LoRA Skill Compilation Is a Double-Headed Coin Flip

September 3, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** I trained the same LoRA adapter ten times, changing only the seed, and the coin flipped: a 12.77-point spread between best and worst adapter, and all ten landed on the same side, at or below the model with no adapter. Here is what moved, what did not, and what the loss curve never told me.

📚 **More on this topic:** Agent Skill Compilation Tested · Skills in the Weights · LoRA Training on Consumer Hardware · InsiderLLM Benchmarks

Every LoRA guide on this site ends the same way, mine included: train the adapter, merge it, ship one file. The last article turned that advice into a claim about agent skills. Compile the skill into the weights and the prompt tax goes away — no 1,383.9 extra tokens on every call. Everyone recommends it. What I could not find anywhere was a measurement of whether the adapter you get is the adapter you would get by running the training again. So I trained the same adapter ten times on identical data with an identical configuration, changing only the seed, and scored all ten against the base model on a frozen 47-item test split. This is that measurement.

What this changes

One adapter, scored once, measures the seed

The ten adapters scored between 15 and 21 of 47 on the same data and the same configuration. One test item is 2.13 points, so the spread from worst seed to best is six items — the distance between “this adapter broke the model” and “this adapter did nothing.” Any two seeds agreed on somewhere between 31 and 44 of the 47 predictions, so even adapters with identical scores were not making the same calls.

None of that is scoring noise: repeat scoring of the same adapter was byte-identical every time. What varies is the training, and a single adapter with a single benchmark number is one draw from that distribution. If you have shipped an adapter on the strength of one score, the number you shipped on was mostly the seed.

At a few dozen examples, expect the adapter to hurt

The base model with no adapter scored 21 of 47, or 44.68%. The ten-seed mean was 41.06%, a mean effect of minus 3.62 points. Two seeds tied the base, eight scored below it, none scored above it. I would have liked one seed over the line — even by a single item — and there is not one.

That is what makes it a double-headed coin. The flip is real — 12.77 points of it between seed 1 and the best two — and it lands on the same side every time. The recipe was the plain one from the consumer-hardware LoRA guide, pointed at a 27B in 4-bit with 24 examples. Every adapter learned its 24 examples. None of them learned the task, and the average one unlearned a little of what the base already knew.

The variance matches prompt compilation, so the substrate is a cost decision

On 30 August I compiled a prompt skill ten times from identical input and the ten skills spread 4.65 percentage points on this same test split. The ten adapters spread 3.62 — same formula, sample standard deviation over ten scores, same 47 items, same task. A fifth narrower, and not narrow enough to buy anything.

If both substrates hand you a draw from a distribution of about the same width, neither is the "reliable" one, and the choice between them comes down to what each costs to produce and to run. The prompt side costs 24 cents per compile and 1,383.9 tokens on every call forever. The weights side costs sixteen and a half minutes of an RTX 3090 per seed, with no tax afterwards. That is the 188-call break-even from the prompt article, with one correction: whichever side you pick, budget for several runs, because one tells you almost nothing.

Thirteen items decide the score

Forty-seven test items, ten adapters — and the whole result lives in thirteen of them. Thirteen items pass under every seed and under the base, and twenty-one fail under every seed and under the base; those 34 do not care whether there is an adapter at all. The other thirteen do. They sort into three reliable gains, four reliable breaks, two of which happen under every seed, and six that go whichever way the seed sends them.

Item	Gold label	Base	Seeds passing	Group
15	answer / chat / all	fail	8 of 10	reliable gain
37	answer / chat / all	fail	9 of 10	reliable gain
61	rag / explore / session	fail	8 of 10	reliable gain

Item	Gold label	Base	Seeds passing	Group
7	answer / chat / all	pass	0 of 10	reliable break, all seeds
34	answer / chat / all	pass	0 of 10	reliable break, all seeds
33	answer / chat / all	pass	1 of 10	reliable break
58	rag / recall / session	pass	1 of 10	reliable break
6	answer / chat / all	pass	7 of 10	seed-dependent
18	rag / recall / session	fail	6 of 10	seed-dependent
35	answer / chat / all	pass	9 of 10	seed-dependent
40	answer / chat / all	pass	4 of 10	seed-dependent
62	rag / recall / session	pass	5 of 10	seed-dependent
73	answer / chat / session	fail	5 of 10	seed-dependent

The three gains are two chat messages the base had been sending to retrieval and one retrieval request it had been sending out to the web as well. The four breaks are three chat messages the adapters now send to retrieval, two of them under all ten seeds, and a retrieval request they now answer from nothing. Three up, four down, so the reliable part of the adapter's work costs one item before you reach the six coin flips — and the six are where your seed decides the score. Seed 1, the outlier, failed ten of the thirteen, which is the entire reason it sits at 15.

Five of the six seed-dependent items flip between answer/chat and rag/recall. That chat-versus-retrieval line is the same boundary the prompt compiler kept tripping over.

The protocol

Here is what I will do from now on, and what you should do before trusting any adapter number.

Train at least three seeds. Score every one of them against the no-adapter base through the same path the adapters trained in. Report the spread, not the best seed. Three seeds will not give you a standard deviation worth quoting, but they will tell you whether your best result is a result or a draw.

