

Open models or commercial APIs

A decision guide for enterprise AI teams · September 2026

Table of contents

One workload at a time	3
What has changed	4
Open models belong on the shortlist	4
Test the endpoint, not the model name	5
Openness and buying are separate choices	5
Check the licence for the exact release	6
Decide what good enough means	6
Test the whole workflow	7
Compare fairly	7
Volume is only the start	8
Rule options out, then compare	9
Step 1. Remove anything that fails a must-have	9
Step 2. Compare what's left	10
The cost of a finished task	11
Cost per accepted task	11
Example 1. A lower token price can mean a higher bill	12
Example 2. Dedicated capacity pays off at high, steady volume	12
Example 3. Look at the whole process before picking a smaller model	13
Only pay for control you'll use	14
Follow the data	15
Make sovereignty testable	16
An Australian example	16
Pick a service you can keep running	17
Different workloads, different answers	18
A recommendation in 30 days	19
Decision worksheet	21
Sources and methodology	22
Next step	23

One workload at a time

Most enterprises won't make one model decision. They'll make dozens, one for each workload: a customer assistant, a document pipeline, an internal research tool. Each has its own tolerance for error, its own rules about where data can go and its own answer to who keeps it running. The right choice for one is often the wrong choice for the next.

There are three broad ways to buy models today. Commercial APIs give you quick access to strong models, and the provider runs the serving. Open models give you more say over where they run, which version you use and how you adapt them. Managed open-model services sit in between. Any of them can be right, depending on the workload, the contract and the team that has to run it.

This guide works through the choice for one workload in four steps. You should come out with a first choice and a realistic fallback.

01	Set the must-haves	Rule out anything that fails a hard requirement on data, licensing, quality or recovery.
02	Test on real work	Compare the options that are left on a sample of your own tasks.
03	Cost the finished task	Work out what one accepted outcome costs, retries and review included.
04	Decide who runs it	Name the team that owns capacity, incidents and upgrades.



Part 1

What has changed

A production AI system rarely makes a single model call anymore. It retrieves documents, calls tools, reasons over several steps and sometimes updates a record. One user question can turn into six billable steps, and the user only sees the last one. If you judge cost or quality by the final answer, you're looking at a small part of what happened.

Step	What happens	Billed as · Seen by user
Model call	Interpret the request and plan the steps	Input and output · No
Retrieval	Search approved documents	Search, index · No
Tool	Run a calculation	Tool session · No
Model call	Internal reasoning	Output · No
Action	Update the record	Authorised action · No
Final answer	The response	Output · Yes

Figure 1. One user request, traced. The answer is one step of six. Cost and risk sit across all of them.

Open models belong on the shortlist

Qwen3.8 publishes language, vision and coding evaluations. Mistral Small 4 takes image and text input and lets you set how hard it reasons. DeepSeek releases downloadable weights with guidance for running them locally.^{10 11}

These are serious options worth testing. The results come from the developers themselves, though, and they won't tell you how a model does on your tasks. Put proprietary models through the same business tests. If both types clear the bar comfortably, control over deployment and cost can reasonably decide it.

Test the endpoint, not the model name

A model name leaves a lot out: effort settings, quantization, context limits, prompts, retrieval and the software that handles tool calls. Independent benchmarks help you build a shortlist,¹ but what you deploy is a specific endpoint with specific settings. Reasoning adds another variable. OpenAI and Anthropic both charge for reasoning or thinking tokens, even when the visible answer is short.^{5 6} Compare the full usage records for the whole workflow.

For executives. You're paying for a business outcome under clear constraints. Use capability rankings to pick candidates, then measure quality, speed and total cost on your own work.



Part 2

Openness and buying are separate choices

Whether a model is open and how you pay for it are two different questions. An open model can be sold through a commercial API. A proprietary model may be available on more than one cloud. You can rent dedicated infrastructure without owning any hardware. Vendors often bundle model rights, deployment and commercial terms, but you should treat them as three separate decisions.

