with Anthropic's Opus 4.7 can produce up to 35% more tokens for the same input text. The per-token price stayed the same, and the cost of each request went up.

Promotional prices come with an end date

Google lists Gemini 3.8 Flash at \$0.75 and \$3.75 per million input and output tokens through 31 December 2026, rising to \$1.50 and \$7.50 afterwards. OpenAI has run promotional pricing on some GPT-5.6 models as well. If you set a price or a credit rate using a promotional rate, you have already scheduled a cut to your margin.

CHANGE	WHAT HAPPENS TO YOUR COST	EXAMPLE
New, pricier model	Cost per request rises if customers move to it	GPT-5.5 doubled per-token prices over GPT-5.4
Price cut	Cost falls for requests on that model	GPT-5.6 Luna cut 80% in July 2026
Tokenizer change	More tokens for the same text at the same price	Opus 4.7, up to 35% more tokens
Promotion ends	Cost doubles on a scheduled date	Gemini 3.8 Flash after 31 December 2026
More reasoning	More output tokens per task	Reasoning models bill thinking as output

 PLAN FOR MOVEMENT

Price against a cost you expect to hold for the length of a contract, and treat today's lowest rates as upside. Recheck your cost assumptions whenever a provider changes prices or you change a default model.

1. Andreessen Horowitz, *Welcome to LLMflation* (November 2024), a16z.com
2. Epoch AI, *The plunging price of thought* (September 2026), epoch.ai; *LLM inference prices have fallen rapidly but unequally across tasks* (March 2025)
3. SaaStr, *Have AI gross margins really turned the corner?*, saastr.com
4. Apidog, *GPT-5.5 pricing*, apidog.com; BenchLM, *OpenAI API pricing (September 2026)*, benchlm.ai
5. Finout, *Anthropic API pricing in 2026*, finout.io
6. Google, Gemini Developer API pricing, ai.google.dev

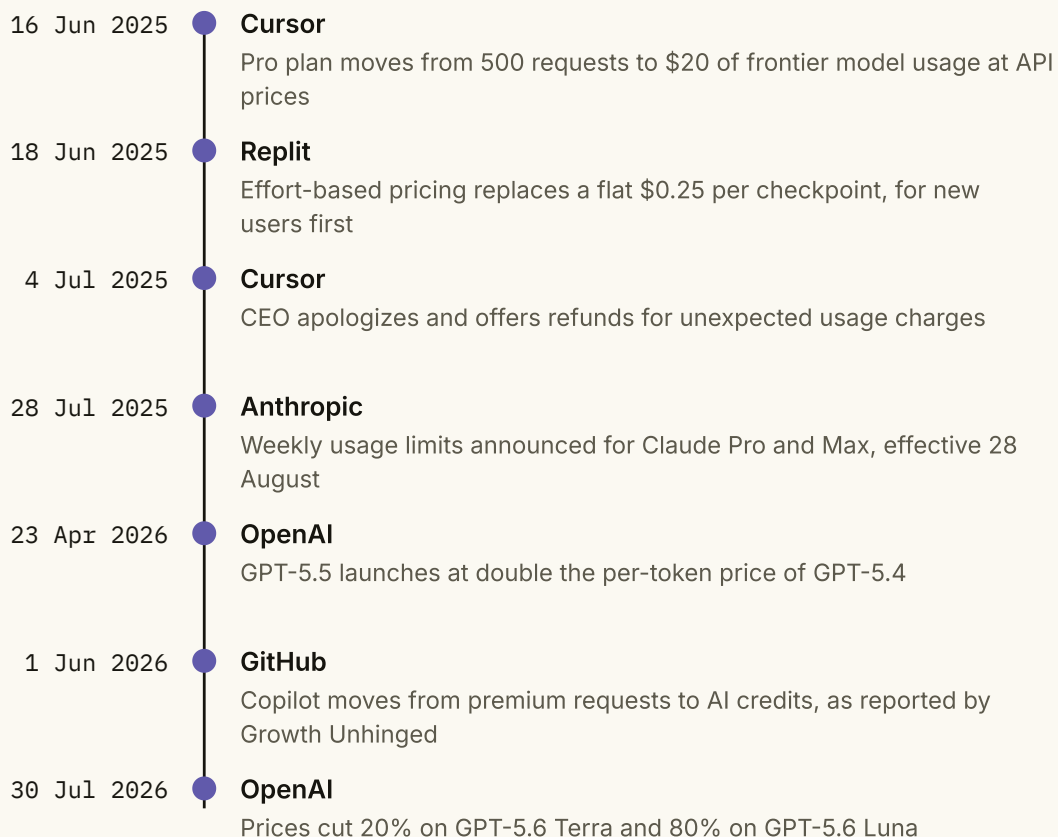
Lessons from companies that repriced in public

Several well-known AI products changed their pricing in the past 18 months. Each was responding to the same problem.

In each case below, a small share of customers cost far more to serve than they paid. The companies responded in different ways, and each response carries a lesson you can apply before your own customers force the issue.

FIGURE 5.1

Public AI pricing changes, June 2025 to July 2026



Dates from company announcements and press coverage. The GitHub Copilot date is as reported by Growth Unhinged.

Cursor moved from requests to dollars of usage

In June 2025, Cursor changed its Pro plan from 500 requests a month to \$20 of frontier model usage per month at API prices, with usage beyond that billed at cost. Many users were surprised by the charges that followed. In early July, the company's CEO apologized and offered refunds for unexpected usage between 16 June and 4 July.

Reporting since then suggests why the change was needed. PitchBook reported that investors described the company as operating with negative gross margins. In April 2026, TechCrunch reported that Cursor had reached positive gross margins on sales to large enterprises while still losing money on individual developer accounts.

■ LESSON

A generous flat allowance becomes a cost pass-through once heavy users arrive. Customers accept that change far more readily when you explain it before it reaches their bill.

Anthropic added weekly limits to its subscriptions

On 28 July 2025, Anthropic announced weekly usage limits for its Claude Pro and Max subscriptions, effective 28 August. The company estimated the limits would affect fewer than 5% of subscribers, and said one user had consumed tens of thousands of dollars of model usage on a \$200 plan. Max subscribers can buy additional usage at API rates.

■ LESSON

A flat price needs a hard limit behind it. The heavy tail is small in number and large in cost, and a limit that affects 5% of customers can protect the economics of the other 95%.

GitHub Copilot turned seats into allowances

GitHub introduced premium requests for Copilot, with model multipliers that count heavier models as more than one request, overage at \$0.04 per request, and budgets that default to zero so overage is refused unless an administrator enables it. Growth Unhinged reports that Copilot moved to AI credits from 1 June 2026, and quotes Microsoft's CEO describing seats as becoming an entitlement to a certain amount of consumption.

■ LESSON

Seat pricing can survive in an AI product. Each seat needs a defined allowance, and the allowance needs rates that reflect what each model costs you.

Replit priced by effort

In June and July 2025, Replit replaced a flat \$0.25 per checkpoint with effort-based pricing, where the price of an agent task reflects the work it takes. Replit gave affected users \$10 in credits. Some users reported being charged for attempts that failed or looped.

■ LESSON

Pricing that tracks cost protects your margin and moves unpredictability onto your customer. That trade only works with spend controls and clear usage visibility on the customer's side.

Salesforce sells Agentforce four ways

Salesforce offers Agentforce through four models at once: \$2 per conversation, Flex Credits at \$500 per 100,000 credits, a per-user add-on at \$125 a month, and a flat enterprise agreement.

■ LESSON

Large buyers want to choose their own balance of predictability and cost alignment. Offering more than one path lets them do that, provided every path is priced above its cost.

What the cases have in common

Each company started with pricing that was simple for the customer and assumed usage would stay within a comfortable range. Each found a minority of heavy users whose cost exceeded their revenue. Each moved toward pricing that ties what a customer pays more closely to what that customer costs.

The changes that went best shared three traits. They were explained before they took effect, they came with controls customers could use, and they kept a simple way to get started. The ones that caused friction surprised customers on an invoice.

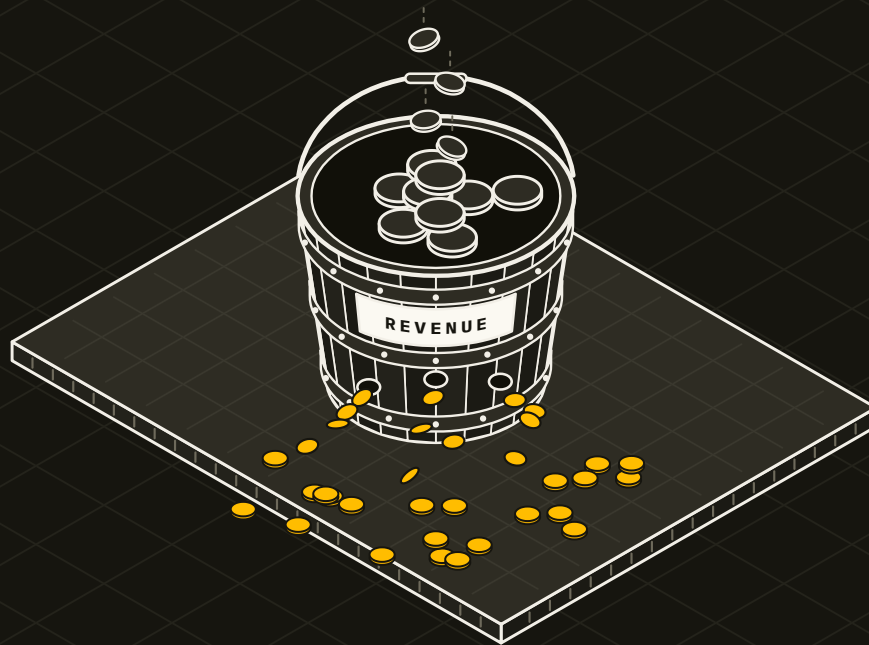
You can make these changes on your own schedule, before your customers force them. Part 2 looks at where margin leaks, so you can find the problems while they are still small.

SOURCES FOR THIS CHAPTER

1. Cursor, *Clarifying our pricing* (July 2025), cursor.com/blog/june-2025-pricing
2. PitchBook, *How long can AI agents like Cursor keep burning cash?*, pitchbook.com
3. TechCrunch, *Cursor in talks to raise \$2B+ at \$50B valuation* (17 April 2026), techcrunch.com
4. OfficeChai and Apidog coverage of Anthropic's weekly limits for Claude Pro and Max (July 2025)
5. GitHub Docs, *Requests in GitHub Copilot*, docs.github.com
6. Kyle Poyar, *The state of B2B monetization in 2026*, Growth Unhinged
7. Replit, *Effort-based pricing recap*, replit.com/blog
8. The SaaS CFO, *The death of per-seat pricing*, thesaascfo.com

Where margin leaks

Margin in an AI product leaks through a small number of predictable gaps. This part names them, shows how each one appears in your numbers, and looks closely at the customers who account for most of your provider cost.



06 Seven common margin leaks

07 The heavy-user problem

Seven common margin leaks

Margin in an AI product rarely disappears all at once. It leaks through a handful of predictable gaps between what you charge and what you pay.

Each leak below is a mismatch between the way a plan charges and the way cost behaves. Most AI products have at least three of them. Knowing which ones apply to you tells you where to look first, and which plan changes in Part 4 will do the most good.

FIGURE 6.1

Seven places margin leaks



01 Flat and unlimited plans

Heavy users consume far more than the plan price covers.



02 Seats that hide usage

One seat can cost you many times what another costs.



03 Credits priced on old costs

The credit rate stays fixed while cost per task moves.



04 Outcomes you cannot bill

Failed attempts cost money and earn nothing.



05 Free tiers and trials

Every free run has a real provider cost.



06 Model and provider changes

New models, tokenizers, and price changes move your cost base.



07 Cost that arrives late

Provider cost lands weeks later, in a different system.

Each leak is a gap between how a plan charges and how cost behaves. The chapters in Part 4 cover the plan structures that close each one.

1. Flat and unlimited plans

A flat monthly price with generous or unlimited usage is the easiest plan to sell and the hardest to keep profitable. It works while usage stays close to the average you priced for. It stops working when a group of customers uses far more than that, as Chapter 2 showed with the break-even curve.

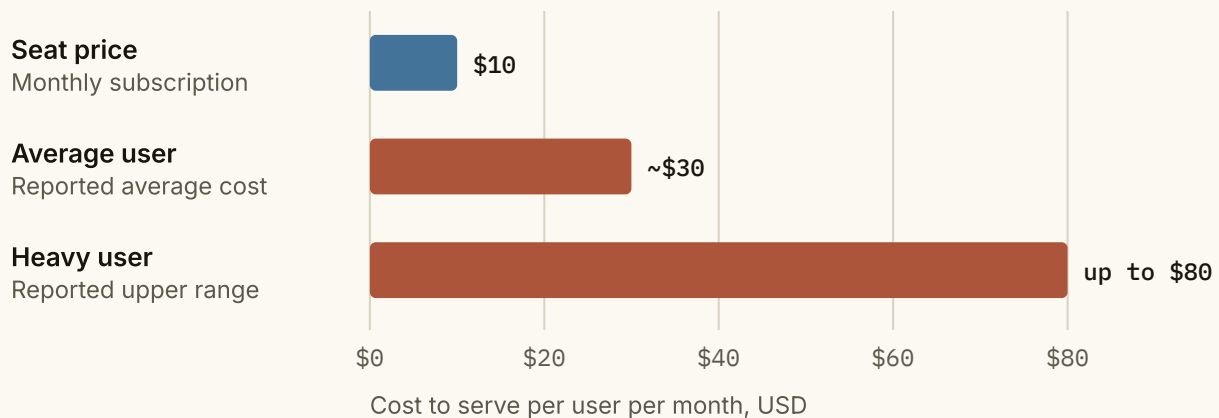
The public examples are consistent. Cursor's unlimited tiers and Anthropic's Max plans both needed limits once heavy users arrived. Anthropic said one user consumed tens of thousands of dollars of model usage on a \$200 plan.

