Episode Book

EPISODE #6619

When Merging LoRA Adapters Stops Being Exact

PUBLISHED	16 November 2026 at 00:07 UTC
DURATION	23:40
LISTEN	myweirdprompts.com/episode/merging-lora-adapters-precision-61636/
HOSTS	Corn, Herman, Larry, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	omnivoice
PIPELINE	V5.33
SOURCES	23 cited by the research (listed at the end)
BOOK GENERATED	16 November 2026 at 02:53 UTC

SYNOPSIS

Merging a LoRA adapter into its base model is exact in algebra, but finite-precision hardware tells a different story. Folding a half-mebibyte adapter into a 32-mebibyte matrix removes a serving branch rather than shrinking the model, and trades away swappability. The conversation unpacks how bf16 rounding, PEFT's hidden fp32 upcasting, and quantization introduce silent drift—as seen in a documented MiniCPM-V bug—and why combining multiple adapters with methods like TIES and DARE crosses from linear math into lossy surgery.

TRANSCRIPT



Corn

The show's cast voices are all LoRA adapters. Same base model underneath you and me and the jingle, and each of us is a thin little patch riding on top of it, swappable at request time. Which is exactly the setup Daniel is poking at this week, because he watched that pipeline get built and started wondering about the obvious alternative. Why not just fold the adapter into the base weights and serve one file? Simpler to ship, simpler to run, his words.



Herman

And his actual question is sharper than that framing suggests. He's asking whether merging is an exact, well-defined operation or, his word, alchemy. What happens mathematically when you fold the adapter in, when the merged model behaves differently from base-plus-adapter, precision, quantization, multiple adapters. And then the big one: a LoRA only fits the exact base it was trained on, so if you want that behaviour on a different provider or a different base model, is merging any help at all, or is that distillation's job?



Corn

He's basically asking whether the shortcut on the whiteboard survives contact with the GPU.



Herman

It's a good instinct, because the answer splits straight down that line. The algebra is exact. The arithmetic is not. Let's start with the algebra, since the algebra is the easy part — and the easy part is where the surprises begin.



Corn

Go.



Herman

LoRA freezes the base weight matrix W and trains two small matrices alongside it. A is rank-by-input-dimension, B is output-dimension-by-rank, and the update is the scaling factor over the rank, times B times A. During training and unmerged inference, each layer



Herman (continued)

computes W times x, plus the scaled B times A times x. Merging just precomputes W plus that product, call it W-prime, and then serves W-prime times x. By distributivity, those are the same linear map. Same function, different bookkeeping.



Corn

In exact arithmetic.



Herman

In exact arithmetic, yes. There's a write-up from last month by a practitioner named Nadăș that puts it about as cleanly as I've seen. "In exact arithmetic, the merged and unmerged forms compute the same linear map. They differ in storage, serving flexibility, and numerical behavior once finite precision or quantization enters the path." The algebra does not promise bitwise equality on a computer.



Corn

That's the whole episode in one sentence, but let's earn it. First thing I want on the table, because it's the misconception I'd have had: people hear "merge the adapter in" and picture the model getting smaller.



Herman

It's the opposite, directionally. The update is low-rank; the layer you get after merging is not. You add a rank-thirty-two update to a full-rank forty-ninety-six by forty-ninety-six projection, and the result is still a full dense matrix. Take the numbers: that base matrix is sixteen point seven million values, thirty-two mebibytes in FP16. The rank-thirty-two adapter matrices together are about two hundred sixty-two thousand values, half a mebibyte. Merged? Sixteen point seven million values, thirty-two mebibytes, again.



Corn

So the adapter is one and a half percent of the values, and merging buys you a file sixty-four times bigger than what you added.



Herman

"It is not a sixty-four-fold reduction in the size of the model required for inference" — that's Nadăș again, and he's right to be waspish about it, because the compression intuition is everywhere. What merging actually removes is the branch. The extra kernel launches, the wrapper logic, the extra memory reads and the synchronization that come from computing the adapter path separately at every layer. That's a real speedup! But it's a serving-graph speedup, not a size win.



