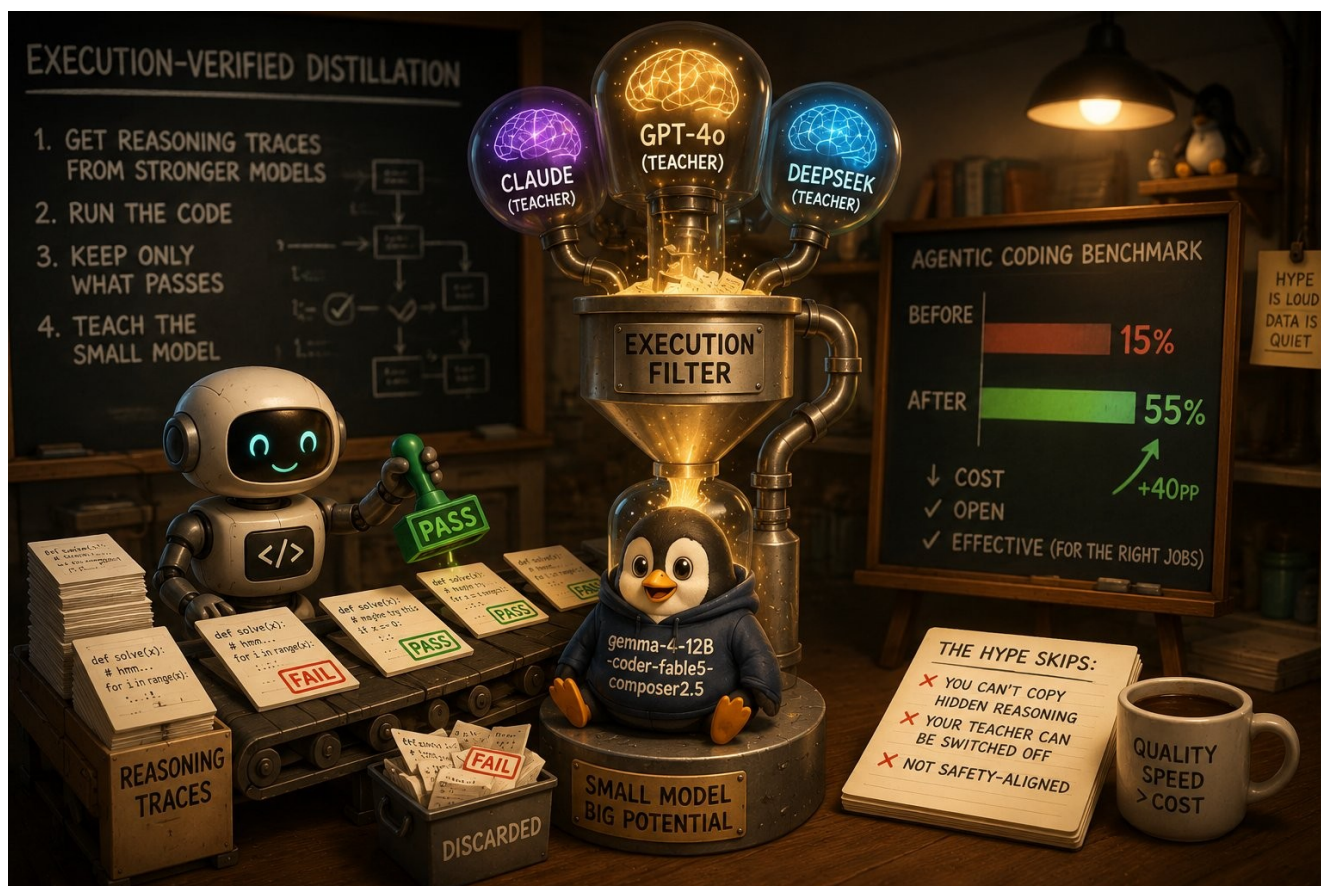


Distilling frontier reasoning into a local model: what actually works

Julien Laurenceau | Published 2026-06-22 | Updated 2026-09-30

<https://pepitedata.com/frontier-model-distillation-local-llm/>

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien_Lau



TL;DR. A small coding model named gemma-4-12B-coder-fable5-composer2.5 got passed around r/LocalLLM last week and was dismissed as a "mashup of random buzzwords." It is not. Strip the emoji from its model card and you find a textbook case of execution-verified distillation: take a cheap open model, teach it with reasoning traces from stronger models, and keep only the traces whose code actually passed its tests. The method is sound, it has been downloaded around 359,000 times in a month, and the creator reports a jump from 15% to 55% on an agentic coding benchmark. The economics are real, but only for a narrow class of work. Cost is not even the main axis: for the work you actually compete on, quality and speed win, and the frontier premium is usually worth paying. This post separates the engineering from the hype, and flags three things the hype skips: you cannot copy a frontier model's hidden reasoning, the teacher you depend on can be switched off, and a model trained this way is not safety-aligned.

The artifact everyone laughed at

The model is a fine-tune of Google's gemma-4-12B-it, published on Hugging Face by an independent author and shipped as GGUF quants. The smallest quant runs in about 4.5 GB of VRAM or unified

memory. The recommended one needs about 6.9 GB. That puts a capable coding assistant on a mid-range laptop or an Apple Silicon Mac, fully offline, with no API key and no cloud bill. It carries an Apache-2.0 license, because Gemma 4 itself moved to Apache-2.0, so the weights are free to use, modify, and redistribute.

The reaction on r/LocalLLM was mostly scorn. "A skip in my testing." "Fake gimmick." "Mashup of random buzzwords." "Fable5 plus Composer 2.5, why not just pick one?" The name does read like someone shook a jar of model labels onto a sticker. One commenter pushed back and quoted the training section of the card, then noted that distilling a small model from larger ones using chain-of-thought traces "is a very well attested approach."

He was right. The skeptics were reacting to the packaging. The method underneath is the interesting part, and it is worth understanding even if you never download this particular file.

What the method actually is

The model is trained on one specific kind of data: verifiable Python coding tasks. These are algorithmic, function-level problems that ship with deterministic tests. That detail is the whole game. It means every training example can be checked by running it, not by trusting that it looks correct.