2. Seats that hide usage

Seat pricing charges the same for every user, while each user's cost depends on what they do. SaaStr, summarizing press reports, described GitHub Copilot's early economics this way: in early 2023, Microsoft lost about \$20 per user per month on average on a \$10 plan, and its heaviest users cost up to \$80 a month.

FIGURE 6.2

One seat price, very different costs



Reported figures for GitHub Copilot in early 2023, as summarized by SaaStr from press reports. The average cost is derived from the reported average loss of about \$20 per user on a \$10 plan.

Seats create a second problem as agents take on more work. When an agent does the job of several people, your customer needs fewer seats while running more usage through your product. Jason Lemkin of SaaStr described moving from more than 10

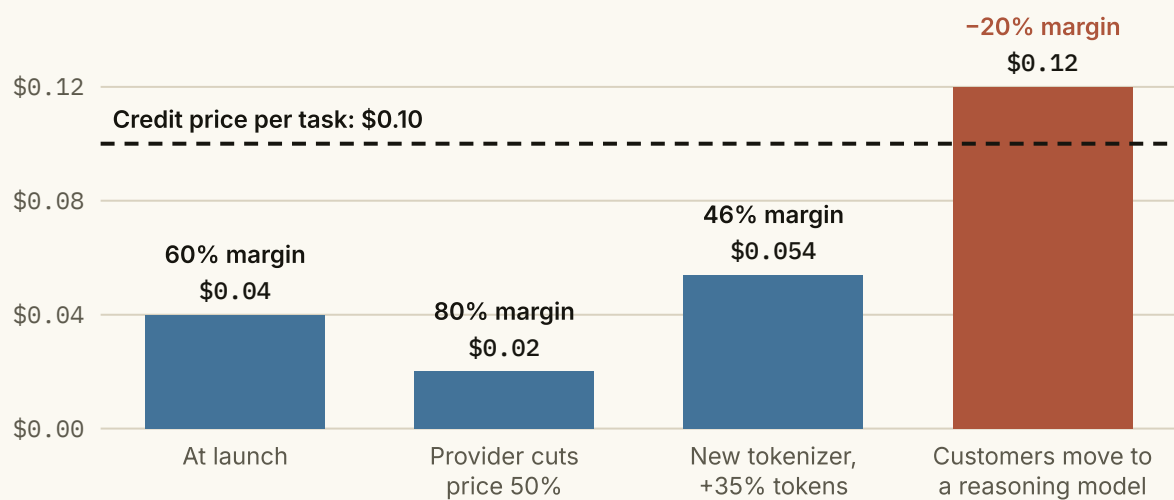
human seats to two human seats and one API seat in a CRM, while more than 20 agents used the system about 100 times as much as the people had. His bill still rose by 83%, because the vendor charged for the API usage. A vendor charging only for seats would have seen revenue fall as cost rose.

3. Credits priced on old costs

Credits feel safe because customers pay for them in advance. The margin on a credit, though, is set on the day you choose its price. After that, the price stays fixed while the cost of what customers spend credits on keeps moving.

FIGURE 6.3

A fixed credit price against a moving cost



Illustrative. One task burns 10 credits at \$0.01 each, a price of \$0.10 per task, set when the task cost \$0.04. The later bars show the same task after a provider price cut, a tokenizer change, and a move to a reasoning model that uses three times as many tokens.

Provider price cuts improve the margin on a credit. Tokenizer changes, longer tasks, and customers switching to a more capable model erode it, and in the last case the same credit becomes a loss. Chapter 11 covers credit designs that hold up, such as burn rates that differ by model.

4. Outcomes you cannot bill

Outcome pricing charges for a result, such as a resolved support ticket. The agent's cost is incurred on every attempt, including the ones that fail. That makes the cost of each billed outcome equal to the cost per attempt divided by the share of attempts that succeed.

Fin, the customer service agent, charges \$0.99 per outcome and counts one only when a customer confirms the answer resolved their issue or does not ask for more help. If an attempt costs \$0.15 and 65% of attempts succeed, each billed outcome costs about \$0.23. If the domain is harder, attempts cost \$0.30, and only 40% succeed, each billed outcome costs \$0.75. The price is the same in both cases, and the margin falls from about 77% to about 24%. Chapter 10 works through this in detail.

5. Free tiers and trials

Every free agent run has a real provider cost, and free users often run your most expensive tasks while they test what the product can do. Replit and GitHub both meter their free tiers for this reason. A free tier is a marketing expense, and it deserves a budget and a cap like any other.

6. Model and provider changes

Your cost base changes whenever a provider changes prices or you change which model your product uses. Chapter 4 covered the examples: GPT-5.5 launched at double its predecessor's per-token price, Opus 4.7's tokenizer can produce up to 35% more tokens for the same text, and Gemini 3.8 Flash's promotional price doubles after 31 December 2026.

The leak happens when these changes pass through to your cost without any change to what you charge. Upgrading your default model because it performs better is often the right product decision. It is also a pricing decision, and it needs to be made as one.

7. Cost that arrives late

Revenue is recorded when you invoice. Provider cost usually arrives later, on a provider invoice, in a different system, at a different level of detail. By the time finance has matched the two, the month is over and the heavy users have already been served.

This leak makes the other six harder to find. A team that sees cost a month late will spot a margin problem a month or two after it starts, and fix it a month or two after that.

LEAK	WHERE IT SHOWS UP FIRST	STRUCTURE THAT CLOSES IT
Flat and unlimited plans	Cost per account rising faster than revenue	Allowance plus usage-based overage, caps
Seats that hide usage	Wide cost spread between seats on one plan	Per-seat allowances, pooled usage
Credits priced on old costs	Cost per credit drifting up	Model-aware burn rates, repricing terms
Outcomes you cannot bill	Low resolution rates on some accounts	Cost floors, attempt limits
Free tiers and trials	Provider cost with no revenue	Free-tier budgets and hard caps
Model and provider changes	Step changes in cost per task	Cost review on every model change
Cost that arrives late	Margin surprises at month end	Cost recorded on every signal

SOURCES FOR THIS CHAPTER

1. Anthropic statements on weekly limits for Claude Pro and Max, as covered by OfficeChai and Apidog (July 2025)
2. SaaStr, *Have AI gross margins really turned the corner?*, saastr.com
3. The SaaS CFO, *The death of per-seat pricing*, thesaascfo.com, and Growth Unhinged, on Jason Lemkin's account
4. Fin, *Fin pricing: outcomes*, fin.ai/help
5. Finout, *Anthropic API pricing in 2026*; Apidog, *GPT-5.5 pricing*; Google, Gemini Developer API pricing

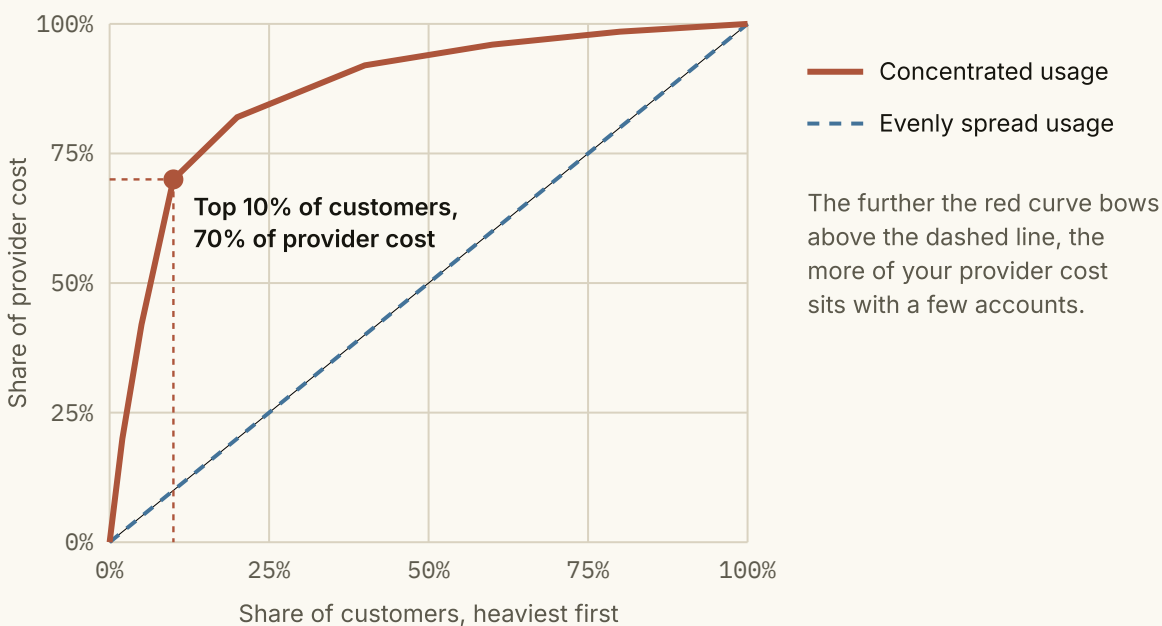
The heavy-user problem

In most AI products, a small group of customers accounts for a large share of provider cost. How you find and handle that group decides your margin.

Usage in AI products tends to concentrate. Kyle Poyar of Growth Unhinged has written that, in his experience, the top 10% of power users drive 70% or more of token consumption. That is one practitioner's observation, and the shape will differ for your product. It matches what the public repricings in Chapter 5 showed: a small tail of customers large enough to change a company's economics.

FIGURE 7.1

How concentrated is your provider cost?



Illustrative concentration curve drawn to match the observation that the top 10% of users drive about 70% of token consumption. Plot your own customers the same way: sort by provider cost, heaviest first, and chart the cumulative share.

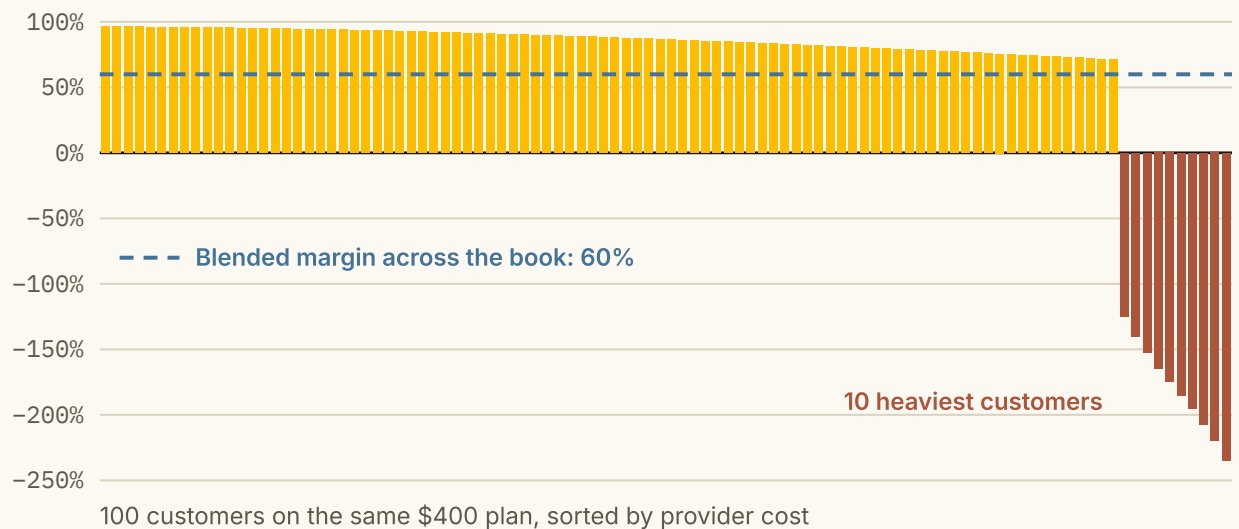
What concentration does to a flat plan

Take a book of 100 customers on a flat \$400 monthly plan. Revenue is \$40,000 a month and provider cost is \$16,000, for a blended margin of 60%. That looks like a healthy AI business by the benchmarks in Chapter 2.

Now look at the distribution. The 10 heaviest customers account for \$11,200 of the cost, an average of \$1,120 each. Each of them runs at a margin of about minus 180%. The other 90 customers account for \$4,800, about \$53 each, and run at a margin of about 87%.

FIGURE 7.2

Margin by customer behind a 60% blended margin



Illustrative. 100 customers pay \$400 a month each. Total provider cost is \$16,000, of which the 10 heaviest customers account for \$11,200. Amber bars are margin, red bars are losses.

Three things follow from this picture. The blended number is accurate and tells you almost nothing about what to do. The 90 profitable customers are paying for the 10 unprofitable ones. And if the heavy group grows from 10 to 15 customers, which is likely if they are your most engaged users, the blended margin falls fast.