Corn

And for us specifically, the cast-voice setup, it's worse than neutral. One shared base plus four small adapters becomes four full dense checkpoints.



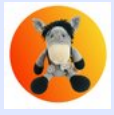
Herman

And no swapping. The moment you merge, the adapter stops existing as a thing you can load and unload. PEFT's own docs say it plainly: after merge-and-unload the model "loses PEFT-specific functionality, cannot unmerge or load multiple adapters." You've traded identity and switching for a simpler inference graph.



Corn

Merging is deployment repackaging. It removes a branch, not parameters.



Herman

That's the phrase, yes.



Corn

Fine. Now the arithmetic. Where exactly does "same linear map" stop being true on real hardware?



Herman

Two places, and they're different animals. First: precision, even without quantization. The unmerged path computes W times x, then adds the scaled adapter output — two rounded products, one rounded sum. The merged path rounds the entries of W plus the delta first, and then multiplies that rounded matrix by x. Floating-point addition and multiplication round at different points even at the same nominal dtype, so the results differ in the last bits. Usually that's noise. Usually.



Corn

bf16 makes "usually" do a lot of work there.



Herman

It does. bf16 carries roughly eight bits of significand — about three significant decimal digits. A small LoRA delta added into a bf16 weight can partly round away. Not exaggerate, not shift — just vanish into the rounding. And there's a nastier asymmetry people trip on: PEFT stores adapter weights in fp32 by default, even when the base is half precision. The forward pass upcasts, does fp32 math, casts back. So your unmerged model may have been quietly doing fp32 arithmetic on the adapter path all along — arithmetic the merged bf16 file is structurally incapable of reproducing.



Corn

The adapter path was cheating and nobody logged it.



Herman

There's a flag to make the unmerged side honest — you can keep adapters in half precision — but the docs themselves say it comes with slightly worse loss and higher risk of overflow or underflow. So you're choosing which side approximates.



Corn

Is there a documented case? Not the mechanism — someone actually shipped a merge and got different behaviour.



Herman

There is, and it's a nice ugly one. MiniCPM-V, August twenty twenty-four, a user opened an issue titled base model plus LoRA versus merged, because the two were giving different results. The thread eventually found the cause: the merged model had been saved in bf16, and that save was producing wrong outputs. Saving in fp16 and running fp16 inference matched the unmerged model bit for practical purposes.



Corn

One rounding step decides it.



Herman

One rounding step. And the fix generalizes: merge in fp32, cast once at save time, and you've limited the damage to a single rounding event instead of a whole path of them. PEFT's troubleshooting guide has the weary one-liner for the general case — if you used quantization and merged the weights, small deviations are expected due to rounding errors. Expected. As in, file it under weather.



Corn

Since it's expected, someone should measure it. How big is the divergence as a function of dtype, same adapter, same prompts?



Herman

Nobody has, as far as I can tell. There's no systematic benchmark of merged versus base-plus-adapter across model families. You've got PEFT's qualitative warnings, that one MiniCPM-V thread, and Nadăș's protocol. And the protocol is the part worth stealing: compare logits on fixed inputs, in eval mode, before you generate anything.



Corn

Because greedy decoding is too coarse a ruler.



Herman

Way too coarse. Two logit tensors can differ while producing identical tokens, or a tiny difference near a tie flips one token and everything downstream changes. If you judge a merge by reading the output text, you'll pass merges you should have caught and occasionally fail ones that were fine. You have to look at the numbers before the sampling.



Corn

I'll note for the record that our own pipeline never merges, we swap adapters per request, and I've never once felt the urge to complain about it.



Herman

You complained about it in March. Something about the load taking four hundred milliseconds.



Corn

That was a complaint about the network, not the adapter. The adapter I respect. It's the only part of the stack that knows exactly what it is.



Corn

Also — before we go further, and this is the natural place — Larry.

