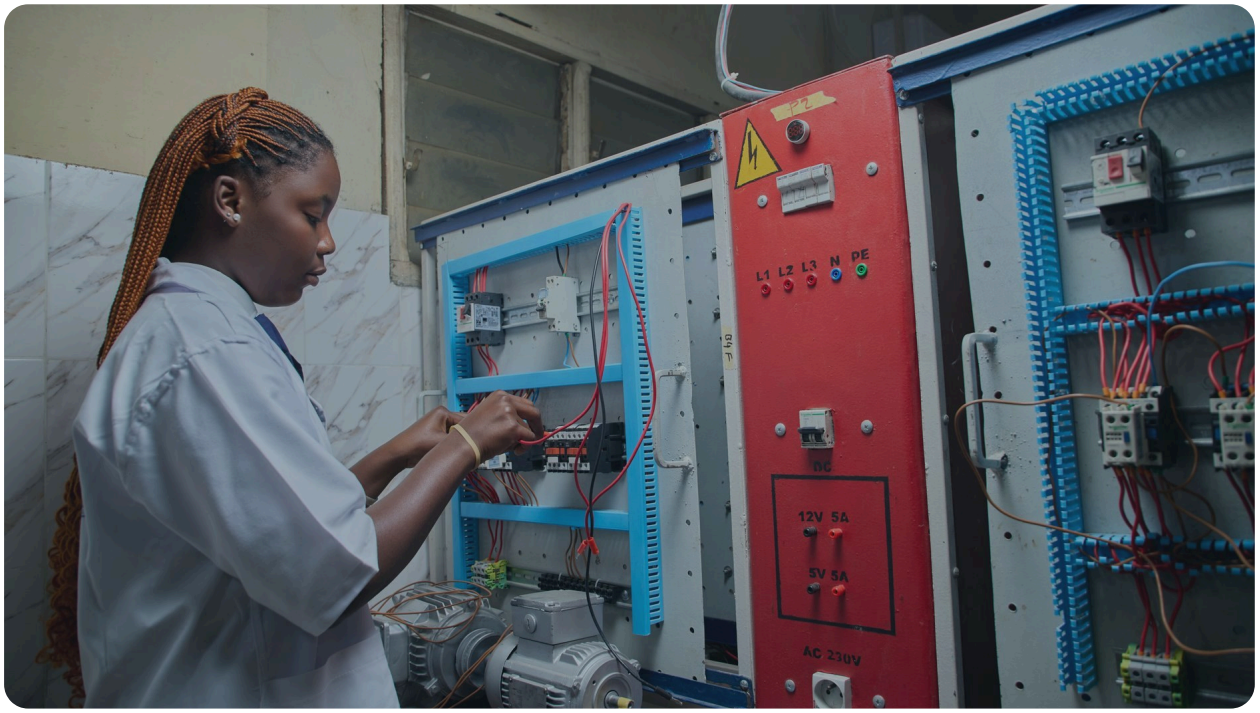
JULY 10, 2026

Don't marry one model: how to avoid AI vendor lock-in

Any model you build on can be retired, repriced, or restricted — often for reasons nothing to do with you. Portability is a design choice you make up front.

By **Dave Taylor**



You avoid AI vendor lock-in by owning your side of the stack — the prompts, the task data, and the way you measure a good answer — and treating the model as a swappable part behind it. That matters because the model you build on today can be pulled out from under you. Anthropic's own docs already list Claude Opus 4.1 for retirement on 5 August 2026, and that's the routine case, not the crisis one. When you wire an AI tool straight into a single provider's specific model, you've made that provider's roadmap your roadmap. The fix is a design decision you make at the start, not a rescue after the deprecation email lands. Stay free to move, and keep the cost of moving small.

What happens when the model you built on disappears?

Most teams find out the hard way: the tool that worked last quarter starts returning errors, and the reason is a model ID that no longer resolves. Providers retire models on a schedule. Anthropic publishes a models overview with dated retirement timelines — a production model gets a retirement date, and after it the API stops serving that version. This isn't a provider behaving badly. It's how the whole industry runs, because keeping every old model live forever is expensive and every provider does the same thing.

The trouble is what you built on top. If your prompts, your output formatting, and your quality bar were all tuned to one specific model's behaviour, retirement isn't a config change — it's a rebuild. You re-test everything, you re-tune the prompts, you re-check the outputs, and you do it under a deadline someone else set. That deadline is the part that hurts: a scheduled retirement gives you months of notice, but the notice doesn't do the work, and the work always lands on top of whatever else the quarter already held. A firm that treated the model as permanent pays that bill in full, usually at the worst possible moment. A firm that treated it as swappable pays almost nothing and barely notices the date go by. The difference isn't luck or better forecasting — nobody predicted the exact retirement date and nobody had to. It's whether the moving parts were separated from the fixed ones on day one, when it cost an afternoon instead of a fire drill.

Three ways a model gets pulled out from under you

There are three distinct ways the ground moves, and they need different defences. Naming them is the first step to designing around them.