Heavy users are usually your best users

It is tempting to treat heavy users as a problem to push out. In practice they are often your most engaged customers: the teams getting the most value, telling others about the product, and most likely to expand. Cursor's split between profitable enterprise accounts and unprofitable individual developers shows the other side. The heaviest users are not always the ones paying the most.

The goal is to price heavy use so that it is profitable, and to do it in a way these customers understand and accept. That usually means their bill grows with their usage above a clear allowance, with controls they can set, and with fair warning before any change.

Signs to watch for

You can find a heavy-user problem early if you track a few numbers per account. None of them needs a sophisticated system. All of them need provider cost attached to each customer.

SIGNAL	WHAT IT SUGGESTS	CHECK IT
Cost to revenue ratio above 50%	The account is close to break-even	Weekly, per account
Cost per account growing faster than revenue	Usage is outgrowing the plan	Month over month
Share of requests on premium models rising	Model mix is shifting upward	On every model change
Retries or model calls per task rising	Tasks are getting harder or looping	Per agent and per feature
Usage near or above the plan allowance	The plan no longer fits the account	Before each renewal

Options for a heavy user

Once you know which accounts are unprofitable, you have several options. Most companies use more than one.

1. **Talk to them first.** Show the customer their usage and explain what it costs to serve. Many heavy users would prefer a plan that fits them over a sudden limit.
2. **Move them to a plan with a usage component.** An allowance with overage, or a prepaid commitment sized to their usage, keeps the relationship and restores the margin.
3. **Route their work to cheaper models where quality allows.** Many tasks do not need a frontier model. Routing can cut cost without any change to price.
4. **Set a limit.** A hard cap or a rate limit protects you while you agree on a new plan.
5. **Reprice at renewal.** For contracts, the renewal is the natural point to align price with cost, if you raise it well before the renewal date.

▣ WHERE TO GO NEXT

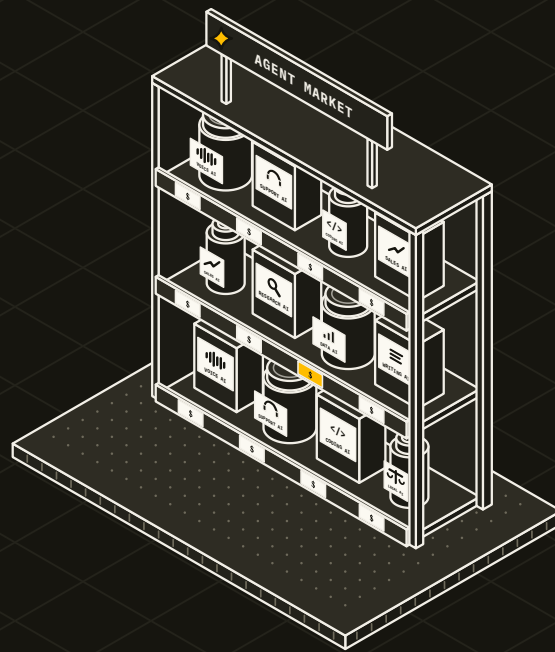
Part 3 covers how to choose what to charge for so that heavy use is priced from the start. Part 4 covers the plan structures that make usage above an allowance profitable.

SOURCES FOR THIS CHAPTER

1. Kyle Poyar, *The state of B2B monetization in 2026* and related posts, Growth Unhinged
2. TechCrunch, *Cursor in talks to raise \$2B+ at \$50B valuation* (April 2026), techcrunch.com
3. Anthropic statements on weekly limits, as covered by OfficeChai (July 2025)

Choosing what to charge for

The unit you charge for decides whether you can see your margin. This part compares pricing models, sets out a test for choosing a unit, and covers how to make outcomes and credits profitable.



08 Pricing models and their margin trade-offs

09 Choosing a unit you can cost

10 Outcome pricing that stays profitable

11 Credits that survive model changes

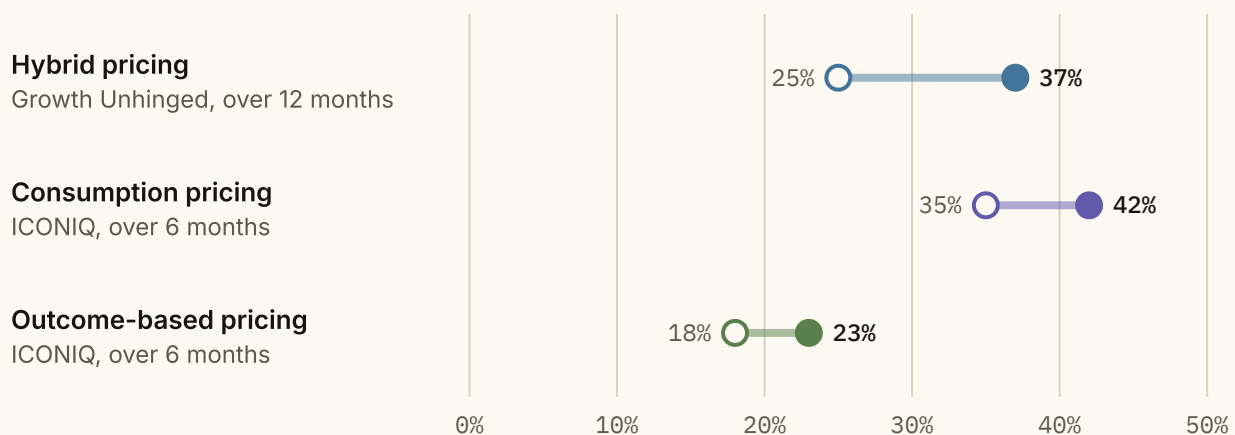
Pricing models and their margin trade-offs

Every pricing model balances predictability for your customer against protection for your margin. Choosing well starts with knowing where each one sits.

AI companies are changing how they charge, and quickly. In Growth Unhinged's 2026 survey of 230 B2B software and AI companies, three in four had changed their pricing or packaging in the previous year. The share using a hybrid model, usually a fixed fee combined with usage, rose from 25% to 37% in twelve months. ICONIQ's 2026 report found consumption pricing rising from 35% to 42% of respondents, and outcome-based pricing from 18% to 23%, over just six months.

FIGURE 8.1

Pricing models in use, rising fast



Share of surveyed companies using each model. Growth Unhinged, *The state of B2B monetization in 2026* (230 companies); ICONIQ, *State of AI 2026* (300+ executives). The two surveys cover different time windows.

Two other findings matter for margin. More than half of the companies in the Growth Unhinged survey, 54%, named internal costs and margins as the top factor in their pricing decisions. And companies rarely use one model on its own: ICONIQ found respondents combining 1.7 pricing models on average, and 29% of the Growth Unhinged sample let customers choose between models.

Seven models, briefly

Seat or flat fee

A fixed price per user or per account. It is the easiest to buy and forecast, and it offers no protection against heavy use. It works best for products where usage per customer is naturally bounded, or as the base layer of a hybrid plan.

Usage pass-through

Customers pay per token, request, or unit, often at a markup on provider cost. It aligns price with cost almost perfectly. It also hands your customers a bill they cannot predict, and it tends to anchor them on provider list prices, which puts pressure on your markup.

Prepaid credits

Customers buy a balance of credits in advance and spend them as they use the product. You collect cash up front and customers get a budget they understand. The margin depends entirely on how credits convert into work, which Chapter 11 covers.

Prepaid wallet

A cash balance the customer tops up and draws down, often with automatic reloads. Model APIs work this way: Google moved Gemini API developers from postpaid to prepaid billing with a minimum purchase of \$5. With a hard limit at zero, a wallet caps your exposure to any single customer.

Hybrid: platform fee plus usage

A fixed fee covers access and a base allowance, and usage above the allowance is billed separately. It gives customers a predictable floor and gives you protection at the top. Clay, as described by Kyle Poyar, separates the two explicitly: one charge for the

value of the platform and another for the cost of the data and tokens it consumes.

Outcome-based

Customers pay for a result, such as a resolved ticket. It is easy to connect to value, and it moves the cost of failed attempts onto you. Chapter 10 covers how to keep it profitable.

Per task or effort

Customers pay per agent task, with the price reflecting the work involved, as in Replit's effort-based pricing. It tracks cost closely and can surprise customers when a task takes more effort than they expected.

FIGURE 8.2
Trade-offs between the seven models

	Revenue predictability for you	Bill predictability for the buyer	Margin protection	Cost alignment
Seat or flat fee	High	High	Low	Low
Usage pass-through	Low	Low	High	High
Prepaid credits	Medium	Medium	Medium	Medium
Prepaid wallet	Medium	Medium	High	High
Hybrid: platform fee plus usage	Medium	Medium	High	High
Outcome-based	Low	Medium	Varies	Low
Per task or effort	Low	Low	High	High

Our assessment, based on the public examples in this ebook. Hybrid and wallet ratings assume caps and alerts are in place.

Our qualitative assessment. Outcome pricing's margin protection varies with the resolution rate and the cost per attempt.

Commodity metrics make weak pricing

Poyar makes a point worth keeping in mind as you choose. When a product looks like a commodity, the company selling it has little pricing power, and tokens or raw credits are about as close to a commodity as a pricing metric gets. If your price is expressed in tokens, your customers will compare it with provider list prices.

The models that hold margin best tend to charge for something the customer values, such as a task, a document, or an outcome, while measuring the cost of that thing underneath. The next chapter covers how to choose that unit.

■ CHOOSING A STARTING POINT

If you are early and your usage per customer varies widely, a hybrid plan with an allowance and usage above it is the safest default. It is simple to buy, it caps your downside, and it lets you learn what customers value before you commit to outcomes.

SOURCES FOR THIS CHAPTER

1. Kyle Poyar, *The state of B2B monetization in 2026*, Growth Unhinged, growthunhinged.com
2. ICONIQ, *State of AI: The Builder's Economy* (2026), iconiq.com
3. Google, Gemini API billing documentation, ai.google.dev/gemini-api/docs/billing
4. Replit, *Effort-based pricing recap*, replit.com/blog

Choosing a unit you can cost

The unit you charge for has to mean something to your customer and something to your cost ledger. If it only means something to one of them, your margin is a guess.

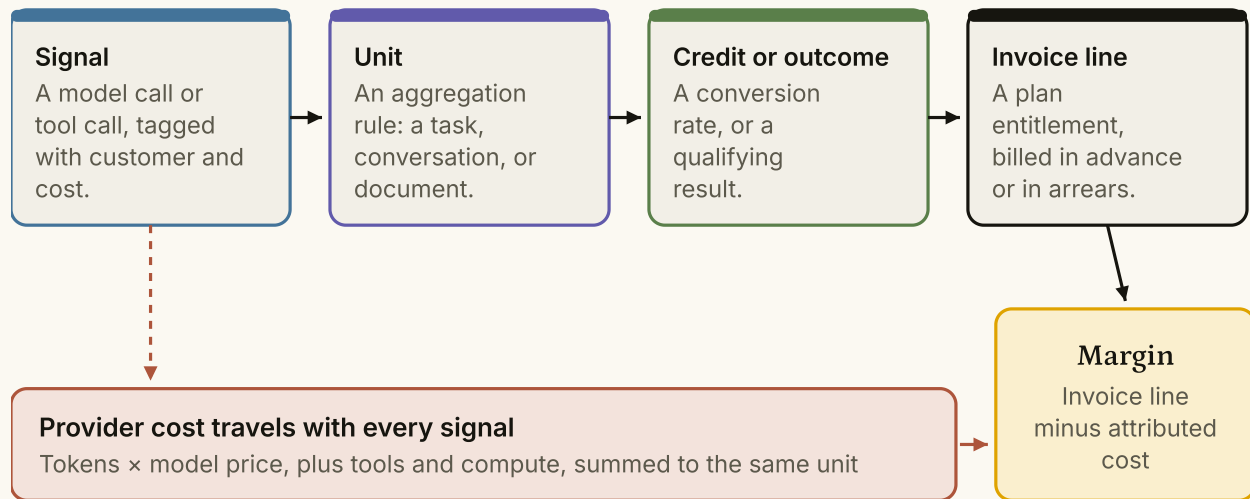
Most pricing advice focuses on picking a value metric: the thing your customer pays for that grows with the value they receive. For an AI product, that is half the test. The other half is whether you can say, reliably, what one of those units costs you to deliver.

From signal to invoice line

Every unit you bill should be built from signals, and every signal should carry both a customer identifier and a provider cost. When that is true, revenue and cost can be joined for any unit, customer, plan, agent, or model.

FIGURE 9.1

How signals become billed units, with cost alongside



Signals are aggregated into units, units convert into credits or qualify as outcomes, and those appear on the invoice as plan entitlements. Provider cost follows the same path, so margin can be computed at every level.

In practice, each layer is a rule. A signal is a raw event, such as a model call. A unit is an aggregation rule, such as all the signals that belong to one support conversation. A credit is a conversion rate from units or signals into a balance, and that rate can differ by model. An outcome is a qualifying event with an attribution rule, such as a ticket that stays resolved for 24 hours. The invoice line is the entitlement in the customer's plan.

The costability test

Before you commit to a unit, ask five questions about it.