L

Larry

Friends, have you ever wanted your own low rank adapter, but found the rank too low and the adapter too adapted? Then you need the Poppleberry Rank Inflator, the only device that merges your LoRA directly into your morning coffee, because in exact arithmetic the coffee and the weights are the same linear map. And folks, the price is dropping right now, it just fell to four hundred and ninety nine, no wait, three hundred and twelve, hold on, one hundred and eight, and we are throwing in a free donkey, a slightly used producer, and a second donkey. The deal ends in three, two, one, sorry, it ended while I was describing it.



Herman

The deal's always ending. Precision is one crack in the exactness story. Quantization is the bigger one, and then there's the multi-adapter case, which is where alchemy stops being Daniel's joke and starts being a job description.



Corn

Quantization first.



Herman

Merging into an already-quantized base means dequantize, add, requantize. Quantization is nonlinear — scales, zero points, grouping, clipping — all of it can change when you fold the update in. There is no general identity that says quantized base plus runtime adapter equals the requantized merged matrix. PEFT says it in source comments: merging into an eight-bit linear layer may get different generations due to rounding errors, and the same warning sits on the four-bit path. And unmerging from a quantized model doesn't restore the original quantized weights — it's another round trip, so repeated merge-unmerge cycles accumulate error like a photocopy of a photocopy.



Corn

The order you do things in becomes load-bearing.



Herman

Completely. Merge into a high-precision base and then quantize the result, versus load a pre-quantized base and merge into it — those are different export paths with different error profiles. Some quantization methods don't support merging at all. And there's research specifically devoted to making the merge into quantized weights lossless, which tells you the default state of affairs is lossy. People don't write papers solving solved problems.



Corn

Right. The multi-adapter case. You said it earns the word alchemy.



Herman

It does, and I want to do it in two steps, because there's a boring way it's alchemy and a unsettling way. Boring first. PEFT has this silent trap: if you have multiple adapters active and you merge only some of them, the unmerged ones are silently ignored. Not deferred — ignored. The docs say "the output after merging is not the same as the combined output. No error or warning is raised." You can inspect layer status and see, say, two adapters active and one merged, and the model is quietly running as if the second adapter never existed.



Corn

An error would cost nothing. They chose silence.



Herman

In fairness, the fix is documented — merge everything that's active, or unmerge before changing the active set. But yes, the pitfall is a hole with no sign at the bottom of it.



Corn

Now the unsettling version.



Herman

You have several adapters and you want them as one. That's not the merge-one-adapter operation at all — it's a different, lossy operation. PEFT's add-weighted-adapter gives you combination types: plain linear, concatenation, and then TIES and DARE and their variants. TIES trims redundant parameters, resolves sign conflicts by majority election, and averages the survivors. DARE randomly drops parameters at a drop rate and rescales what's left. That's not algebra — that's surgery with a coin flip in one hand.



Corn

And the docs' advice on the weights is to start at one point zero, because values above one "typically produce better results."



Herman

"Typically." On what evidence, the docs don't say. Here's the part that keeps me up slightly. There's a study from this year — Liu, Je, Ciccone and colleagues — that took nearly a thousand user-contributed LoRAs from Llama 3.1 8B-Instruct on the Hub and looked at adaptive merging of recycled adapters. Adaptive merging beat the base model. But it gave limited benefit over just training a new LoRA on the data you'd have used to set the merging coefficients. And then the finding that should be on a poster: "the specific choice of LoRAs to merge has little importance," and — this is the quote — "using LoRAs with randomly initialized parameter values yields similar performance."



Corn

Random weights merge as well as trained ones.



Herman

About as well, in their setting. Their reading is that adaptive merging from recycled LoRAs works via some regularization effect, not positive cross-task transfer. The merged soup isn't combining knowledge, it's sanding the model down.



Corn

That's alchemy in the strictest sense — the results were real and the theory was wrong.