The training set is a distillation from two teachers.

The primary teacher, Composer 2.5, solved each problem and produced its reasoning out loud. The author then ran the generated code against the task's tests and kept only the solutions that passed. The reasoning a passing solution produced becomes a training example. Reasoning that led to broken code is thrown away. This is rejection sampling, and it is the difference between learning from "code that looks plausible" and learning from "code that demonstrably works."

The second teacher patches the gaps. The author collected the problems where Composer 2.5 failed and handed them to Fable 5, Anthropic's most capable widely released model. Fable 5 re-derived a fresh solution and its reasoning, and those were gated on passing the same tests. This recovers the hard cases the primary teacher missed. The author tags these traces separately so the two sources stay distinguishable.

So the recipe is: a strong primary teacher for broad coverage, a stronger second teacher to clean up the failures, and an execution gate on everything before it enters training. None of that is a gimmick. It is the standard shape of a serious distillation pipeline, and the execution gate is exactly what most open coding datasets lack. Many of them are scraped code dumps where nobody knows whether the code runs. Here, every sample had to prove itself.

The two-teacher detail is what drew the mockery, and it is actually the most defensible choice in the whole project. You use your cheaper, faster teacher for the 80% it gets right, and you spend your expensive teacher only on the 20% it gets wrong. That is good cost discipline, not indecision.

Does it work?

The honest answer is: the creator reports that it does, on one benchmark, in one local harness. Treat the numbers as a signal, not as an independently verified result.

The number that matters is from the newer v2 release, on tau2-bench telecom. That benchmark asks a model to act as an agent: diagnose a problem, fix it, then verify the fix, which mirrors real terminal and debugging work. Running locally at 8-bit precision, the creator reports:

Model (tau2-bench telecom, local, Q8_0)	Score
gemma-4-12B-it (base)	~15%

Model (tau2-bench telecom, local, Q8_0)	Score
Distilled v2	~55%

That is roughly a 3.5x improvement on the same hardware, from a fine-tune rather than a bigger model. The qualitative description is more telling than the percentage. The base model "gives up early" and hands the problem back to the user. The distilled model "keeps going" and works the problem the way a much larger model would. That behavior, persistence on a multi-step task, is precisely what you are trying to transfer when you distill reasoning traces rather than answers.