Option	What you get	What to watch
Proprietary API	Fast access to capable models. The provider runs serving.	The provider controls versions and features. You still own application quality and governance.
Managed open-model API	Open models without running the serving stack.	Access to the weights doesn't mean this service lets you customise them.
Dedicated managed open model	Control over deployment and capacity, with operational support.	Commitments and idle headroom cost money. Agree who does what.
Open model you run yourself	Direct control of versions, infrastructure and adaptation.	You need the capacity, security, reliability and upgrade skills to sustain it.

Figure 2. Moving down the list, you get more control and take on more of the running.

Check the licence for the exact release

Downloadable weights give you the trained parameters under whatever terms come with them. They don't give you the base model's intellectual property, the full training data or unlimited rights. The Open Source Initiative's definition covers data information, code and parameters, so "weights available" is a narrower claim.¹⁴

DeepSeek-V4-Pro	MIT
Qwen3.6-35B-A3B	Apache 2.0
Llama 4	Custom community licence

Terms vary from release to release. Procurement should review the specific artifact and how you plan to use it, instead of assuming one licence covers a whole lab or open models in general.^{9 10 12} Write down separately your rights to use, modify, redistribute and serve the model, and any conditions on derivatives or outputs. For a managed API, read the service terms as well as the model terms.



Part 3

Decide what good enough means

Start with the task and what happens when the model gets it wrong. A draft that an expert will check can carry more error than an automatic change to a customer's account. Before you test any model, agree on four things:

01	Correct	What counts as a correct result
02	Evidenced	What has to be backed by evidence
03	Escalate	What can be handed to a person
04	Never	What must never happen

Test the whole workflow

For extraction, check field accuracy and whether each document is complete. For support, check that answers are grounded, escalations go to the right place and cases get resolved. For an agent, check the state of the system afterwards, whether its actions were authorised and how it recovers when a tool call fails. An explanation that sounds right doesn't prove the workflow worked.

Build the test set from real work: common requests, rare but costly failures, long documents, the languages you actually use, vague inputs and hostile content. Keep your development cases apart from a held-out evaluation set.

Evidence	Good for	Limit
Independent benchmarks	Shortlisting and broad comparisons	The tests may not look like your tasks
Model and system cards	Reported behaviour, methods and limitations	A developer's disclosure doesn't certify your application
Your own evaluation	Measuring outcomes in the system you'll run	Results hold for the tasks and settings you tested

Figure 3. Three kinds of evidence. Only the last one measures your outcome.^{2 3 4}

Compare fairly

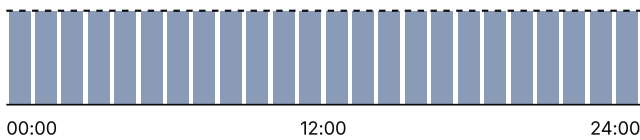
Give every candidate the same documented tuning allowance. Then lock the model version, endpoint, prompts, retrieval, effort, tool access and limits for the final run. Use blind human review where judgement matters and automated checks where you can. Report success rates, critical failures and uncertainty together. A small difference in score may not be worth a migration.

Speed counts toward acceptance. Measure the share of tasks that meet both the quality bar and the response-time target. A fast wrong answer fails, and so does a right answer that arrives too late. Don't report quality and speed from different samples.

Volume is only the start

Monthly token volume doesn't describe demand very well. The same total can arrive evenly across the month or in a rush every morning, and those two patterns need different headroom, queuing and contracts. Work out the shape of demand before you choose capacity.

Steady



Morning burst

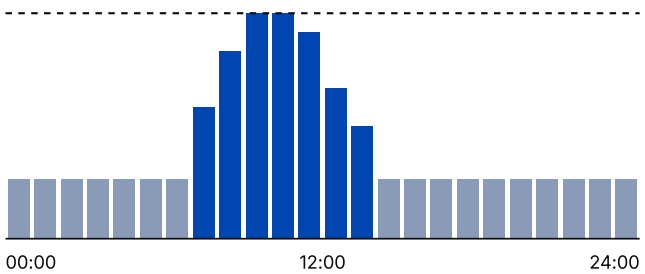


Figure 4. Two days with the same total demand, by hour. The dashed line is the capacity needed to serve the peak without a queue. The morning burst needs more than twice as much, unless some of the work can wait.