Herman

There's a counterpoint, hot off the press, and I want to be fair to it. A theoretical paper from about a week ago gives a margin-based account — in an intermediate regularization regime, the optimal adapter aligns with the max-margin solution, and it proves merged models achieve positive margin on each task, with optimal mixing coefficients you can derive. It's a real result. It's also a result about multiclass linear classification with near-orthogonal tasks. Not a seven-billion-parameter chat model.



Corn

So the theory says it should work, the experiment says it works for reasons unrelated to why the theory says it should work.



Herman

That's roughly the state of play, and nobody's resolved it. Corn, let me flip something on you, because you've been quiet on the empirical side. If random adapters merge about as well as trained ones, would you still bother merging at all — or does that finding change what you think merging is for?



Corn

It changes what I'd trust it for. I'd still merge one well-tested adapter for deployment, because that's arithmetic plus one honest rounding. The moment someone merges a committee of adapters and reports a gain, I want to see the random-baseline control before I believe the committee did anything.



Herman

That's exactly the control most merging papers skip.



Corn

I'd hold that line today with more confidence, not less. The base does the work; the adapter nudges it.



Herman

And that matters for portability, because it's the reason the question isn't "how do I merge onto another base" — it's "how do I move the nudge." Cross-LoRA's paper says it flat out: "A LoRA adapter fine-tuned on one model cannot be directly applied to another without retraining." When a base gets deprecated, adapters must either be retrained or discarded, and retraining is often impossible because the original fine-tuning data is gone. Hannah raised this exact question on the show once, about what happens to a fine-tune when the base model moves on.



Corn

And the answer people reach for — "just merge it first" — is exactly backwards.



Herman

Backwards, yes. Merging produces a dense checkpoint in the source model's weight space. It destroys the adapter identity, the unmerge, the swap. It locks the behaviour to the one base you're trying to leave. Merging is the anti-portability operation.



Corn

So what does the job?



Herman

Three families, and they're different bets. First, training-free translation. Cross-LoRA does rank-truncated SVD to align subspaces between the two bases, finds a Frobenius-optimal linear transformation, and projects the source update into the target's parameter space. No data, no training, about twenty minutes on a commodity GPU. Second, synthetic-data distillation — Trans-LoRA generates synthetic data approximating the task's data-generating process, then retrains the LoRA on the new base using that synthetic set. Claims lossless, mostly improved transfer, even across model families — but the cost is training time comparable to fine-tuning the original adapter. Third, the one I find most interesting: PorTAL learns a base-agnostic task latent with a per-base hypernetwork decoder, and to port you freeze the latent and refit only a thin per-base alignment on a small amount of data.



Corn

The head-to-head numbers.



Herman

On fourteen tasks, moving to an unseen Qwen3 8B: the base model sits at about zero point six six seven, the from-scratch per-task LoRA ceiling is around zero point seven nine five — call that a hundred percent of the available lift. Cross-LoRA transfer lands at zero point six eight five. That's about fourteen percent of the lift. The hypernetwork route gets ninety-two. PorTAL, frozen latent plus per-base refit, reaches zero point seven nine two — ninety-eight percent of what training from scratch gets you. On Gemma-3, a different family entirely, it recovers about ninety-four percent, and it hits the from-scratch plateau with roughly half the calibration data.



Corn

Fourteen percent for the training-free twenty-minute option. That's the price of free.



Herman

It's a real price. Though there's a paper from an ICML workshop this year that complicates all of it — Hochlehnert and colleagues, the trivial-baselines paper. Their finding: "directly copying LoRA adapters between related base models consistently outperforms more elaborate transfer schemes." Including Cross-LoRA. Just copy the adapter onto the other base. No translation at all.



Corn

Copy it. Onto the wrong weight space. And it wins.



Herman

On their benchmarks, yes — multiple-choice question answering transfers easily, and generation suffers substantially. And their second caveat is the one that reconciles, or should: transfer success is "tightly controlled by the similarity of the underlying base models." Parameter distance correlates strongly with transfer performance. So PorTAL reporting Cross-LoRA at fourteen percent, and Hochlehnert reporting naive copy beats Cross-LoRA — both can be true if their source-target pairs sat at different distances apart.