Be careful with what this proves. It proves the method moved a narrow, verifiable capability a long way on a 12B model. It does not prove the model is good at anything outside that domain. The card says as much: it is specialized for Python and algorithmic coding, general-knowledge facts should still be checked, and it is English-centric.

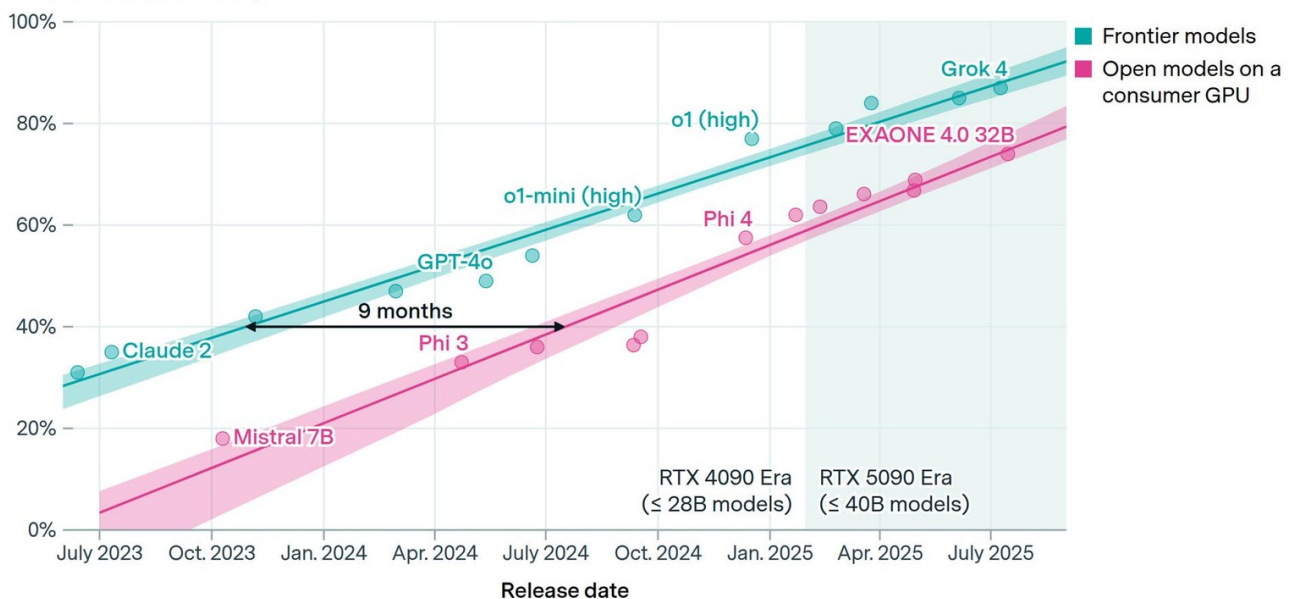
The gap is not a fluke, it is a measured trend

This one model is a single data point in a pattern that Epoch AI has quantified. Their analysis tracks GPQA Diamond accuracy over time for two groups: frontier models, and open models small enough to run on one consumer GPU. The second line tracks the first about nine months behind, and the lag has held roughly steady.

Models that fit on a consumer GPU trail the absolute frontier by only 9 months

EPOCH AI

GPQA Diamond accuracy



CC-BY

epoch.ai

Open models that fit on a consumer GPU trail the absolute frontier by about nine months on GPQA Diamond. Source: Epoch AI, CC-BY.

The framing is blunt. Frontier AI performance becomes accessible on consumer hardware within about nine months. Epoch puts the broader range at 5 to 22 months for open-weight models of any size. They name three drivers of the trend, and one of them is model distillation, the exact technique behind the Gemma fine-tune above. That model is not an outlier. It is the mechanism of the trend made concrete: a

single top-end gaming GPU under \$2,500 now runs models that match the frontier of nine months ago.