Describe demand in terms of volume, concurrency, bursts, context length, response-time target and how much work can wait. Each of these changes how much capacity you need. Then look at predictability, which comes in four kinds:

01	Demand	When work arrives
02	Cost	How calls, tokens and tools turn into spend
03	Latency	How response times spread out under load
04	Behaviour	How stable the outputs are across inputs, repeat runs and model changes

A fixed monthly bill for capacity solves only part of this.

Track median and 95th-percentile completion times, and count timeouts and rejected requests. Leave those out and an overloaded service can look fast. Model low, expected and high demand with clear adoption assumptions. Reserve capacity only when measured demand and your resilience needs justify it.

Capacity only helps if it gets acceptable work done in time. Your GPUs can be fully busy while customers sit in a long queue.



Part 5

Rule options out, then compare

Choosing takes two passes. First, drop any option that fails a requirement you can't compromise on. Then compare what's left on ten factors, weighted for the workload. You end up with a first choice and a credible fallback.

Step 1. Remove anything that fails a must-have

Some requirements aren't up for trade. A lower price doesn't make up for a use of data you aren't allowed, or for a quality failure that causes harm. Check every option against these five. If it fails one, it's out, whatever it costs.

01	Permitted use and licensing	The licence and terms allow this use
02	Data handling and location	Data is processed and stored where it has to be
03	Minimum quality and safety	The workflow meets the lowest acceptable standard
04	Service and recovery	You can meet availability and recovery targets
05	Access and audit	You control who has access and can review what happened

For each check, record the evidence, who gave it to you and what it covers. It has to apply to the exact model, deployment and use you're proposing. If you don't know the answer, treat it as an open question, not a pass.

Step 2. Compare what's left

Score the remaining options on the ten factors below. Set the weights before you see any results, so they reflect what the workload needs and not the answer someone was hoping for. A regulated claims process might weight assurance and control most. An exploratory internal assistant might weight capability and speed to value.

01	Capability	Does the whole workflow meet the acceptance criteria?
02	Volume	What work must the service handle now and at forecast demand?
03	Predictability	How stable are demand, cost, latency and behaviour?
04	Data exposure	What information and obligations are involved?
05	Sovereignty	Who can access, operate, change or interrupt the service?
06	Model control	Which changes, versions and adaptations do you need to control?
07	Insight and assurance	What can you inspect, test and prove?
08	Concentration and exit	Could an alternative take over?
09	Operational capability	Who keeps security, performance and recovery going?
10	Economics and delivery	What does accepted work cost, and when does value start?

Record how confident you are in each score as well as the score itself, and mark each one confirmed, indicative or unverified. If the winner only wins because of a few unverified assumptions, test those before you commit.



Part 6

The cost of a finished task

A token price is the rate for one billing category. A request cost is what one API call costs. A task cost adds up every call and supporting service needed for one business outcome. Mixing these up is how a cheap-looking service ends up expensive to run.

For each request, multiply each billed token category (uncached input, cache reads, cache writes and billed output) by its rate, then add any tool, request or session charges. Follow each provider's accounting rules. Output totals may already include reasoning tokens, and adding the breakdown again would count them twice.^{5 6 7 8}

Cost layer	What to include
Model usage	Every billable input, cache and output category across every call
Workflow services	Search, retrieval, embeddings, reranking, storage, code execution and tools
Failures and exceptions	Retries, fallbacks, failed attempts, human review and rework
Running costs	The share of engineering, monitoring, security, support and evaluation
Commercial terms	Commitments, unused entitlements, discounts, region and tier terms

Cost per accepted task

Total cost across all attempts, failed ones included, divided by the number of accepted business tasks, each counted once.

Agree what "accepted" means before you test. Count each finished task once, even if it took several calls or retries. Report cost per attempted task, acceptance rate, critical failures and review rate next to it, so a good unit cost can't hide a system that rejects too much work.

Count application and governance work on both sides. An API doesn't remove integration or evaluation work, and a private deployment isn't free once the hardware is paid for. Keep one-off implementation costs separate from recurring ones.

Example 1. A lower token price can mean a higher bill

Two hypothetical endpoints do the same text task. Each gets 10,000 uncached input tokens and returns 1,000 visible answer tokens. The cheaper endpoint reasons for longer, and all reasoning is billed as output.