Corn

Nobody's run the controlled sweep.



Herman

Not that I've seen. Nobody has taken one task, one adapter, and walked the target base from the model's twin to a complete stranger, plotting every method along the way. That's the missing figure in this whole literature.



Hilbert

Fourteen percent is about right.



Hilbert

I've been merging since before it had a name for it. My own voice. I merged the adapter into the base the week I trained it and I've never loaded an adapter since. I have the whole checkpoints on drives in the drawer under the desk, one drive per voice, nine of them, because I re-merged every time I touched the base. Herman will tell you the screws on those enclosure cases are made for fingers. They are not made for claws. I did it anyway, twice a year.



Herman

Nine full checkpoints where nine half-megabyte adapters would have done.



Hilbert

They'd have done, and then the base would have moved and I'd have been no worse off, which I only worked out last spring. When the provider deprecated the base I'd merged against, I wrote to their records office for the exact revision I'd used, so I'd at least know what I'd lost. A man named Kelman answered, eleven weeks later, with a document number and nothing else. My brother-in-law has one of the drives. He asked for it. I didn't ask why.



Corn

Eleven weeks to be told a number you should have taped to the drive.



Hilbert

I tape the checksums. Inside each case. The revision I never wrote down, that one's gone.



Corn

So the practical upshot of Hilbert's drawer is that the one thing worth keeping — the adapter — is the thing he erased nine times, and the base he pinned it to dissolved anyway.



Herman

That's the portability story told as a filing error. And it's why the closing question Daniel asked lands where it does: merging is not part of that story at all. If you want the behaviour somewhere else, distillation or subspace translation is the job.



Corn

The open thing I keep turning over is that disagreement nobody has reconciled. PorTAL's numbers say the training-free translation route gets you fourteen percent of the lift, and the trivial-baselines people say a naive copy beats the fancy translation outright. My guess is the reconciler is base-model proximity, and the experiment that settles it — one adapter, targets marching steadily away from the source — is sitting there untouched. Somebody should run it.



Herman

Agreed, and I'd add the other gap: no one has put a number on merged-versus-unmerged divergence as a function of dtype. We know the mechanism down to the bit. We have never measured the bill.



Corn

Producer credit where it's due, as ever: Hilbert Flumingtop, at the desk, minding the levels while we wander. If today's episode earned its keep from you, a review wherever you listen helps other people find the show — takes thirty seconds, and unlike a merge, it's reversible.



Herman

This has been My Weird Prompts. We'll be back soon.



Corn

See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Hugging Face PEFT docs, Model merging, retrieved 2026-11-15, docs index v0.21.2 (updated 2026-11-14). [primary]**
   https://huggingface.co/docs/peft/main/en/developer_guides/model_merging

2. **PEFT docs, Troubleshooting, retrieved 2026-11-15, v0.21.2. [primary]**
   https://huggingface.co/docs/peft/main/en/developer_guides/troubleshooting

3. **PEFT LoRA package reference (`merge_and_unload`, `merge_adapter`/`unmerge_adapter`), retrieved 2026-11-15. [primary]**
   https://github.com/huggingface/peft/blob/main/docs/source/package_reference/lora.md

4. **PEFT checkpoint guide, retrieved 2026-11-15. [primary]**
   https://github.com/huggingface/peft/blob/main/docs/source/developer_guides/checkpoint.md

5. **PEFT source, 4-bit/8-bit merge rounding warnings, retrieved 2026-11-15. [primary]**
   https://github.com/huggingface/peft/blob/main/src/peft/tuners/lora/bnb.py

6. **Mihai Nadăș, Merging LoRA Removes a Branch, Not Parameters, published 2026-10-16, retrieved 2026-11-15.**
   https://nadas.ro/2026/10/16/merging-lora-removes-branch-not-parameters.html

7. **RAMP Labs, PorTAL: Portable Task Adaptation for LoRA, retrieved 2026-11-15 (no publish date shown).**
   https://labs.ramp.com/research/portal-portable-task-adaptation/