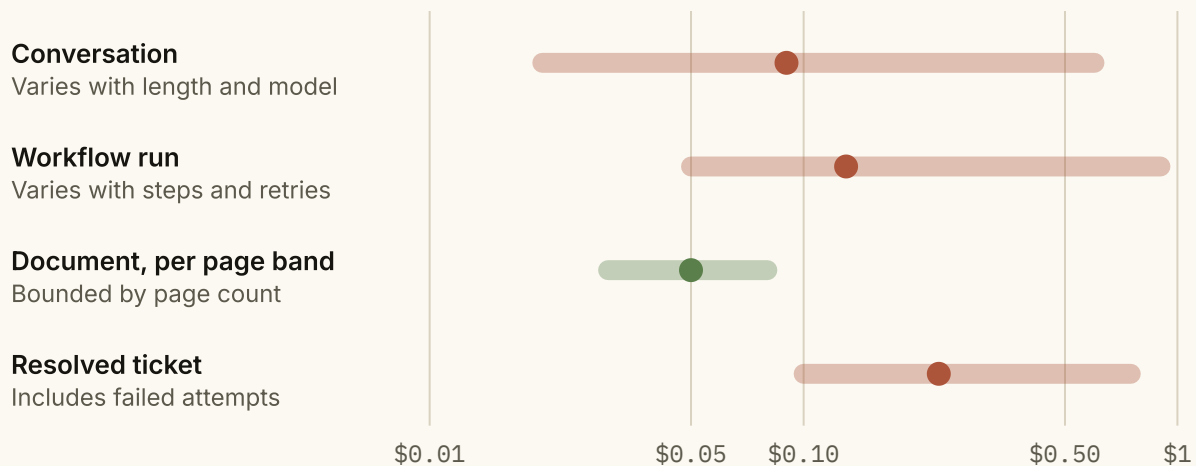
1. **Does your customer understand it?** They should be able to relate it to the work they want done without reading your docs.
2. **Can you measure it exactly?** It must be derived from signals you record, with clear rules for edge cases such as partial or failed work.

3. **Is its cost bounded?** The cost of one unit should fall within a range you can price for. A unit whose cost can vary by 30 times is hard to price at a fixed rate.
4. **Can your customer predict how many they will use?** Predictable volume makes budgeting and prepaid commitments easier to sell.
5. **Does it grow with the value delivered?** More units should mean more value for the customer, so that a larger bill feels fair.

The third question is the one most teams skip. The chart below shows why it matters. Two units can have similar median costs and very different ranges.

FIGURE 9.2

Cost per unit for four candidate units



Bar: range of cost per unit, from low to the 95th percentile. Dot: median. Log scale.

Illustrative. A conversation or workflow run can cost many times its median when it runs long or retries. A document priced by page band has a narrow range. A resolved ticket includes the cost of failed attempts, so its range is wide.

Making a unit costable

If the unit your customers value has a wide cost range, you can usually narrow it without changing what customers see.

- **Band it.** Price documents by page band or conversations by length band, so that long ones cost more.

- **Bound it.** Set a limit on model calls or retries per unit, and stop or escalate when it is reached.
- **Route it.** Send simple units to smaller models and reserve frontier models for the units that need them.
- **Split it.** Charge a base price per unit plus a premium for options that drive cost, such as a more capable model or a larger context.

CANDIDATE UNIT	CUSTOMER UNDERSTANDS IT	COST BOUNDED	NOTES
Tokens	Developers only	Yes	Precise and commodity-like; invites comparison with provider prices
Seat	Yes	No	Needs an allowance behind it in an AI product
Conversation or task	Yes	Partly	Band by length or cap model calls per task
Document or record processed	Yes	Mostly	Band by size; cost scales with pages or fields
Workflow run	Yes	Partly	Limit steps and retries per run
Resolved outcome	Yes	No	Cost includes failed attempts; see Chapter 10

 A RULE OF THUMB

Price at the level your customer thinks in, and measure at the level your provider bills in. The gap between the two is where your margin is decided.

Outcome pricing that stays profitable

Charging for results is the closest link between price and value. It also means paying for every attempt that does not produce a result.

Outcome-based pricing has moved from theory to practice in customer service. Fin, the customer service agent from the company formerly known as Intercom, charges \$0.99 per outcome. Salesforce Ben reported that Fin was approaching \$100M in annual recurring revenue inside a company with more than \$400M, and Salesforce agreed in June 2026 to acquire Fin for about \$3.6 billion. Sierra also prices on outcomes. It publishes no rate card, and TechCrunch reported that it reached \$100M in annual recurring revenue in November 2025.

Fin's approach also shows how much detail an outcome definition needs. Fin counts an outcome when the customer confirms the answer resolved their issue, or does not ask for further help after Fin answers. When Fin runs on another company's help desk, there is also a base plan of \$49 a month that includes 50 resolutions. In an interview with GTMnow, Fin's president described a performance guarantee of up to \$1M.

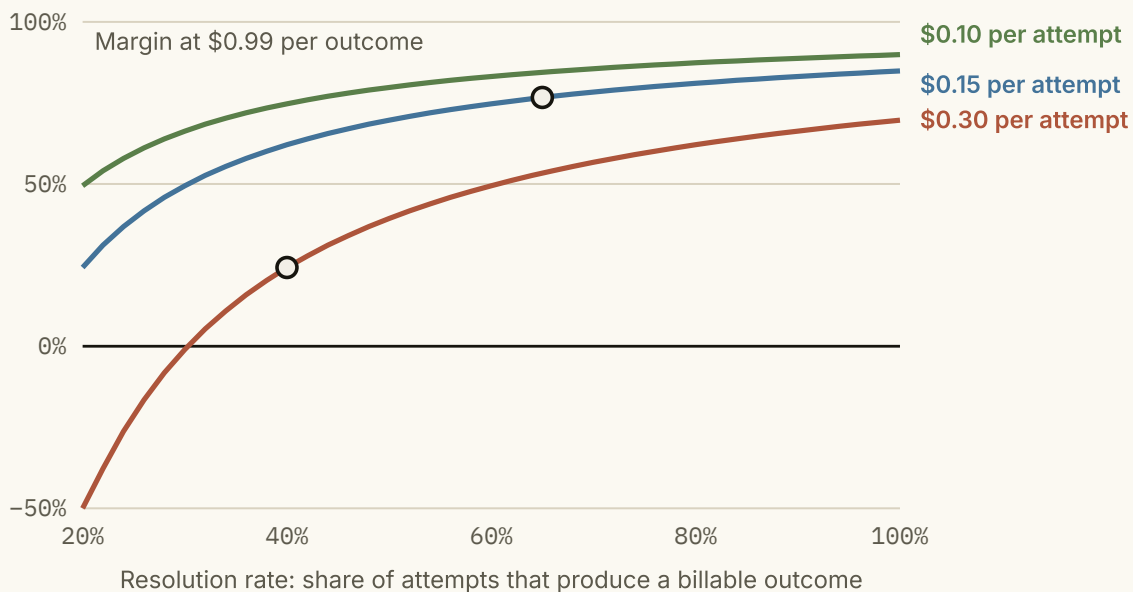
The arithmetic of outcome margin

An outcome's cost is incurred on every attempt, and revenue is earned only on the attempts that succeed. So the cost of each billed outcome is the cost per attempt divided by the resolution rate.

SCENARIO	COST PER ATTEMPT	RESOLUTION RATE	COST PER OUTCOME	MARGIN AT \$0.99
Mature agent, familiar domain	\$0.15	65%	\$0.23	77%
Harder domain, more agentic	\$0.30	40%	\$0.75	24%
Same harder domain, improved to 60%	\$0.30	60%	\$0.50	49%

FIGURE 10.1

Outcome margin depends on two numbers



Illustrative, at a price of \$0.99 per outcome. The open circles mark the first two scenarios in the table. Below a resolution rate of about 30%, an agent costing \$0.30 per attempt loses money on every outcome it bills.

Two levers move outcome margin: the cost of an attempt and the share of attempts that succeed. Improving the agent's resolution rate raises margin as surely as cutting cost does, and it also makes the product better for the customer. That alignment is one of the

strongest arguments for outcome pricing.

Four tests for an outcome

Poyar's CAMP framework is a useful check on whether an outcome is ready to price. Each letter is a question.

TEST	THE QUESTION TO ASK
Consistency	Does the agent produce the outcome reliably across customers and use cases?
Attribution	Can you show the outcome happened because of your agent and not something else?
Measurability	Can you detect the outcome from data you already record, without manual review?
Predictability	Can your customer forecast how many outcomes they will pay for?

Designing outcome pricing that holds

- **Write the definition down.** Say exactly what counts, what does not, and the window in which an outcome can be reversed.
- **Know your resolution rate per customer.** A blended rate can hide customers whose rate is far below average and whose outcomes cost more than they pay.
- **Set a platform fee or minimum.** A base fee covers fixed costs and the cost of attempts during onboarding, when resolution rates are lowest.
- **Limit attempts.** Cap the model calls an agent can spend on a single case before escalating to a person.
- **Price new use cases separately.** A new domain starts with a lower resolution rate. Price it on usage until its rate is known.

■ WHEN TO WAIT

If you cannot yet measure your resolution rate by customer, price on usage or on a hybrid plan first. Move to outcomes once the rate is stable and you can see it in your data.

SOURCES FOR THIS CHAPTER

1. Fin, *Fin pricing: outcomes*, fin.ai/help
2. Salesforce Ben and Salesforce announcements on the acquisition of Fin (June 2026); Salesforce Form 10-Q for the quarter ended 31 July 2026
3. GTMnow, *GTM 178: How Intercom built the highest-performing AI agent*, interview with Archana Agrawal, gtmnow.com
4. TechCrunch, *Bret Taylor's Sierra reaches \$100M ARR in under two years* (21 November 2025)
5. Kyle Poyar, Growth Unhinged, on the CAMP framework for outcome-based pricing

Credits that survive model changes

Credits give customers a budget and give you cash up front. Whether they give you margin depends on how they burn.

Credits are one of the fastest-growing pricing structures in AI. In Growth Unhinged's 2026 survey, 29% of companies used credits and another 33% planned to add them within six to twelve months. About half of companies with more than \$50M in annual recurring revenue planned to add them that year. Using data from the PricingSaaS 500, Growth Unhinged reports that credit models grew 126% year over year in 2025.

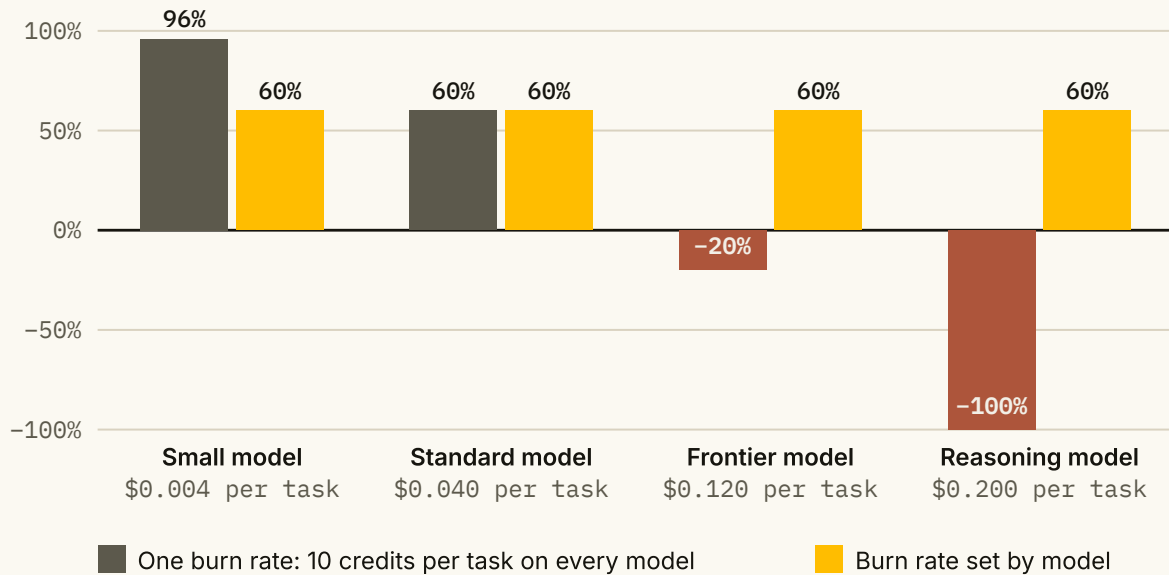
The appeal is easy to see. Customers buy a known budget. You collect cash before you incur cost. A single currency can cover many features and models. The risk, as Chapter 6 showed, is that the credit's price is fixed while the cost of what customers spend it on keeps moving.

One burn rate is a margin bet

The simplest design charges the same number of credits for every task. That works until customers spread their usage across models with very different costs.

FIGURE 11.1

One burn rate compared with a burn rate per model



Illustrative. Credits sell at \$0.01. A single rate of 10 credits per task prices every task at \$0.10. Model-based rates set credits per task at cost divided by \$0.004, which holds a 60% margin on every model: 1 credit on the small model, 10 on the standard model, 30 on the frontier model, and 50 on the reasoning model.

With a single rate, the small model earns 96% margin and the reasoning model loses 100%. Your overall margin then depends on your customers' model mix, which you do not control and which tends to move toward more capable models. With a rate per model, every task earns the same margin, and customers can see why a harder task costs more.