The first is deprecation — the model is simply retired. This is the routine one, dated in advance, and Opus 4.1's 5 August 2026 retirement is the clean example. You get notice, but notice doesn't rebuild your tool for you. The second is repricing — the economics change while you sit still. A model that was cheap enough to run on every ticket becomes a model you have to ration, or a cheaper competitor makes yours look expensive overnight. Your tool still works; it just stops paying back. We've dug into that dynamic in why AI token costs keep rising, and it's the vector teams underprepare for most, because nothing breaks — the invoice just grows.

The third way is the one nobody plans for: access or policy change, where the model becomes unavailable for reasons that have nothing to do with your business. In June 2026, the US government applied export controls to Anthropic's two newest models — Claude Fable 5 and Claude Mythos 5 — on 12 June. Anthropic added a new safety classifier, and the controls were lifted by 1 July, an episode the company documented in its note on

redeploying Fable 5. Worth being precise here: that early government scrutiny was narrowly scoped to frontier, national-security-relevant models — it is not a blanket pre-release arrangement across every model a provider ships. But the lesson generalises. A model you depend on can go dark on a policy timeline, a regional restriction, or an account suspension, none of which appear on the provider's product roadmap. If your tool has exactly one model it can call, any one of these three takes it offline.

Portability is a design choice, not a vendor promise

No provider is going to hand you portability — it runs against their interest, and you shouldn't expect it. Lock-in is the business model. The switching cost you feel when a model retires is a feature from the vendor's side of the table, not a bug, and every provider builds their tooling to make their own models the path of least resistance. If you want the freedom to move, you design it in yourself, because the only place it can come from is your own architecture.

The mechanism is a swappable layer between your tool and the model. Your application talks to that layer in a standard way; the layer decides which provider and which model actually handles the call. Swap the target, and every tool downstream keeps working — the same pattern that lets you pick deliberately in the first place, which we walk through in how to choose an AI model. This isn't exotic engineering. It's the same discipline you'd apply to any critical supplier: you don't hard-wire your invoicing to one payment processor's proprietary calls if you can route through an abstraction that lets you switch banks without rewriting the ledger. The model is a supplier. Treat it like one. The abstraction costs a little more to build once, and it turns every future model change from a project into a setting — which is exactly the trade a senior operator should want to make.

What owning your side of the stack looks like

Three things have to live with you, not with the vendor, for portability to be real. Get these wrong and the swappable layer is theatre.

The first is your prompts and system instructions — kept in your own repository, version-controlled, written to be as model-neutral as you can manage, so re-pointing them at a new model is a tuning pass, not a blank page. The second is your task data: the real inputs your tool handles, and the outputs you expect back. That's your record of what "working" means, and it belongs to you regardless of which model produced it. The third — the one most teams skip — is an evaluation set: a fixed batch of representative cases with known-good answers you can run against any candidate model. Without it, "does the new model work?" is a gut feeling. With it, it's a number you get in an afternoon.

Those three assets are what make a swap cheap. When a model retires or reprices, you point the layer at a candidate, run your eval set, read the score, and decide. No re-discovery, no guessing, no rebuild. The model sits behind the layer as the one genuinely replaceable part, and the parts that took real work to get right — the prompts, the data, the definition of good — never move. This is also why single-platform bets bite hardest at the product level, not just the model level: when OpenAI discontinued its Sora product in 2026 (web and app on 26 April, API on 24 September), analysts at the Futurum Group framed it as exposing "the fragility of relying on a single AI platform" — a whole capability, not just one model version, gone on the vendor's timeline.

What this means for an SMB

For a small or mid-sized business, the risk isn't abstract — it's a tool your team now depends on going quiet with no warning you controlled. You don't have a platform team to absorb a surprise migration, and you can't afford a week of downtime on a workflow that's become load-bearing. That's exactly why CxD builds client automations to be provider-agnostic from the start. The prompts, the task data, and the evaluation set belong to the client, and the model sits behind a swappable layer — so a deprecation or a price change is a config change, not a rebuild you have to fund out of nowhere.

It's the same readiness discipline we run before pointing any tool at live data. You map what the tool actually needs, you define what a good answer looks like, and you keep the pieces that matter under your own control — so the model becomes the cheapest, most replaceable part of the system instead of its single point of failure. The ownership question is the whole game here: if the prompts live in a vendor's console and the "good answer" definition lives in one engineer's head, you don't own portability no matter what the architecture diagram claims. The readiness work we do with firms is what makes that separation real rather than aspirational — it's where the prompts, the data, and the eval set move under the client's control. When a client's model retires — and one always does — the fix is a line in a config file and a quick eval run, not a call to explain why the tool that ran the business last week just stopped.

Where this leaves you

The model is the part you can't control and shouldn't try to. Providers will retire, reprice, and occasionally restrict access on timelines they set — Opus 4.1's 5 August 2026 retirement is only the dated, polite version of a thing that happens constantly. What you can control is your side: prompts, task data, and an evaluation set you own, with the model sitting behind

a swappable layer so any change downstream is a setting, not a salvage job. Build it that way once and every future model change stops being an emergency. Don't marry one model. Keep the freedom to move, and keep the cost of moving small.

If you're running an AI tool your business now depends on — and you're not sure how hard it would be to switch models if you had to — book a free AI consult and we'll look at where you're locked in and what it would take to get free.