ARCHITECTURE NOTE · APPLIED AI PLATFORMS

Model Selection Is a Tiering Problem

Why the right question is not which model is best, but how many steps the task takes.

Chandrakanth Anne · Enterprise Applications Architect · September 2026

Most teams ask “which model should we use?” and treat the answer as a procurement decision — evaluate a few, pick a winner, standardise.

It is the wrong shape of question. You already know this if you have ever designed a storage estate.

Nobody puts every workload on the fastest tier. You tier by access pattern: hot data on flash, warm on disk, cold on object, and the architecture is the routing between them. Putting everything on the premium tier is not safe it is just expensive, and it hides the fact that nobody profiled the workload.

Model selection is the same problem. And the routing key turns out to be something more specific than “how hard is this task.”

What a “mini” model actually is

Every vendor ships a small, cheap, fast variant alongside its flagship. The names differ mini, Flash, Lite, Haiku but the engineering is the same three levers, usually applied together.

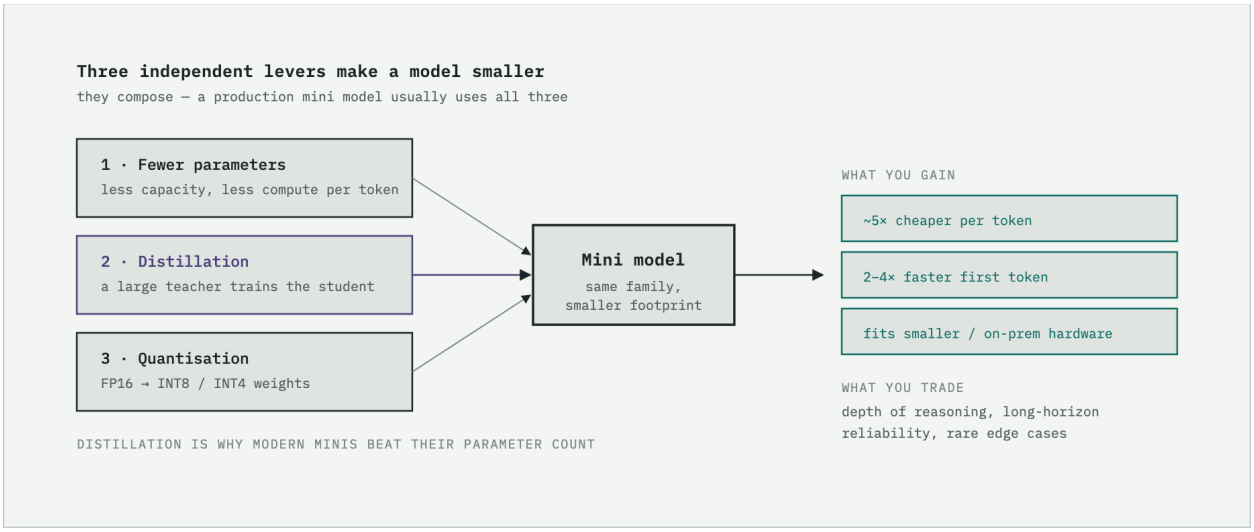

Fig. 1 *Three independent levers make a model smaller. They compose — a production mini model usually uses all three.*

Fewer parameters. Less capacity to store patterns, less arithmetic per token. This is most of the speed and cost win.

Distillation. The interesting one. You run the large model across enormous volumes of prompts, capture its outputs, and train the small model to reproduce them. The student is not learning from raw text it is learning from a strong model's already-reasoned answers. That is a far cleaner training signal, and it is why a distilled mini today outperforms from-scratch models several times its size from a few years ago.

The limit is subtle and worth understanding: the student learns the teacher's *outputs*, not its *capacity*. It reproduces the shape of good reasoning without the headroom to derive it fresh on a genuinely novel problem. That is precisely why minis look strong in demos and turn brittle at the edges.

Quantisation. Weights are normally 16-bit floats. Round them to 8-bit or 4-bit integers and the model shrinks two to four times in memory for a small accuracy cost. This is mainly a deployment lever it is how a model fits on hardware you control, which matters when the data cannot leave the building.

The gap is not a fixed number

The capability difference between a mini and a flagship is not a constant you can look up. It depends almost entirely on how many steps the task chains together.