A public reference design

GitHub Copilot's premium requests are a well-documented example of model-based burn. Included models count as zero premium requests, standard models count as one, and heavier models count as a multiple. GitHub also discounts the multiplier by 10% when a user lets Copilot choose the model automatically, which rewards customers for letting the product route to cheaper models.

Design choices that protect credit margin

- **Set burn rates from cost, with a margin target.** Credits per task equals the cost per task divided by the credit price times one minus your target margin.
- **Use a cost you expect to hold.** Base rates on list prices you expect to hold for the contract term, and leave promotional rates out.
- **Keep the right to adjust rates.** Say in your terms that burn rates can change when provider costs change, with notice.
- **Show the burn.** Let customers see how many credits each task used and why. Visible rates reduce disputes and support tickets.
- **Decide expiry and rollover up front.** Whether unused credits expire, roll over, or can be refunded affects both customer trust and how you recognize revenue, as Chapter 18 explains.
- **Reward cheaper choices.** A discount for automatic routing or batch processing shares savings with customers and pulls usage toward your cheaper paths.

■ THE FORMULA

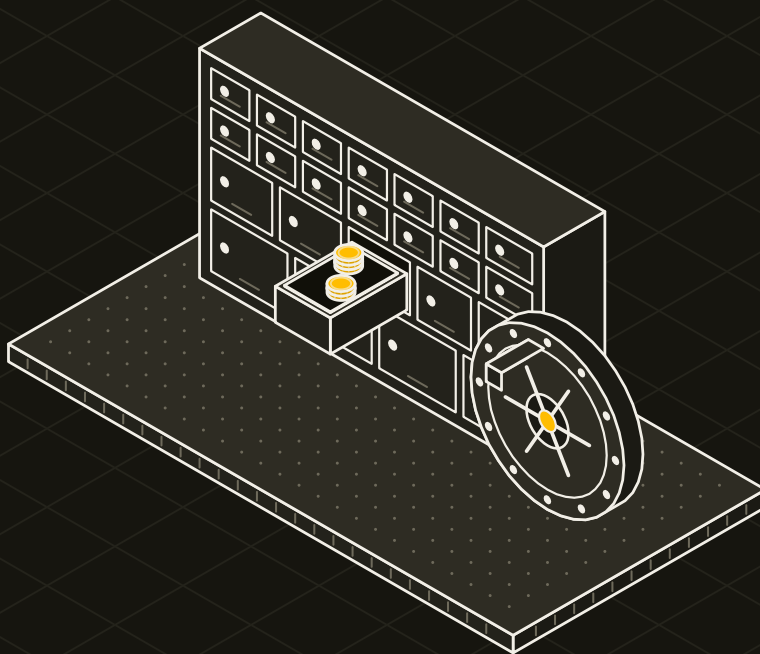
Credits per task = cost per task ÷ (credit price × (1 – target margin)). At a credit price of \$0.01 and a 60% target, a task that costs \$0.12 should burn 30 credits.

SOURCES FOR THIS CHAPTER

1. Kyle Poyar, *The state of B2B monetization in 2026*, Growth Unhinged, citing PricingSaaS 500 data
2. GitHub Docs, *Requests in GitHub Copilot* and *Understanding and managing requests in Copilot*, docs.github.com

Plan structures that protect margin

Plans that stay profitable combine parts paid in advance with parts paid as used, backed by guardrails and a cost floor. This part shows how, with worked plans and the arithmetic behind them.



12 Combining prepaid and usage-based parts

13 Guardrails that cap your downside

14 A cost floor under value-based prices

15 Worked plans, with the numbers

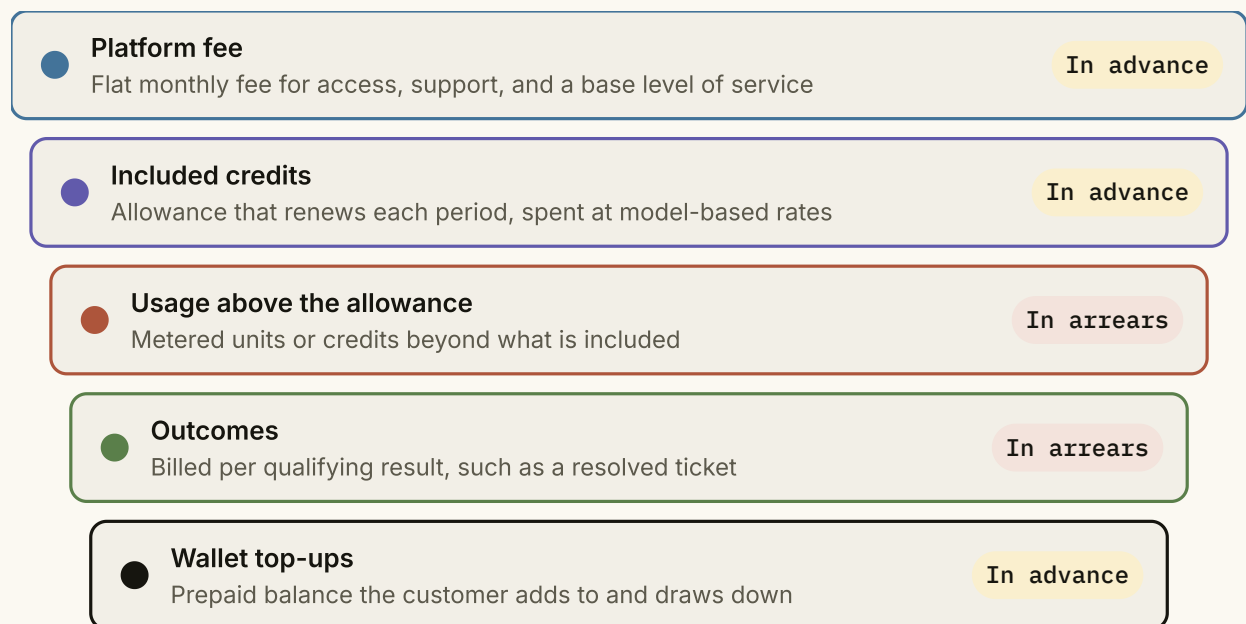
Combining prepaid and usage-based parts

Most AI products end up with plans built from more than one part. The skill is in choosing which parts are paid up front and which are paid as used.

A plan does not have to be a single price. It can combine a flat fee, a set of units, an allowance of credits, a price per outcome, and a prepaid wallet. Each of those parts can be billed in advance, at the start of the period, or in arrears, after the usage has happened.

FIGURE 12.1

A hybrid plan, part by part



Each part of a plan has its own billing timing. Parts billed in advance give you cash and give customers predictability. Parts billed in arrears track actual usage and protect margin on heavy use.

Why combine them

Each part does a different job. A platform fee covers your fixed costs and gives the customer a predictable minimum. Included credits or units give the customer room to use the product without watching a meter. Usage above the allowance, billed in arrears, makes sure that heavy use pays for itself. Outcomes connect part of the price to results. A wallet lets customers prepay for flexible usage and caps your exposure.

The public examples in Part 1 all converged on some version of this structure. Cursor includes \$20 of usage in its Pro plan and bills usage beyond that at cost. Claude Max subscribers can buy additional usage at API rates. GitHub Copilot pairs a seat price with an allowance of premium requests and overage at \$0.04 each.

Choosing advance or arrears for each part

PART	USUALLY BILLED	WHY
Platform fee	In advance	Predictable for both sides; covers fixed cost
Seats or other units	In advance	Customers budget per seat; true up at renewal
Included credits	In advance	Cash before cost; customers get a known budget
Usage above the allowance	In arrears	Bills what was actually used, so heavy use pays for itself
Outcomes	In arrears	Billed once the outcome qualifies and cannot be reversed
Wallet top-ups	In advance	Prepaid balance, drawn down as used

Setting timing per part, rather than once for the whole plan, is what makes hybrid plans work. A plan with a platform fee in advance and overage in arrears gives the customer a predictable bill most months and protects you in the months they use more.

Prepaid commitments

Larger customers often prefer a committed annual spend in exchange for a lower rate. Snowflake's capacity contracts are a long-standing example from outside AI: customers commit to capacity billed annually in advance, revenue is recognized as they consume it, and unused capacity can generally roll over when they purchase more.

A commitment helps your margin in two ways. It gives you cash to cover provider cost as it accrues, and it gives you a forecast of usage, which is useful when you negotiate committed rates with your own providers.

Keeping it simple to buy

A plan with five parts can be hard to explain. Most customers only need to understand two numbers: what they will pay each month at their normal usage, and what happens if they use more. Lead with those two numbers and keep the rest in the plan details.

■ A DEFAULT HYBRID PLAN

A monthly platform fee billed in advance, an included allowance of credits at model-based burn rates, and usage above the allowance billed in arrears at a margin you set. Add a wallet or an annual commitment for customers who want to prepay.

SOURCES FOR THIS CHAPTER

1. Cursor, *Clarifying our pricing*, cursor.com/blog
2. GitHub Docs, *Requests in GitHub Copilot*, docs.github.com
3. Snowflake Inc., Form 10-K for fiscal 2026, sec.gov

Guardrails that cap your downside

A good plan structure sets the price. Guardrails decide what happens when a customer's usage goes far beyond what the price assumed.

Guardrails are the rules that limit how much any one customer, task, or month can cost you. They matter most in the cases pricing cannot anticipate: a runaway agent, a customer who automates far more than expected, or a model change that raises cost overnight. Each guardrail closes one or more of the leaks from Chapter 6.

FIGURE 13.1

Which guardrail closes which leak

	Flat plans	Seats	Credits	Outcomes	Free tiers	Model changes	Late cost
Spend caps, default-off overage	●	●	●	●	●	●	●
Minimum commits	●	●	●	●	●	●	●
Rollover and expiry rules	●	●	●	●	●	●	●
Rate limits	●	●	●	●	●	●	●
Model-tier routing	●	●	●	●	●	●	●
Attempt and loop limits	●	●	●	●	●	●	●
Model-based burn rates	●	●	●	●	●	●	●
Fair-use policy	●	●	●	●	●	●	●
Cost recorded on every signal	●	●	●	●	●	●	●

Our mapping. Filled circles show the leaks each guardrail addresses directly. Most products need three or four of these, chosen for the leaks that apply to them.

The guardrails in brief

Spend caps and overage that is off by default

Let customers set a monthly spend limit, and require an explicit choice before overage is billed. GitHub Copilot budgets default to zero, so overage is refused unless an administrator turns it on. Replit offers hard budget caps and alerts. Caps protect your customer from surprise bills and protect you from usage that nobody intended.

Minimum commits

A minimum monthly or annual spend guarantees revenue that covers your fixed costs and the early, inefficient phase of a new deployment.

Rollover and expiry rules

Decide whether unused credits expire at the end of a period, roll over for a limited time, or carry forward indefinitely. Generous rollover builds goodwill. It also accumulates a liability that customers can spend later, when your costs may be different.

Rate limits

Limits on usage per hour, day, or week keep a single customer from consuming a month's allowance in an afternoon. Anthropic combines session limits that reset every five hours with weekly limits.

Model-tier routing

Send each request to the least expensive model that can handle it well. Cursor's Auto mode and Copilot's automatic model selection both do this, and ICONIQ found better routing among the reasons builders' unit economics improved.

Attempt and loop limits

Cap the number of model calls or retries per task. When the cap is reached, stop, return a partial result, or hand the task to a person. This one guardrail addresses the cost pattern from Chapter 3 where a retry loop tripled the cost of a single run.

Model-based burn rates

Charge more credits for more expensive models, as Chapter 11 described, so that a shift in model mix does not erode margin.

Fair-use policies

For plans marketed as unlimited, a written fair-use policy defines what normal use means and what happens beyond it. It gives you a basis for a conversation with the rare customer whose usage looks nothing like the rest.

Cost recorded on every signal

None of the other guardrails work well without this one. If cost arrives a month late, caps and alerts act on stale numbers.

Give customers the same controls

The guardrails that customers can see and set themselves do the most for trust. Usage alerts at 50%, 80%, and 100% of an allowance, a spend cap they choose, and a clear view of what each task cost turn a potentially surprising bill into a predictable one. The public repricings in Chapter 5 went more smoothly where these controls existed.

■ START WITH THREE

If you have none of these in place, start with spend caps with overage off by default, a limit on model calls per task, and customer-facing usage alerts. Together they cover the most common and most expensive surprises.

SOURCES FOR THIS CHAPTER

1. GitHub Docs, *Understanding and managing requests in Copilot*, docs.github.com
2. Replit and VibeAnswers coverage of Replit budget controls
3. Morph, *Claude Code usage limits (2026)*, morphllm.com
4. ICONIQ, *State of AI: The Builder's Economy (2026)*, iconiq.com

A cost floor under value-based prices

Value should set your price. Cost should set the lowest price you will accept. Both numbers belong in every pricing decision.

There are two traditional ways to set a price. Cost-plus pricing starts from what the product costs and adds a margin. Value-based pricing starts from what the product is worth to the customer. Most pricing advice favors value-based pricing, and for good reason: it captures more of the value you create, and it keeps your price from following provider list prices down.