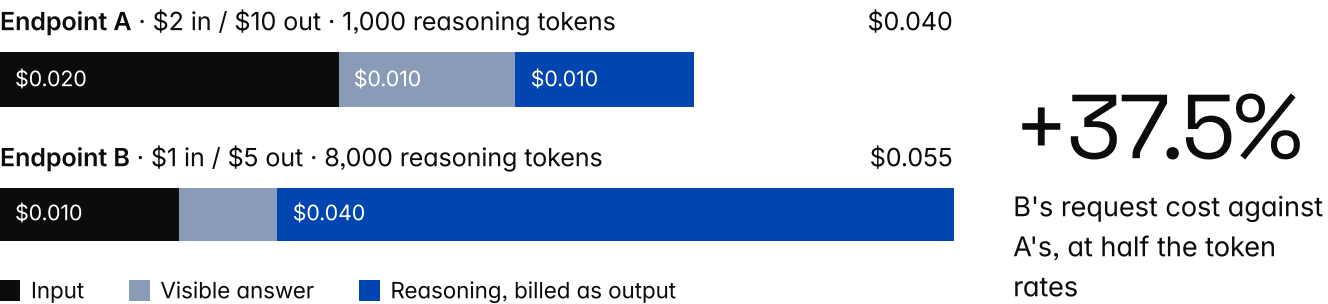


Figure 5. Cost per request by billing category. A = (10,000 × \$2 + 2,000 × \$10) / 1M. B = (10,000 × \$1 + 9,000 × \$5) / 1M. Illustrative rates only.

Now say 100,000 tasks are each attempted once. A gets 95,000 accepted and B gets 90,000. Request spend is \$4,000 for A and \$5,500 for B, so the model cost per accepted task is about \$0.0421 for A and \$0.0611 for B. Compare billed input and output on the endpoint you'd actually use, not the length of the answer on screen.

Example 2. Dedicated capacity pays off at high, steady volume

The two options bill differently. An API charges for usage, so cost starts at zero and rises with every task. Dedicated capacity is mostly a fixed monthly cost for reserved GPUs, plus a small cost per task.

API	No fixed cost. Calls and tokens average \$0.15 (example) per accepted task.
Dedicated	\$25,000 a month for reserved GPUs, hosting and the team that runs them, plus \$0.01 per accepted task for the things that still grow with volume, such as logging, storage and supporting services.

Assume quality is the same and dedicated capacity has been tested at one million accepted tasks a month. At low volume the API is cheaper because you haven't committed to anything. As volume grows, the lower per-task cost of dedicated capacity pays back the fixed cost.

Here break-even is \$25,000 / (\$0.15 – \$0.01), about 178,571 accepted tasks a month. Burstier traffic, longer contexts or stricter recovery targets can all cut usable capacity. A fixed monthly invoice makes the budget predictable. It doesn't give you unlimited throughput.