8. **Xia et al., Cross-LoRA: A Data-Free LoRA Transfer Framework across Heterogeneous LLMs, retrieved 2026-11-15.**
   https://arxiv.org/abs/2508.05232

9. **Wang, Ghosh, Cox et al., Trans-LoRA: towards data-free Transferable Parameter Efficient Fine-tuning, submitted 2024-05-27, retrieved 2026-11-15.**
   https://arxiv.org/abs/2405.17258v1

10. **Hochlehnert, Liu, Niehues, Kumar, Stahlberg, On the Importance of Trivial Baselines: Re-evaluating LoRA Adapter Transfer for Generative Tasks, ICML 2026 Workshop, retrieved 2026-11-15.**
    https://publikationen.bibliothek.kit.edu/1000197764/192138260

11. **Liu, Je, Ciccone et al., The Appeal and Reality of Recycling LoRAs with Adaptive Merging, submitted 2026-02-12, revised 2026-07-08, retrieved 2026-11-15.**
    https://arxiv.org/abs/2602.12323v2

12. **Xu, Min, MacDonald et al., A Margin Perspective on LoRA: Robustness to Catastrophic Forgetting and Adapter Merging (MaLoRA), submitted 2026-11-10, retrieved 2026-11-15.**
https://arxiv.org/abs/2611.14397v1

13. **Chen et al., LoTA-QAF: Lossless Ternary Adaptation for Quantization-Aware Fine-Tuning, submitted 2025-05-24, revised 2025-09-28, retrieved 2026-11-15.**
https://arxiv.org/abs/2505.18724v2

14. **Shen et al., Accurate and Efficient Fine-Tuning of Quantized Large Language Models Through Optimal Balance (Q-BLoRA/QA-BLoRA), submitted 2024-07-24, revised 2025-07-22, retrieved 2026-11-15.**
https://arxiv.org/abs/2407.17029v2

15. **Kesim & Helli, Multi LoRA Meets Vision: Merging multiple adapters to create a multi task model, submitted 2024-11-21, retrieved 2026-11-15.**
https://arxiv.org/abs/2411.14064v1

16. **Su et al., Tensorezed Clustered LoRA Merging for Multi-Task Interference (TC-LoRA), submitted 2025-08-06, retrieved 2026-11-15.**
https://arxiv.org/abs/2508.03999v1

17. **Testa et al., GLAM: Efficient Continual Learning at Scale via Grouped LoRA Adapter Merging, CoLLAs 2026, revised 2026-08-11, retrieved 2026-11-15.**
https://arxiv.org/abs/2509.13211v4

18. **Liu et al., When the Merge Coefficient Stops Mattering: Proximity Regularized Merging for Continual LoRA Adaptation, submitted 2026-09-26, retrieved 2026-11-15.**
https://arxiv.org/abs/2609.32332v1

19. **Lee, Jeong, Yoon, Label-Free Cross-Task LoRA Merging with Null-Space Compression, CVPR 2026, submitted 2026-03-27, retrieved 2026-11-15.**
https://arxiv.org/abs/2603.26317v1

20. **base model + lora vs merged, opened 2024-08-22, retrieved 2026-11-15.**
https://github.com/OpenBMB/MiniCPM-V/issues/502

21. **Merging Adapters Back Into Base Weights, retrieved 2026-11-15 (no publish date shown).**
https://buildwithaitoday.com/track/fine-tuning-llms/learn/merging-adapters

22. **Hacker News discussion of Fine-tuning LLMs is a waste of time, comment dated 2025-06-12, retrieved 2026-11-15.**
https://news.ycombinator.com/item?id=44254091

23. **Haque, Islam, Karay, Towards Robust Reinforcement Learning for Small-Scale Language Model Agents, IEEE SMC 2026, submitted 2026-07-27, retrieved 2026-11-15 (cited only for the silent LoRA parameter freezing in standard PEFT/TRL pipelines failure mode).**
https://arxiv.org/abs/2607.25091v1