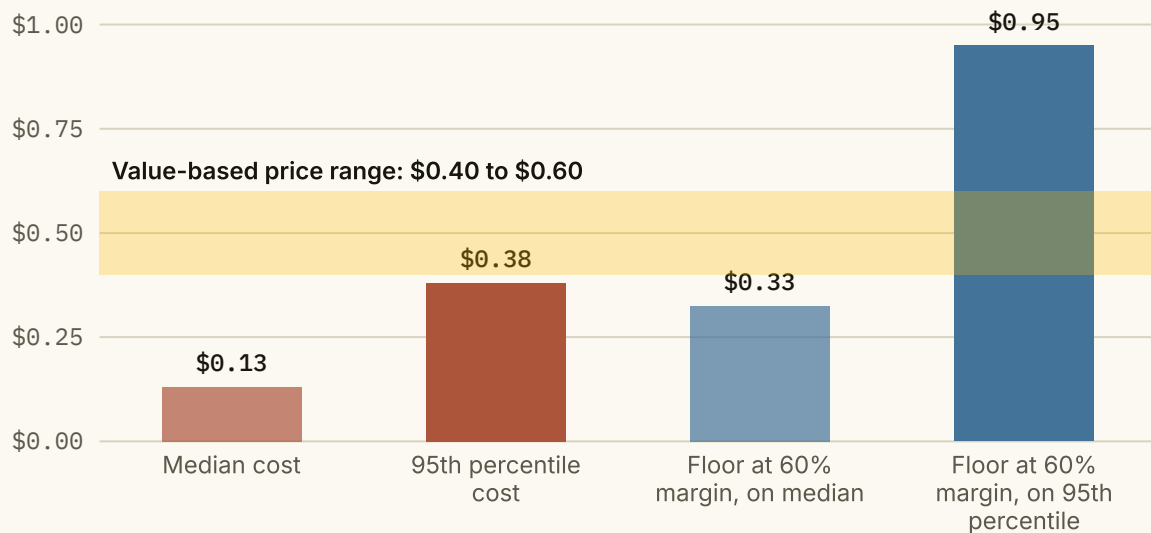
In an AI product, value-based pricing on its own has a gap. A price can be well below the value delivered and still below the cost of delivering it, especially for heavy tasks. The practical approach is to use both: a value-based price, with a floor set by cost that no unit is allowed to go below.

Setting the floor

The floor for a unit is its cost divided by one minus your target margin. The important choice is which cost to use. The median cost tells you what a typical unit costs. The 95th percentile tells you what your expensive units cost, and those are the units that decide whether a plan loses money.

FIGURE 14.1

Cost, floor, and value for one agent task



Illustrative, using the agent run from Chapter 3: a median cost of \$0.13 and a 95th percentile of \$0.38. Floors are set for a 60% target margin. The amber band is a value-based price range for the task.

In this example, the value-based price range of \$0.40 to \$0.60 sits comfortably above the floor for a median task, which is \$0.33. It sits well below the floor for an expensive task, which is \$0.95. Pricing at \$0.50 would be profitable on most tasks and would lose money on the expensive ones.

Three ways to close the gap

1. **Cap the cost.** A limit on model calls per task that keeps the 95th percentile cost at or below \$0.20 brings the floor down to \$0.50. At a price of \$0.50, every task earns at least 60%.
2. **Price the expensive tasks separately.** Charge a premium for long documents, larger context, or a more capable model, so the tasks that cost more also pay more.
3. **Accept a lower margin on some tasks, knowingly.** If expensive tasks are rare, a blended margin can stay above target. This works only if you track the share of expensive tasks and it stays stable.

Enforcing the floor

A floor only works if something enforces it. In practice that means burn rates tied to cost, limits on model calls per task, and a regular review of cost per unit against price per unit. When a provider changes prices or you change a default model, rerun the floor calculation before the change reaches customers.

■ THE FLOOR FORMULA

Floor price = cost per unit ÷ (1 – target margin). Use the 95th percentile cost when your plan has no limit on cost per unit, and the capped cost when it does.

SOURCES FOR THIS CHAPTER

1. Kyle Poyar, Growth Unhinged, on commodity pricing metrics and value-based AI pricing

Worked plans, with the numbers

This chapter puts the earlier ideas together into complete plans and shows the arithmetic, so you can rerun it with your own numbers.

Plan A: from a flat plan to a hybrid plan

Start with the book of 100 customers from Chapter 7. Each pays a flat \$400 a month. Total revenue is \$40,000 and provider cost is \$16,000, for a blended margin of 60%. The 10 heaviest customers cost \$1,120 each on average and run at about minus 180%. The other 90 cost about \$53 each.

Now replace the flat plan with a hybrid. Each customer pays a \$250 platform fee that includes \$100 of usage at cost. Usage beyond that is billed at 1.6 times provider cost.

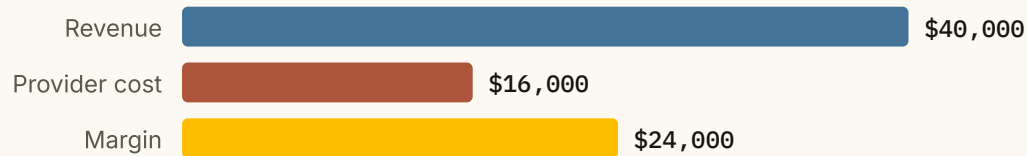
	FLAT PLAN	HYBRID PLAN
Light customer (90 accounts)	Pays \$400, costs \$53	Pays \$250, costs \$53
Heavy customer (10 accounts)	Pays \$400, costs \$1,120	Pays \$250 + (\$1,020 × 1.6) = \$1,882
Heavy customer margin	About -180%	About 40%
Light customer margin	About 87%	About 79%
Total revenue	\$40,000	\$41,320
Total provider cost	\$16,000	\$16,000
Blended margin	60%	61%
Accounts at a loss	10	0

FIGURE 15.1

Flat plan and hybrid plan across the same 100 customers

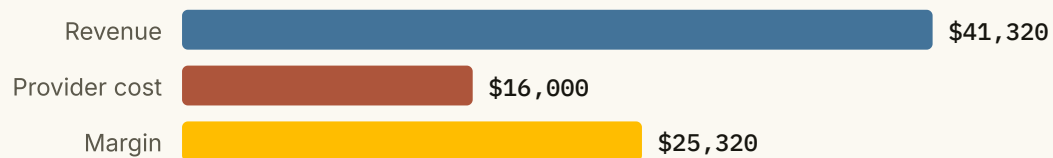
Flat plan: \$400 a month

60% margin · 10 accounts at a loss



Hybrid plan: \$250 plus overage

61% margin · no accounts at a loss



Illustrative. Assumes light customers stay within the included \$100 of usage and heavy customers keep their usage at the same level under the new plan.

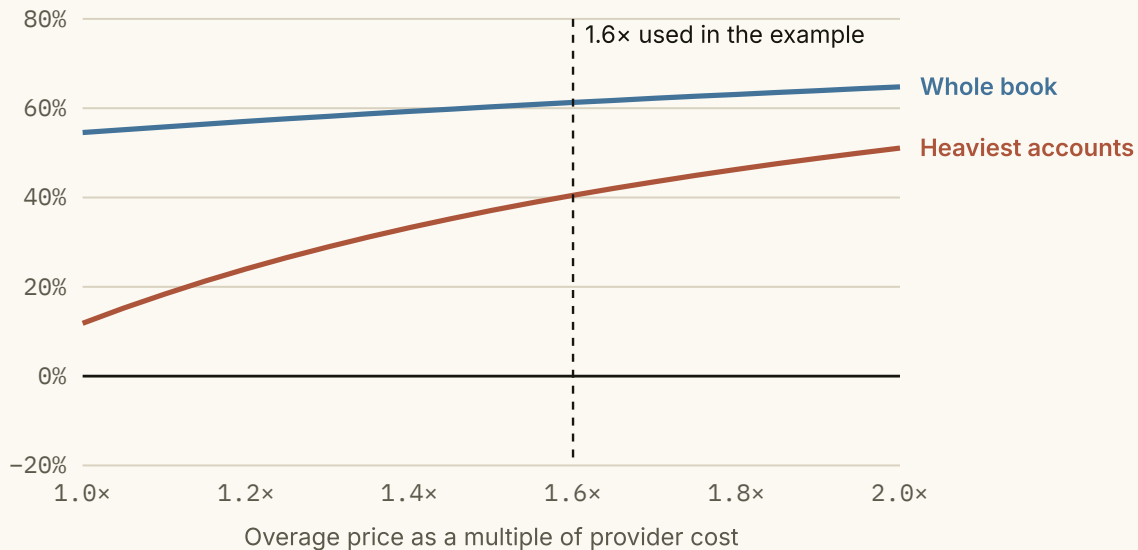
The blended margin barely changes. What changes is its composition. Under the flat plan, 90 customers paid for 10. Under the hybrid plan, every customer pays for their own usage, and none is served at a loss. Light customers also pay less, which makes the plan easier to sell to new customers.

The example depends on two assumptions worth testing. It assumes heavy customers accept the overage rather than leaving, and it assumes light customers stay light. If some heavy customers leave, total revenue falls and so does total cost, often by more. If light customers grow, they start paying overage, which is the intended result.

How the overage multiplier changes the result

FIGURE 15.2

Margin as the overage price rises



Illustrative, for the hybrid plan above. At 1.0 times cost, heavy accounts break even on their overage and earn margin only on the platform fee. At 1.6 times, they earn about 40%.

Plan B: outcomes with a platform fee

A customer service agent charges a \$500 monthly platform fee and \$0.99 per resolved conversation, with a limit of eight model calls per conversation before escalating to a person. Suppose the agent costs \$0.15 per attempt and resolves 65% of attempts, and the customer handles 4,000 conversations a month.

LINE	CALCULATION	AMOUNT
Resolved conversations	$4,000 \times 65\%$	2,600
Outcome revenue	$2,600 \times \$0.99$	\$2,574
Platform fee		\$500
Total revenue		\$3,074
Provider cost	$4,000 \text{ attempts} \times \0.15	\$600
Margin	$(\$3,074 - \$600) \div \$3,074$	80%

The platform fee protects the early months. If the resolution rate starts at 40% during onboarding, outcome revenue falls to \$1,584 while cost stays at \$600, and the platform fee keeps margin at about 71%. Without it, margin would fall to 62%.

Plan C: credits with model-based burn

A developer tool sells 10,000 credits for \$100 each month, a credit price of \$0.01, with burn rates set for a 60% margin: 1 credit per task on a small model, 10 on a standard model, and 30 on a frontier model. Additional credits cost the same rate and are billed in arrears.

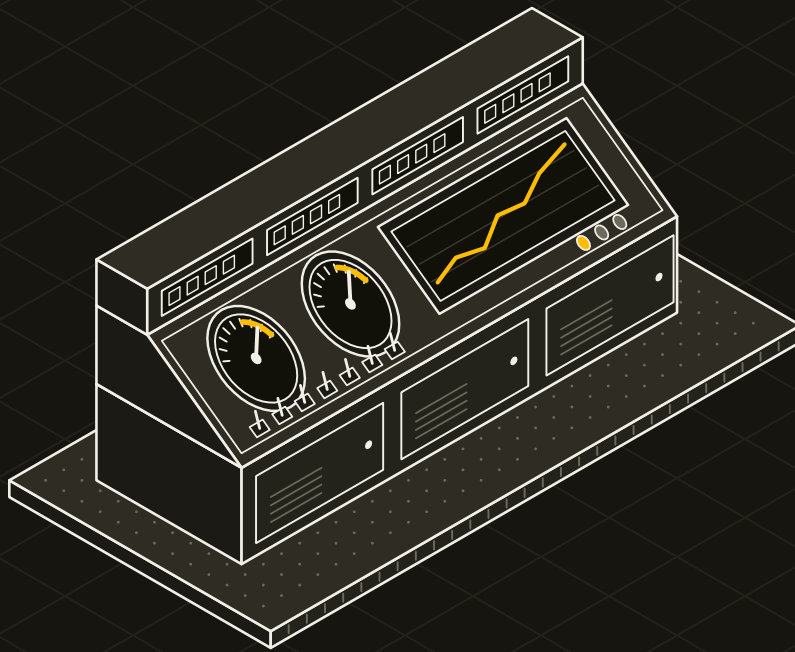
A customer who spends 2,000 credits on small-model tasks, 5,000 on standard-model tasks, and 3,000 on frontier-model tasks uses exactly their allowance. Their provider cost is $2,000 \times \$0.004 + 500 \times \$0.04 + 100 \times \$0.12$, which is \$40. The margin is 60%, the same as the target, whatever mix of models the customer chooses.

■ RERUN IT WITH YOUR NUMBERS

For any plan, you need four inputs per customer segment: price, usage, cost per unit at the median and 95th percentile, and the share of customers in each segment. The rest is arithmetic.

Running pricing as a system

Pricing keeps working only if you can see cost per customer, watch the right metrics, account for prepaid balances, and change prices without surprising anyone. This part covers each of those.



- 16 Attributing provider cost to customers
- 17 The margin metrics to watch
- 18 Accounting for prepaid credits
- 19 What your billing system needs to do
- 20 Repricing without losing trust

Attributing provider cost to customers

Everything in this ebook depends on one capability: knowing what each customer costs you, soon after they cost it.

Provider cost arrives as a monthly invoice from each model provider, at the level of your account. To manage margin, you need the same cost at the level of each customer, and ideally each agent, feature, and model. Cost attribution is the work of getting from one to the other.