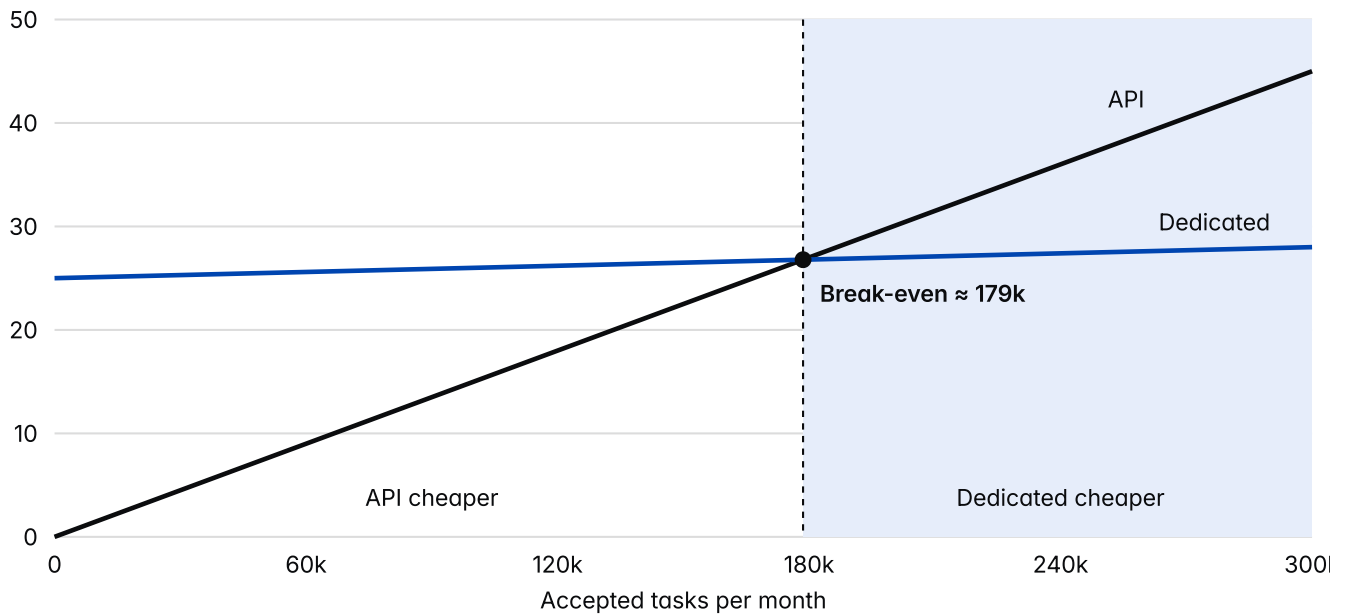


Figure 6. Monthly cost in USD thousands against accepted tasks per month, within tested capacity. A hypothetical scenario, not a measured comparison of products.

Example 3. Look at the whole process before picking a smaller model

Effort settings change how much a model reasons, and that changes quality, output volume and latency.^{5 6} Test ordinary, hard and ambiguous cases at several settings, and cap the calls, time and spend per task. Routing easy cases to a cheaper setup helps if you can reliably tell which cases are easy. A model saying it's confident isn't enough evidence to accept its answer.

Take 100,000 documents. Both workflows below end with 98,000 accepted after review. Each reviewed document takes three minutes at \$60 an hour, which is \$3 a document.

Workflow A · 20% reviewed · \$0.633 per accepted document \$62,000



Workflow B · 5% reviewed · \$0.204 per accepted document \$20,000



■ Inference (\$2,000 vs \$5,000) ■ Review labour

-\$42,000

B against A, per month,
inference plus review

Figure 7. Spending more on inference removed most of the review cost. All inputs are hypothetical.

This shows process cost, not guaranteed cash savings. Less review time may free people up without reducing payroll, and a lower review rate only helps if acceptance and critical-error standards still hold.



Part 7

Only pay for control you’ll use

Control is worth having when it changes an outcome. Pinning a version can protect a validated process from a surprise update. An approved adaptation can make a model better at a specialist task. Either way, you need people, evidence and a process before it's worth anything.

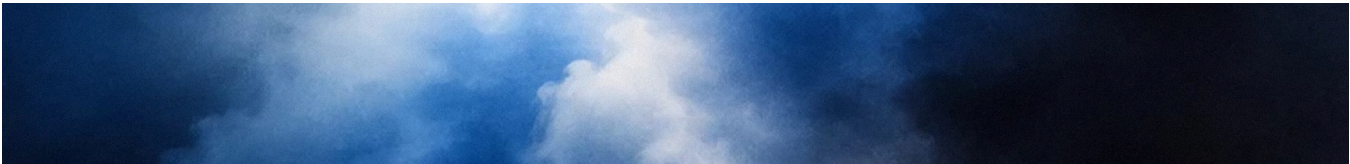
Record rights separately for the base model, customer inputs, generated outputs, fine-tuned weights or adapters, and the application around them. Having a downloaded copy doesn't make you the owner of the original intellectual property. If a provider hosts your fine-tuning, check whether you can export the result and use it elsewhere.

Layer	What to ask for	What you still need to find out
Behaviour	Published evaluations and known limitations	Whether the results hold for your tasks
Operation	Request traces, usage and version information	Whether the records are complete enough to audit
Artifacts	Weights, architecture and permitted changes	How to validate a modified deployment safely
Training	Data information, code and training artifacts	What you can actually trace or reproduce

Figure 8. Transparency comes in layers. Ai2's Olmo 3 is an example at the training layer: it publishes data, code and intermediate artifacts.¹³

Access to the weights doesn't explain any individual answer. Decide what assurance you actually need: tracing sources, reproducing results, independent testing, or an explanation the affected person can understand.