Read Epoch's own caveat, because it is the same one this whole post turns on. Small open models are more likely to be tuned for specific benchmarks, so the real-world lag is probably longer than nine months. A benchmark score is not general capability. The chart shows small open models reaching frontier parity on one test. It does not show them matching the frontier across everything you might ask. That is exactly why distillation pays off on narrow, measurable tasks and disappoints as a general substitute.

For a decision-maker the implication is sharp. If frontier-level performance on your specific, measurable task lands on hardware you already own within roughly a year, then paying a metered frontier bill for that task buys you a time advantage, not a permanent moat. Sometimes that time advantage is worth every dollar. Sometimes it is a recurring cost for something the open ecosystem is about to hand you for nearly free. Telling those two cases apart, before the bill compounds, is the job.

The economics, which is the real reason to care

Now the money. The frontier API is metered. A distilled local model is not.

Fable 5 costs about \$10 per million input tokens and \$50 per million output tokens. Those are fair prices for a model at that capability. They also scale linearly with usage, forever. A local 12B model costs you a fixed amount of hardware and then runs at close to zero marginal cost for every call after that.

The trade is one-time cost against per-call cost. Consider an illustrative example. The assumptions are explicit so you can substitute your own.

	Frontier API at volume	Distill once, run locally
One-time build	None	~50,000 verified traces, plus fine-tune compute
Per-call cost	~\$0.02 to \$0.06 per completion	~\$0 marginal
Annual cost at 5M completions	roughly \$100k to \$300k	one-time build, then hardware only
Capability	Broad, frontier-level	Narrow, near-frontier on the trained task only

The one-time build is not free. Generating tens of thousands of execution-verified traces with a frontier teacher, including the wasted attempts that fail the test gate, is real money. Call it low tens of thousands of dollars, plus a few hundred dollars of fine-tuning compute. But it is paid once. The frontier bill is paid every month and grows with adoption.

The asymmetry is the whole argument. If your workload is narrow, high-volume, and verifiable, distillation converts a recurring operating cost into a one-time capital cost, and moves inference onto hardware you already own. That is the same logic behind "avoid unnecessary cloud spend," applied to AI rather than to storage or compute.

The trap is assuming the distilled model is a general substitute for the teacher. It is not. It is a specialist. You are not cloning Fable 5. You are extracting one skill from it and pressing that skill into a model small enough to run on a laptop.

Cost is only one axis, and often not the deciding one

The cost case is real, but it is not the whole decision, and treating it as the whole decision is its own mistake. For a large class of work, quality and speed matter more than price per token, and the rational

move is to pay the premium for the best model.

Look at where the frontier earns its price. In competitive markets the best option takes most of the reward. The strongest model writes the code that ships, wins the deal, or closes the case, and second best does not get a proportional share. This is the winner-take-all pattern you see in markets and in sport. A quality gap that looks small on a benchmark decides the outcome in practice. When your task sits on that competitive edge, a model that is nine months behind is not a bargain. It is a handicap.

Speed compounds the same way. Capability is correlated with time, and the team that reaches a new capability first gets to build on it first. That advantage accrues while everyone else catches up. The Epoch data cuts both ways here. The nine-month lag that makes a local model cheap is the same nine months your competitor spends shipping on the frontier. If the work is on your innovation frontier, paying to stay current is not waste. It is the point.

So the honest rule is not "distill to save money." It is to match the model to the role the task plays.

- **Commodity work**, narrow and verifiable and not a differentiator: buy the cheapest thing that clears the bar. Distillation is often that thing.
- **Differentiating work**, where quality or speed decides who wins: pay for the frontier and do not flinch at the per-token price. The premium is buying the win, not the tokens.

Most organizations run both kinds of work at once. The error is using one policy for both. Paying frontier prices to run a commodity job at scale is waste. Distilling a last-generation model for the capability your product competes on is a quiet way to fall behind.