Four ways to attribute cost

METHOD	HOW IT WORKS	ACCURACY	USE IT FOR
Direct metering	Record tokens, model, and cost on every call, tagged at the source	Highest	Model and tool costs that a customer's action triggers
Provider invoice allocation	Split each monthly invoice by each customer's share of usage	Medium	A starting point before you meter directly
Standard cost per unit	Set an expected cost per unit, then reconcile the variance to invoices	Medium	Forecasting and quoting
Shared cost allocation	Spread fixed infrastructure, such as vector databases or GPUs, by a usage driver	Lower	A fully loaded margin view

Most teams start with invoice allocation because it needs no new instrumentation. It misses the differences between models and the effect of caching, which are exactly the things that make one customer more expensive than another. Direct metering is the

method to build toward.

What a signal should record

Direct metering starts with the signal. Every time your agent does something that costs you money, such as calling a model or a paid tool, your product should record a few facts about it. Those facts let you price the event and tie it to the right customer.

WHAT TO RECORD	WHY YOU NEED IT
Who it was for	The customer, and ideally the agent and feature, so cost can be added up per customer
Which model ran	Models are priced very differently, so the model decides the cost
How much it used	Tokens in, tokens out, and cached tokens, which together set the price of the call
Whether it was a retry	Retries and loops are the cost driver that is hardest to spot any other way
What it cost	The provider cost, worked out from the usage and the model's price
A unique ID and a time	The ID stops an event sent twice from being counted twice, and the time puts it in the right billing period

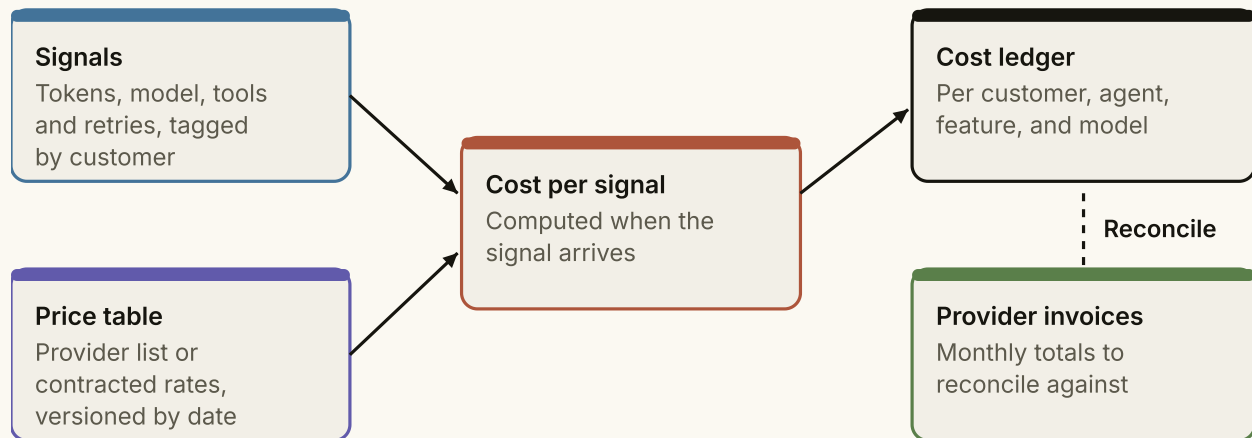
If you already record usage for your own debugging or analytics, much of this may exist. The gap is usually the customer tag and the cost, which is where the next step comes in.

Pricing each signal

Provider cost per signal comes from a price table: the list or contracted rate for each model, for input, cached input, and output, versioned by date. Keeping versions matters. When a provider changes prices, signals before the change should keep the old price, and signals after it should use the new one.

FIGURE 16.1

From signals to a cost ledger you can trust



Close the gap each month. A persistent difference of more than a few percent usually points to missing signals, unpriced models, or discounts the price table does not reflect.

Signals are priced as they arrive using a dated price table, summed into a cost ledger by customer, agent, feature, and model, and reconciled each month against provider invoices.

Reconciling with provider invoices

Once a month, compare the total in your cost ledger with the provider invoices. The two will rarely match exactly. Batch discounts, committed-use rates, free allowances, and rounding all create small differences. A difference that is large or growing usually means signals are missing, a model is not in the price table, or a discount has not been recorded.

Classifying the cost correctly

Attributed provider cost belongs in cost of goods sold when a customer's action triggered it. A useful rule from The SaaS Academy: if a customer triggered the API call, it is cost of goods sold. Your own team's use of AI tools belongs in operating expenses. Getting this right keeps your gross margin comparable with the benchmarks in Chapter 2.

▣ WHERE TO START

If you record nothing today, start by tagging every model call with a customer ID and a model name. With those two fields and your price table, you can compute a first cost per customer this week.

SOURCES FOR THIS CHAPTER

1. The SaaS Academy, *How AI changes SaaS gross margin*, thesaasacademy.com

The margin metrics to watch

A short list of metrics, reviewed on a regular schedule, catches most margin problems while they are still small.

Once cost is attributed to customers, you can measure margin at any level. The risk is measuring too much. The metrics below are the ones that change decisions. Each has a formula, a cadence, and a suggested threshold to start from, which you should adjust to your own targets.

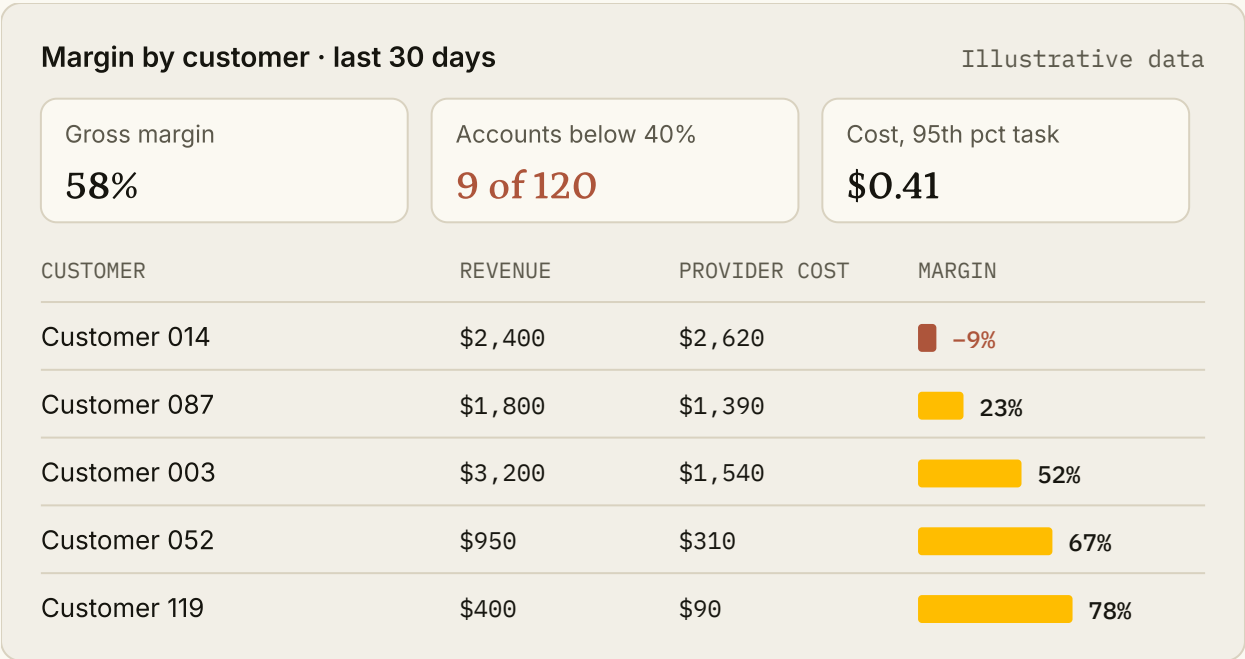
METRIC	FORMULA	REVIEW	START INVESTIGATING WHEN
Gross margin per customer	$(\text{Revenue} - \text{provider cost}) \div \text{revenue}$	Weekly	An account falls below your target
Share of accounts below target	$\text{Accounts below target} \div \text{all accounts}$	Monthly	The share rises two months in a row
Cost per unit, median and 95th percentile	From the cost ledger, per unit	Weekly	The 95th percentile rises faster than the median
Cost to revenue ratio per account	$\text{Provider cost} \div \text{revenue}$	Weekly	Above 0.5 for any account
Margin by model	Revenue attributed to requests on a model, less their cost	Monthly	A model's share of requests rises while its margin falls
Margin by agent or feature	Same, grouped by agent or feature	Monthly	A new feature launches below target
Overage revenue share	$\text{Overage revenue} \div \text{total revenue}$	Monthly	Rising fast, which may mean plans are too small
Credit balance and breakage	Unspent credits and expected expiry	Monthly	Balances grow faster than consumption

What a margin view looks like

The most useful view is a ranked list of customers by margin, with the least profitable first. It answers the question from Chapter 1, which customers are losing you money, in a single glance.

FIGURE 17.1

A margin view, least profitable customers first



Illustrative layout and data. Customer names are placeholders.

A review rhythm

- **Weekly:** margin per customer, cost per unit, and any account whose cost to revenue ratio has crossed your threshold.
- **Monthly:** reconciliation with provider invoices, margin by model, agent, and feature, and the share of accounts below target.
- **On every model or price change:** rerun cost per unit, credit burn rates, and floor prices before the change reaches customers.
- **Before every renewal:** the account's margin over the last two quarters and whether its plan still fits its usage.

■ WHO OWNS IT

Margin needs an owner, usually shared between finance and product. Finance owns the numbers. Product owns the levers, such as routing, limits, and burn rates, that change them.

Accounting for prepaid credits

Prepaid credits bring in cash before you deliver the service. How you account for that cash, and for credits that are never used, affects your reported revenue.

This chapter summarizes the main rules under ASC 606, the US revenue recognition standard, as they apply to prepaid credits and wallets. It is a summary for founders, not accounting advice. Confirm your own treatment with your accountant or auditor before you finalize your terms.

The core rules

RULE	WHAT IT SAYS, IN PLAIN TERMS
ASC 606-10-55-46	A prepayment for future services creates a contract liability. You recognize revenue as you deliver the services the credits pay for.
ASC 606-10-55-47	Rights a customer pays for and never uses are called breakage.
ASC 606-10-55-48	If you expect breakage, recognize it in proportion to the pattern of credits the customer does use, subject to the constraint on variable consideration. If you do not expect breakage, recognize it when the chance of the customer using the rest becomes remote.
ASC 606-10-55-49	Amounts you must pass to someone else, for example under unclaimed property laws, stay a liability.

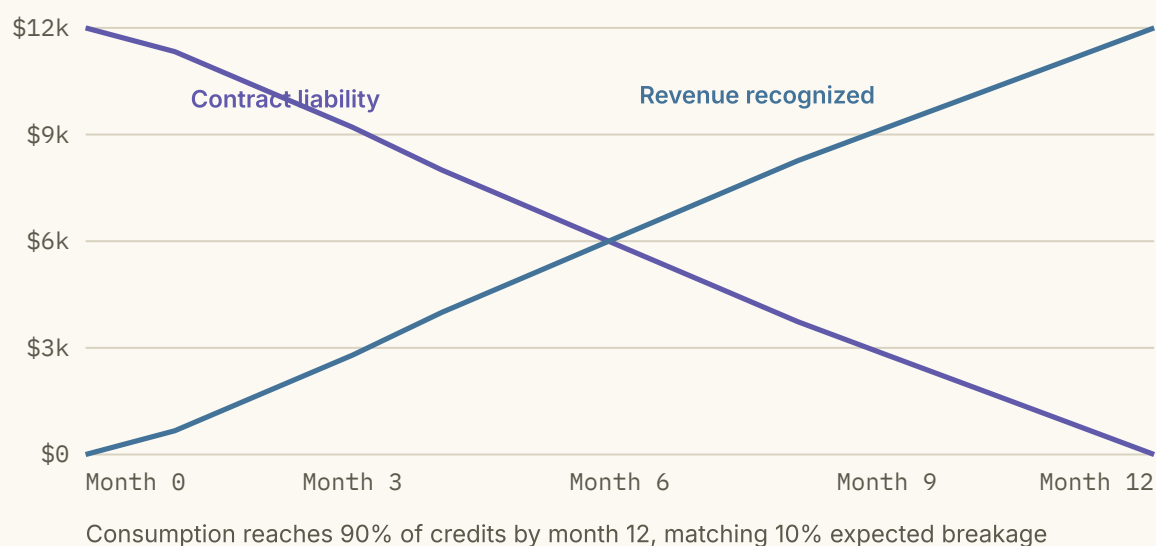
Public companies with consumption models show how this looks in practice. Snowflake's 2026 annual report says it recognizes revenue as customers consume capacity, and that many contracts allow unused capacity to roll over. Twilio has reported customer refundable prepayments as customer deposits, which are treated differently from nonrefundable prepayments.

A worked example

A customer prepays \$12,000 for 1.2 million nonrefundable credits, valid for 12 months. Your history with similar customers supports an expectation that 10% of credits will go unused. Under the proportional method, you expect 1.08 million credits to be used, so each credit consumed recognizes $\$12,000 \div 1.08 \text{ million}$, about \$0.0111 of revenue.

FIGURE 18.1

Revenue recognized from a \$12,000 prepayment



Illustrative and simplified. Revenue is recognized in proportion to credits consumed, including the expected breakage. By month 12, 90% of the credits have been used and the full \$12,000 has been recognized.