Customise for a reason. Try better task design, retrieval and prompts first. Fine-tune when there's a specific gap in behaviour, you have the rights to the data, you have a held-out test and someone owns the adapted model. Control you can't maintain adds risk.



Part 8

Follow the data

A regional endpoint covers only part of the system. Customer data can also end up in retrieval indexes, tool calls, application traces, caches, support tickets and backups. Review the path the workload really takes, including failures and escalations.

01	Application	User inputs and documents
02	Gateway and model	Routing, inference and cache
03	Retrieval and tools	Indexes and external services
04	Logs and monitoring	Payloads, traces and metrics
05	Backups and state	Saved files and recovery
06	Support and operations	People and subprocessors

Figure 9. Where data goes. Check location, access, retention and deletion at every stop.

OpenAI's documentation separates regional storage from regional processing. Anthropic separates inference geography from workspace geography.^{15 16} Check where data is stored, where it's processed and what each control actually covers, one at a time.

Requirement	The question that makes it testable
Residency	Where may each type of information be stored, backups and caches included?
Processing location	Where may inference, preprocessing and tool execution happen?
Retention	How long may payloads, state and logs be kept, and what are the exceptions?
Access	Which administrators, support staff and subprocessors can see the data?

Training use	Can inputs or outputs be used to train or improve any model?
Deletion	What gets deleted, on what schedule, and what proof do you get?

No-training commitments, retention controls and residency controls answer different questions. Put managed open-model APIs through the same review as proprietary APIs. Whether a model is open tells you nothing about how the provider handles your data.

Make sovereignty testable

Sovereignty is about who has authority over the service as well as where it runs. A local data centre can meet one location requirement and leave every other legal and operational dependency where it was. Look at four kinds of control:

01	Legal	Contracting entities, jurisdictions, subprocessors and how legal requests are handled
02	Operational	Who administers the service and can push changes
03	Technical	Access boundaries, network routes, keys and audit evidence
04	Continuity	Whether the service keeps running if a provider or connection goes down

Open weights can help with continuity, because an approved deployment can keep running without sending every request back to the original developer. That only holds if you have working infrastructure, complete artifacts, compatible software and a team to run it.

An Australian example

The OAIC's guidance on AI products recommends due diligence on intended use, testing, human oversight, security and access to personal information.¹⁷ For entities covered by APRA's CPS 230, reliance on service providers and concentration risk, including geographic and downstream dependencies, are explicit concerns. CPS 230 doesn't apply to every enterprise that buys AI.¹⁸

For each workload, keep an obligations register. List the jurisdiction, sector obligations, contract restrictions, internal policy, the data involved, the action the AI supports, the control required and the person who reviews it. Don't assume all personal data has to stay in the country, or that local hosting makes you compliant on its own. Procurement documents should spell out the required controls, locations, exceptions, audit rights and remedies, rather than lean on the word "sovereign".

A quick test. Could your security, legal and operations teams each explain, without comparing notes, what the sovereignty claim covers, what it leaves out and how you'd check it?



Part 9

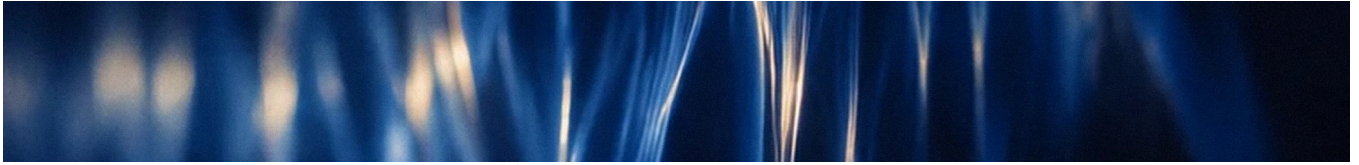
Pick a service you can keep running

Handing serving to a provider can get you live sooner and cut infrastructure work. You're still accountable for the application. Running open weights yourself gives you more control, and you also take on the serving stack, capacity planning and releases.