What the hype skips

Three things never make it into the launch posts. All three matter more to a production decision than the benchmark does.

You cannot copy a frontier model's hidden reasoning. The marketing language talks about learning "the reasoning" of a frontier model. There is a precise limit here. Fable 5's raw chain of thought is never returned by the API. You receive a summarized version of its thinking, or you receive the reasoning it chooses to write into its visible answer. The private trace is not exposed on any setting. Anthropic even has a refusal category for requests that try to extract a model's internal reasoning. So when a card says it distilled a frontier model's chain-of-thought, what it really captured is the model's visible, output-level reasoning. That is still useful. It is not the same as cloning the model's private deliberation, and anyone budgeting a distillation project should know the difference before they start.

The teacher you depend on can be switched off. This is not hypothetical. The third-party writeup on this same model reports that access to Fable 5 was removed, and that because the Fable 5 dataset was small, the next version cannot lean on it without overfitting. The roadmap is shifting back to the primary teacher and possibly to a different frontier model entirely. If your pipeline depends on a specific teacher's outputs, you have a supply risk. The tap can close, prices can change, and provider terms can restrict using outputs to train competing models. Treat your teacher access as a dependency to be managed, not a constant. Confirm the licensing position before you build on it.

A model distilled this way is not safety-aligned. The card is admirably blunt: the training data is task-focused with no safety hedging, so the model refuses less than the base model, and it is explicitly not safety-aligned. For a personal coding assistant that is a feature. In a product, it is a liability you now own. Stripping the safety behavior out of the traces is part of what makes the model feel sharp and direct. It also means you have removed guardrails that you will have to rebuild yourself before anything customer-facing touches it. That rebuild is a real line item, and it is invisible in the download count.

When this is the right call, and when it isn't

Distillation into a small local model is the right move when three conditions hold together.

- **The task is narrow.** You are improving one capability, not building a general assistant.
- **The task is verifiable.** You can check each training example automatically, the way passing tests check generated code. Without that gate, you are just copying a bigger model's confident mistakes into a smaller one.
- **The task is high-volume.** The per-call savings only beat the one-time build cost if you make a lot of calls.

Coding against tests fits all three. So do structured extraction with a schema you can validate, and classification where you hold ground truth. Those are the cases where a distilled specialist on owned hardware quietly outperforms a metered frontier API on cost, latency, and data control.

It is the wrong move when the task is broad, when you cannot verify outputs cheaply, or when volume is low enough that the frontier bill is already small. In those cases, distillation is effort spent to build a worse model. The failure mode the r/LocalLLM crowd was reaching for, even if they could not articulate it, is real: stacking teacher names and labels without an execution gate produces exactly the buzzword mashup they accused this model of being. The label is not the method. The execution gate is the method.

The takeaway

The model that got laughed at is a clean example of a technique that more teams should be using, applied to the one domain where it works best. The lesson is not "download this file," and it is not "distill everything to cut the bill." The lesson is to match the model to the job. For commodity work that is narrow and verifiable, the gap between a frontier API and a cheap local model is smaller than your bill assumes, and closing it is a clear engineering decision. For the work your product competes on, quality and speed win, and the frontier premium is usually worth paying.

The expensive mistake is running one policy for both. If you are about to commit to a frontier spend that scales with usage, or about to distill a model for something that decides whether you win, it is worth a second opinion on which kind of task you are really looking at. That is exactly what an independent review answers, before the bill or the lag compounds.

If your team is weighing frontier API spend against a local or fine-tuned model, an outside read before you commit is usually cheaper than the bill. [Book a 15-minute intro call](#), or take a focused hour on the decision with an [Expert Call](#).

Update 2026-09-29. This post is now part of [AI Infrastructure and SRE: Running Models in Production](#), the field guide to running models in production.

Related posts

- [AI Infrastructure and SRE: Running Models in Production](#)
- [How big models teach small models](#)
- [Model fusion beats the frontier model. The cost case.](#)
- [The frontier model has no teacher](#)

PDF snapshot of the public article. The online version is authoritative; links remain clickable.