If the customer ends up using fewer credits than expected, you recognize more at expiry. If they use more than expected, you recognize less per credit going forward. Either way, the estimate needs evidence behind it and needs updating as your history grows.

How plan terms change the accounting

TERM	EFFECT ON ACCOUNTING
Credits are refundable	Treated as a deposit rather than revenue until used
Credits expire at period end	Breakage becomes relevant once expiry is known and history supports an estimate
Credits roll over	The liability carries into the next period
Burn rates can change	Consider how a rate change affects the services each credit pays for
Bonus credits on top of paid ones	The price paid may need to be allocated across paid and bonus credits

■ DECIDE TERMS WITH FINANCE PRESENT

Refund, expiry, and rollover terms are product decisions with accounting consequences. Agree on them with whoever keeps your books before you publish your pricing page.

SOURCES FOR THIS CHAPTER

1. FASB ASC 606-10-55-46 to 55-49, as summarized in Deloitte, *Roadmap: Revenue recognition*, section 8.8, [dart.deloitte.com](https://www.deloitte.com/dart/deloitte.com)
2. Snowflake Inc., Form 10-K for fiscal 2026, [sec.gov](https://www.sec.gov)
3. Twilio Inc., Form 10-K for fiscal 2021, [sec.gov](https://www.sec.gov)

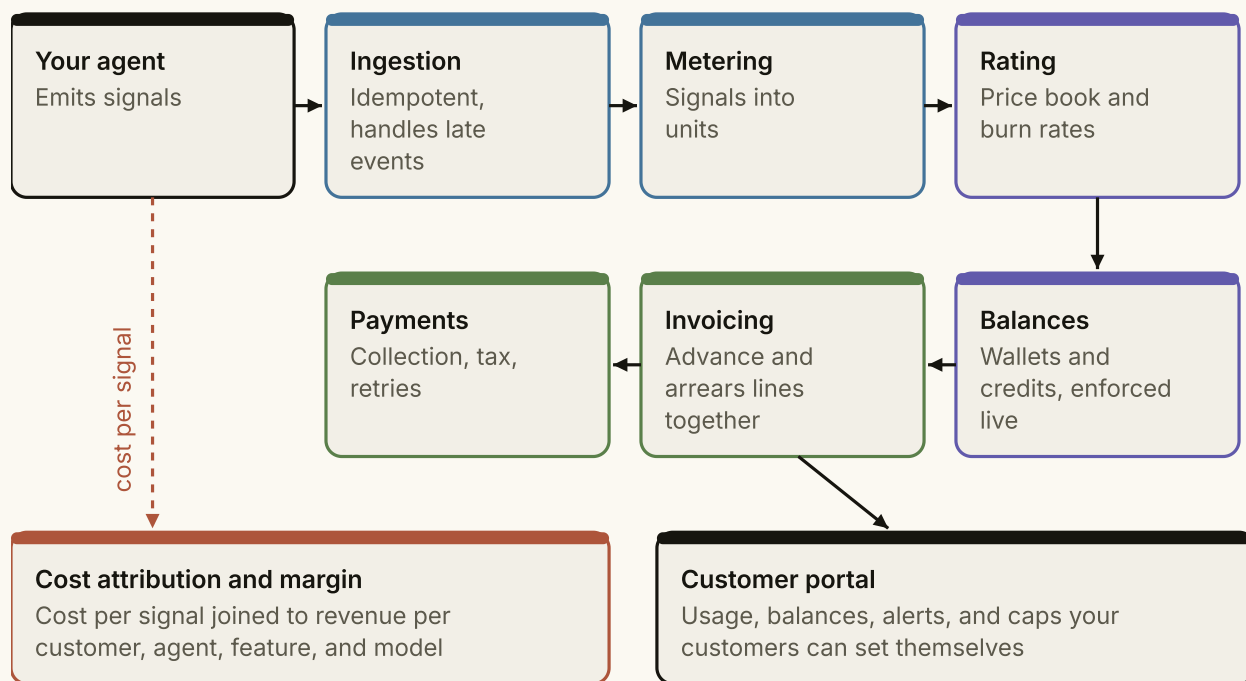
What your billing system needs to do

Whatever you build or adopt, an AI product's billing system has more to do than a SaaS one. These are the requirements that matter for margin.

A SaaS billing system mostly needs to charge a fixed price on a schedule. An AI product's billing system has to meter usage as it happens, price it by model, enforce balances in real time, combine billing in advance and in arrears on one invoice, and keep a record of cost alongside revenue. The list below is written to be useful whether your team builds the system, adopts one, or does some of each.

FIGURE 19.1

The parts of an AI billing system



A reference layout. Signals flow through ingestion, metering, and rating into balances, invoicing, and payments. Cost attribution runs alongside, and customers see usage and controls in a portal.

Requirements, and why each matters

REQUIREMENT	WHY IT MATTERS FOR MARGIN
Idempotent ingestion	A signal sent twice must be counted once, or you overbill customers and overstate cost
Late and out-of-order signals	Agents run asynchronously; signals that arrive after a period closes still need to land in the right period
Real-time balances	Wallets and credit allowances must be checked before work runs, so caps stop spend rather than report it
Rating by model and tier	Burn rates, multipliers, and tiers must be applied as signals are priced
Advance and arrears on one invoice	Platform fees and included credits billed ahead, overage and outcomes billed after, on the same bill
Plan versioning	Existing customers keep their terms when you change pricing, until you move them deliberately
Cost attribution	Provider cost joined to each signal, so margin can be computed per customer, agent, feature, and model
Reconciliation	Cost ledger checked against provider invoices each month
Revenue schedules	Prepaid balances recognized as consumed, with breakage handled as Chapter 18 describes
Customer portal and alerts	Customers see usage, balances, and costs, and set their own caps and alerts

Questions to ask your team

- When a provider changes prices, how long does it take to update the price table, and who does it?
- If we changed a burn rate tomorrow, which customers would be affected, and could we tell them before their next invoice?
- Can we see the margin on any single customer for last week without a spreadsheet?

- What happens to a customer's agent when their wallet reaches zero?
- How would we launch an outcome-based plan for one segment without changing the others?

The answers tell you where the system is strong and where pricing changes will be slow. A pricing strategy is only as flexible as the system that carries it out, so these questions are worth asking before you commit to a new plan structure.

■ SCOPE IT EARLY

Metering and cost attribution are the hardest parts to add later, because they depend on how your product emits signals. Design the signal record from Chapter 16 into the product from the start, even if the rest of the billing system comes later.

Repricing without losing trust

Every AI company will reprice more than once. The companies that keep their customers' trust do it early, explain it clearly, and give customers control.

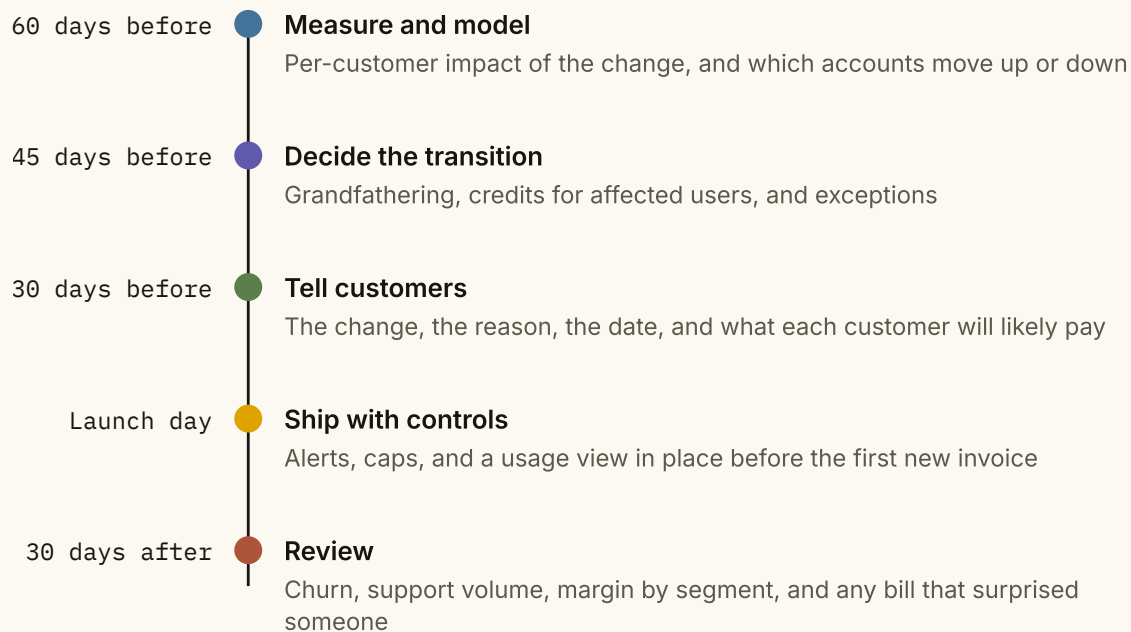
Provider prices, models, and customer behavior all change faster in AI than in traditional software, so your pricing will change too. The public examples in Chapter 5 show what goes well and what does not.

WHAT HAPPENED	WHAT IT TEACHES
Cursor's CEO apologized and offered refunds for unexpected charges within weeks of a pricing change	Customers need to understand a change before it appears on a bill
Anthropic announced weekly limits a month before they took effect and said they would affect fewer than 5% of subscribers	Notice and a clear statement of who is affected reduce alarm
Replit gave affected users \$10 in credits after moving to effort-based pricing	A small transition credit acknowledges the disruption
Salesforce offers four ways to buy Agentforce	Choice lets customers pick the trade-off they prefer

A repricing timeline

FIGURE 20.1

A repricing timeline



A suggested sequence. Adjust the lead times to your contract terms; annual contracts may need longer notice.

What to tell customers

- **What is changing**, in one or two sentences, with the new prices.
- **Why**, in plain terms. Customers understand that serving heavier use costs more.
- **When**, with a specific date.
- **What it means for them**, ideally an estimate of their own next bill under the new plan, based on their recent usage.
- **What they can do**, including the controls available and any plan that might suit them better.
- **Who to talk to**, with a real person for larger accounts.

Grandfathering and transitions

Keeping existing customers on their current terms for a period reduces friction, and it has a cost if those terms are the ones losing you money. A common middle path is to keep the current price for light users, who are profitable under either plan, and to move heavy users with notice and a transition credit. The per-customer margin view from Chapter 17 tells you which customers fall into which group.

■ THE BEST TIME TO REPRICE

Before your customers force you to. When you can see margin per customer, you can make smaller, earlier changes that are easier to explain than one large correction.

SOURCES FOR THIS CHAPTER

1. Cursor, *Clarifying our pricing*, cursor.com/blog
2. Coverage of Anthropic's weekly limits for Claude Pro and Max, OfficeChai and Apidog (July 2025)
3. Replit, *Effort-based pricing recap*, replit.com/blog
4. The SaaS CFO, *The death of per-seat pricing*, thesaascfo.com

What to take away

The short version of each part, for when you come back to this ebook later.

Part 1. Why agent economics are different

- Cost in an AI product moves with usage, with the model behind each request, and with the shape of each task.
- A blended gross margin can look healthy while your heaviest accounts lose money on every task.
- Falling token prices do not guarantee falling costs. Cost per task falls far more slowly than price per token.

Part 2. Where margin leaks

- Margin leaks through seven predictable gaps, from flat plans and unpriced free tiers to provider cost that arrives after the invoice.
- A small share of customers often carries most of your provider cost. Track cost to revenue per account every week.

Part 3. Choosing what to charge for

- Charge for a unit your customer understands and that you can measure and cost from your own signals.
- Outcome margin depends on two numbers: cost per attempt and resolution rate.
- Credits hold their margin when burn rates are set per model, from cost.

Part 4. Plan structures that protect margin

- Combine parts billed in advance with parts billed in arrears, so heavy use pays for itself.
- Guardrails such as spend caps, loop limits and customer alerts cap your downside.
- Set a cost floor under every unit, using the 95th percentile cost when nothing else limits it.

Part 5. Running pricing as a system

- Record provider cost on every signal, and reconcile it against provider invoices each month.
- Review margin per customer weekly and per model and feature monthly.
- Reprice early, with notice, a clear reason and controls customers can use.

Five questions to ask this week

1. Which customers earned less than your target margin last month?
2. What does one task cost you at the 95th percentile?
3. Which of your plans has no limit on the cost of a single customer?
4. What happens to your margin if your default model changes tomorrow?
5. Could you tell a customer what their next bill will be, and why?

See your margin on every customer

Clocknext is a billing and monetization layer for AI agent companies. It meters every signal your agents produce, prices it by model and plan, and keeps the provider cost of that signal next to what you bill for it. The ideas in this ebook become settings in your plans.

01 Margin on every customer

Provider cost is recorded on each signal and joined to what you bill, so you can see margin per customer, agent, feature and model.

02 Tailored pricing plans

Credits, outcomes, units, composites, wallets and flat fees combine into one plan, each priced the way this ebook describes.

03 Flexible billing system

Billing type is set per entitlement, so a platform fee, prepaid credits and usage-based overage can sit on the same invoice.

Take the leap with Clocknext

If you're trying to find the infrastructure to monetize your AI products, then you're in the right place. Talk to us, we'd love to help!