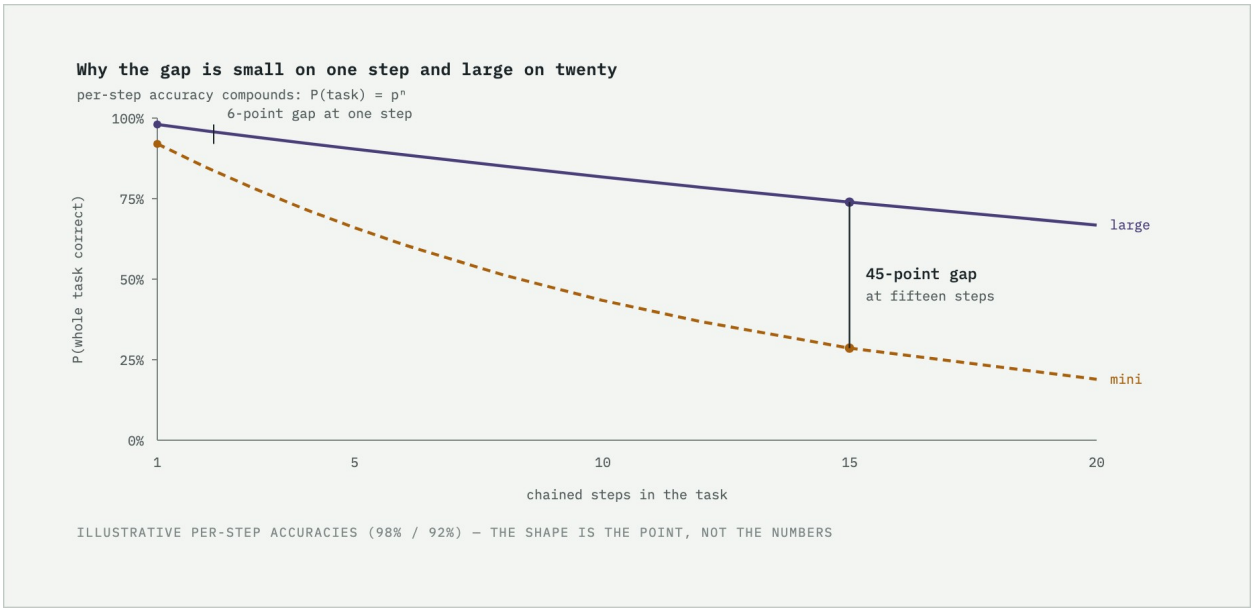

Fig. 2 *Identical per-step difference, completely different outcomes. Accuracy multiplies rather than averages.*

Suppose a mini is right 92% of the time per step, and the large model 98%. On a single-step task you are arguing over six points often not worth five times the price.

Chain fifteen steps and those same two models land at 29% and 74%. A 45-point gap, from the identical per-step difference, because accuracy multiplies rather than averages. 0.92 to the fifteenth is 0.29.

> **Nothing about the models changed. Only the shape of the work.**

The routing rule

Split on step count, not on how hard the task feels.

Teams almost always route on perceived difficulty, and perceived difficulty is unreliable it reflects how hard the task looks to a human, which has little to do with where a language model struggles. Step count is something you can actually inspect.

Single-step, verifiable, high-volume work is where minis win outright. Classify this record. Extract these six fields. Is this ticket about billing or provisioning? The gap is small, the volume makes the cost difference real, and an error surfaces immediately instead of propagating.

Anything agentic belongs on a capable model. Not because minis cannot call tools they can but because an agent loop *is* a chained-step task. It is the one shape where a small per-step deficit compounds into a mostly-failing system.

There is a third pattern worth knowing: a capable model orchestrating cheap workers. The expensive model makes the decisions; minis do the bulk reading and extraction underneath it. You get flagship judgement at something much closer to mini economics.

Before you switch models at all

Model size is a cost lever, but it is not the first one you should reach for. The ordering that actually works:

1. **Prompt caching.** Free. Usually the largest single win, and most teams have it silently broken by a timestamp or an unsorted JSON blob sitting in the cached prefix.
2. **Token hygiene.** Free. Stop re-sending context the model does not need.
3. **Reasoning effort on the model you already have.** A capable model at reduced effort frequently matches a mini at full effort.
4. **Then, and only then, a smaller model.**

Step three is the one people skip, and skipping it is expensive in a non-obvious way. Prompt caches are model-scoped. The moment you split traffic across two models you forfeit cache reuse between them, and you now maintain two behaviour profiles and two eval suites. Plenty of cascades have cost more to operate than the single-model setup they replaced.

Three things that catch teams out

Judge cost per completed task, not per request. A mini that needs three attempts and a human correction is not cheaper than one call to a larger model. This is the single most common error in tiering decisions, and it is invisible if you only watch per-request spend.

"Mini" is a moving target. Today's small models routinely beat flagships from eighteen months ago. Any routing decision you make now has a shelf life. Re-measure; do not inherit.

Without evals, you cannot tier at all. If you drop a route to a smaller model and have no scored test set, you will not learn it degraded a customer will tell you. Non-deterministic components need measurement, not assertions. The eval suite is not overhead on this work; it is the instrument that makes the decision possible.

The short version

Stop asking which model is best. Ask how many steps the task takes, whether the output is cheaply verifiable, and what an error actually costs you.

Then tier accordingly the same way you would tier storage, and for the same reason. Not because the premium tier is bad, but because an architecture that routes everything to it has not been designed. It has just been paid for.

Where have you seen this go wrong the team that put everything on the flagship and got a bill they could not defend, or the one that pushed a multi-step workflow onto a mini and spent months debugging what looked like a prompt problem?