Do not pick the seed by its training loss. The lowest-loss seed scored 19 of 47; the two best seeds had the highest losses of the clean batch. Loss on 24 examples measures memorisation of 24 examples, and the test measures something else.

The per-seed bill on an RTX 3090 is 9 seconds to load the base, 981 to 984 seconds to train, and 169 to 171 seconds to score — sixteen and a half minutes of training and three of scoring, about three and a quarter hours of card time for ten. Each adapter is 319 MB on disk, 79.7 million trainable parameters. Power was not logged this run — the sampler recorded memory only — so there is no watt-hour figure to put beside that.

VRAM is the real constraint: the training process peaked at 23,904 MiB of 24,576, leaving 672 MiB of headroom. My first attempt at seeds 3 to 10 ran with the desktop up and a soak test in the background, and seven of the eight seeds went out of memory in their first forward pass. The second attempt stopped the display manager, paused the soak, and gated every seed on at least 21,864 MiB free; all eight trained clean. On a 3090 this recipe is a headless evening. On a 12 GB card it is not this recipe at all — the 27B base in NF4 alone was 16,766 MiB resident immediately after load, by the 1-second nvidia-smi sampler and identical across all eight traced seeds, before the optimiser got a byte.

Limits

The n here is 24 training examples and 47 test items, on one task, one Qwen3.6-27B base, and one hyperparameter set, and nothing above is a statement about any bigger box than that. It does not show that LoRA cannot learn intent classification, because the 15-item validation split was never scored — deliberately — and a validation-gated stopping rule or more data could move the mean, the spread, or both. Scoring went through PEFT with the base in NF4, the configuration the adapters trained in, because llama.cpp's runtime adapter application turned out to move predictions on its own even with every adapter's scale pinned to zero, and that is a separate finding for a separate article. I have not relabelled any of the thirteen items and I am not going to until there is a reason that is not "the adapters would score better."

Method and full results

Base model Qwen3.6-27B in HF safetensors, loaded NF4 through bitsandbytes with double quantisation and bf16 compute. LoRA rank 16, alpha 32, dropout 0.05, on q, k, v, o, gate, up and down projections. AdamW at 1e-4, batch 1 with gradient accumulation 4, ten epochs, 60 optimiser steps, loss on answer tokens only. Training data is the 24-item compile split; the test split is 47 items, frozen on 30 August 2026 and shared with the prompt experiment. Seeds 1 to 10 seed `random`, `numpy`, `torch` and the shuffle together. Seeds 1 and 2 ran in a pilot on 31 August, seeds 3 to 10 in one batch on 2 September, all on the same trainer file. The primary metric is exact match: predicted tool, mode and scope all equal to gold after the deployed sanitise pipeline.[1] Scoring used greedy decoding through PEFT with the base in NF4, and the

base was scored the same way with no adapter attached. Full records are on the benchmarks page.

Seed	Exact	vs base	Final epoch loss
1	15/47 = 31.91%	−12.77 pp	0.0355
2	20/47 = 42.55%	−2.13 pp	0.0016
3	19/47 = 40.43%	−4.26 pp	0.0000
4	19/47 = 40.43%	−4.26 pp	0.0005
5	19/47 = 40.43%	−4.26 pp	0.0019
6	21/47 = 44.68%	0.00 pp	0.0062
7	19/47 = 40.43%	−4.26 pp	0.0021
8	20/47 = 42.55%	−2.13 pp	0.0011
9	20/47 = 42.55%	−2.13 pp	0.0019
10	21/47 = 44.68%	0.00 pp	0.0127
Mean	**41.06%**	**−3.62 pp**	

Spread is the sample standard deviation over the ten percentages: 3.62 pp. Seed 1 came from the pilot rather than the batch — it ran on the final configuration and was scored the same way, so it stays in. Over seeds 3 to 10 alone the spread is 1.89 pp. The mean effect and the spread are the same figure by coincidence, and I checked the arithmetic twice.

Loss did not predict score. Seed 3's final loss was 3.5e-6 — the lowest of any run — and it scored 19. Seeds 6 and 10 had the highest final losses of the clean batch and scored 21. Across the ten, the rank correlation between final loss and exact match is close to zero; within the eight batch seeds it is positive, which with eight points I read as the absence of the expected direction rather than the presence of the opposite one. Seed 1's loss fell to 0.0003 by epoch eight, rebounded to 0.0856 in epoch nine and finished at 0.0355. Four other seeds had smaller rebounds in epochs six and seven and scored anywhere from 19 to 21.

The base scored 21 of 47 here and 23 of 47 in the prompt experiment. The difference is the path: the prompt skills ran through llama.cpp on a Q4_K_M GGUF, the adapters through PEFT on an NF4 load. Compare each experiment against its own base, and do not read the two-item gap as a quantization finding.

Scoring noise on the PEFT path was zero. Seed 1 was scored five times and seed 2 twice, and every repeat was byte-identical in its predictions.

1. The results record also carries a second, looser metric, “effective match”, which ignores scope on non-retrieval items. It was introduced on 30 August 2026 during the prompt experiment, after that experiment’s baseline had been scored and before its compiled arms ran, and it was not pre-registered. Its numbers are in the results record and do not change the null. They appear nowhere else in this article. ↩