Responsibility	Managed service	Run it yourself
Serving capacity	The provider runs it within contracted limits. You forecast demand.	You provision, test and scale it
Model availability	The provider publishes supported models and lifecycle terms	You keep and maintain the permitted artifacts
Application quality	You evaluate prompts, retrieval, actions and outcomes	The same, plus any serving changes
Security boundary	Shared, as set out in the contract and architecture	You secure the wider infrastructure
Incidents and recovery	Joint escalation. Check the provider's commitments and fallbacks.	You handle infrastructure and application response, and test restores

Name the team that will handle incidents, security patches, model upgrades and cost overruns. One specialist who can get a model running isn't an operating capability. For private deployments, measure precision, context length, batching and concurrency under load. The active parameter count of a mixture-of-experts model won't tell you its memory footprint.^{9 10}

Treat any change to the model, prompts, retrieval corpus, tools, serving engine or quantization as something that may need re-testing, and keep a previous configuration you can roll back to. An agent that can spend money, change records or share information needs limits on what it's allowed to do, an audit trail and an escalation path, wherever its model runs.



Part 10

Different workloads, different answers

Take a hypothetical mid-sized enterprise with three production proposals and a small central AI team. These are decision hypotheses, not customer case studies. Each answer still depends on the must-haves and the test results.

01	Customer support with uneven demand	Start with a qualifying commercial API. Traffic is hard to predict and the team wants to launch quickly, so an API avoids committing to infrastructure. Test proprietary and managed open-model endpoints on grounded answers, correct escalation and response time during spikes. Look at dedicated capacity again if demand settles down.
02	A recurring document pipeline	Try an open model on managed dedicated infrastructure. Steady volume, fields you can check automatically and an overnight queue suit this setup, as long as it meets the document-level bar and lowers the total process cost.
03	Sensitive internal research	First, work out which deployments are allowed. Controlled data paths and evidence for conclusions come first. Evaluate private open models and any regional commercial services that qualify. If the preferred private model misses the quality bar, narrow the task, add review or hold off on automating it. Hosting it locally doesn't make it fit for purpose.

The organisation can end up with a mix of approaches without running three governance systems. Keep one workload register, one evaluation process, one cost ledger and one access policy, and let deployment choices differ where the evidence supports it.



A recommendation in 30 days

A focused evaluation can produce an investment recommendation within a month, if the workload, data access and reviewers are ready. Complex procurement or regulated validation can take longer. Treat the schedule as a plan for a decision package, not a promise that you'll be in production.

Week	Work	Output
1 · Define	Choose a business task, map the data, set acceptance and service thresholds, document the current process	Workload brief, must-haves and baseline
2 · Evaluate	Select eligible configurations, tune within an agreed allowance, lock the settings, test on held-out work	Quality results, critical failures and confidence
3 · Operate	Run load, failure and recovery tests, capture usage, calculate cost, inspect access and logs	Cost scenarios, operating plan and control evidence
4 · Decide	Review terms and rights, test an alternative, close the gaps, get named sign-off	Recommendation, conditions, owners and triggers

Figure 10. Four weeks, four decision documents.

Keep the evidence reproducible. For every configuration you test, record the date, model identifier or artifact hash, endpoint, region, serving version, precision, prompts, retrieval setup, effort, output limits, tool access and retry policy. Sensitive evaluation traces need their own retention and access controls.

Agree what has to be true before the first production user gets access:

01 **Quality and service accepted**

02 **No open must-have failures**

03 **A named incident owner**

04 **Budget limits enforced**

05 **A configuration you can roll back to**

06 **A reviewed contract**

Then roll out to a limited group and track acceptance, review time, tail latency and spend by task type.

Agree too on what would trigger another evaluation: a big change in demand, rising review rates, a retired model, new data obligations, a new tool with wider access, or an alternative that looks clearly better. A recommendation you can defend says when it stops being valid.

Decision worksheet

Fill in one per business task and keep it with the evaluation results.

Workload brief

Business task and accountable owner

Acceptance criteria and critical failures

Data classes and required locations

Monthly demand, bursts and growth range

Completion and recovery targets

Current process cost and review time

Scores, eligible options only

Factor	Weight	Option A	Option B	Evidence
Capability				
Volume				
Predictability				
Data exposure				
Sovereignty				
Model control				
Insight and assurance				
Concentration and exit				
Operational capability				
Economics and delivery				

Weights add up to 100%. Score 1 for weak, 3 for adequate and 5 for strong, with a reason based on evidence. Mark missing evidence as unverified. If a small change in weights flips the result, explain the trade-off.

Questions for each provider or operating team

1. Delivered service. Which exact model, configuration and endpoint? What can change without your approval?

3. Data use. What are the training, retention and deletion terms, and the exceptions?

5. Performance and price. At what load does the quoted performance hold? Which charges, tiers and overages apply?

2. Data path. Where do inference, tools, logs, state and backups run? Who can access them?

4. Rights and portability. What rights apply to inputs, outputs, weights and adaptations? What can you export?

6. Assurance and recovery. What audit evidence and recovery commitments exist? Can the fallback meet the same requirements?

The recommendation. "Go ahead with this configuration for this workload, within these data and action limits, because it meets the recorded thresholds at the expected running cost. Review it if the named assumptions about demand, quality, contract or control change."

Sources and methodology

This guide draws on independent benchmark methods, model and system documentation from developers, provider documentation and Australian regulatory guidance. The framework and recommendations are our own analysis, and the cost scenarios are hypothetical. Pricing and residency terms change, so recheck them for the exact product, endpoint, region and contract before you buy.

- 1 Artificial Analysis. Model and API comparison. Independent comparison of capability, task cost, speed and endpoint accuracy. Live data.
- 2 Artificial Analysis. Intelligence benchmarking methodology. Evaluation coverage and limits of aggregate scores.
- 3 OpenAI. GPT-6 Astra system card. Developer disclosure of evaluations, safeguards and limitations. September 2026.
- 4 Anthropic. Claude Opus 4.1 system card addendum. August 2025, updated September 2025. Example of safety evaluation categories.
- 5 OpenAI. Reasoning models. Reasoning-token accounting, output limits and incomplete responses.
- 6 Anthropic. Steering thinking. Adaptive thinking, effort controls and billing for thinking tokens.
- 7 OpenAI. API pricing. Live rates and billing categories.
- 8 Anthropic. Claude Platform pricing. Live rates, cache pricing and additional services.
- 9 Qwen. Qwen3.6-35B-A3B model card. Architecture, evaluation settings, Apache 2.0 licensing and serving considerations.
- 10 DeepSeek. DeepSeek-V4-Pro model card. Capabilities, local deployment and MIT licensing.
- 11 Mistral AI. Introducing Mistral Small 4. March 2026. Multimodal inputs, configurable reasoning effort, Apache 2.0.
- 12 Meta. Llama 4 Maverick model card and license. Release-specific documentation and custom community license.
- 13 Ai2. Olmo 3 research release. Weights, training data, code and intermediate artifacts.
- 14 Open Source Initiative. Open Source AI Definition 1.0. Requirements for data information, code and parameters.
- 15 OpenAI. Data controls in the OpenAI platform. Retention controls, regional storage and processing, and exclusions.
- 16 Anthropic. Data residency. Inference geography and workspace geography.
- 17 OAIC. Privacy and commercially available AI products. Australian privacy guidance on product due diligence and ongoing assurance.
- 18 APRA. CPS 230 Operational Risk Management. Operational resilience and service-provider risk for applicable entities.
- 19 Xinference. Platform and deployment overview. Publisher's description of its Model API and dedicated inference offerings.

Next step

Test one workload with Xinference

Xinference offers a Model API and dedicated inference for open models.¹⁹ Use the framework in this guide to compare it with what you run today: the same task, agreed acceptance criteria, realistic traffic and the full cost.

Bring three things

01 **A representative set of tasks**

02 **Your data requirements**

03 **A rough baseline cost**

You'll leave with a written deployment recommendation covering measured quality, response time, total cost and anything still unresolved.

To request a workload discussion, visit xinference.co.

Published by Xinference, September 2026. This guide supports technical and commercial evaluation. Legal and regulatory obligations need to be assessed for your organisation and intended use.

