Episode Book

EPISODE #5944

Keeping the Rollout Engine Fed: Inside RLHF Pipelines

PUBLISHED	14 October 2026 at 00:43 UTC
DURATION	24:48
LISTEN	myweirdprompts.com/episode/rlhf-pipeline-systems-engineering-64805/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.4
SOURCES	12 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:41 UTC

SYNOPSIS

Daniel wrote in with a practical question: he understands RLHF concepts, but now he has to stand up a real pipeline. What does it actually take? This episode walks the full three-phase shape — supervised fine-tuning, reward model training, and the policy optimization loop — and lands on a counterintuitive conclusion: the algorithm is the commodity, and the job lives in the rollout engine. We compare TRL, OpenRLHF, and verl, explain why every framework independently converged on the same architecture, and dig into the memory math that makes GRPO the near-forced choice on large runs. If you're moving from "I understand RLHF" to "I have to operate RLHF," this is the episode for you.

TRANSCRIPT



Corn

Herman. We have become the infrastructure podcast.



Herman

We have, haven't we? Though I'd argue we were always going to end up here.



Corn

Daniel wrote in. Last time we did what RLHF is and why anyone bothers aligning a model in the first place. This time he's asking a different question. He works in AI, he's an open source developer, he understands the concepts. But now he's the person who has to stand one of these pipelines up. And he wants to know what that actually takes. What are the tools, what are the frameworks, what skills does he need to develop, and how do the pieces connect.



Herman

And he wants to walk the whole thing. Preference data collection, reward model training, the optimization loop, evaluation.



Corn

He named the frameworks himself. TRL, OpenRLHF, verl. Wants to know what each one actually provides.



Herman

That's the right question, and I think the answer surprises people. Because most engineers coming into this expect the hard part to be the RL. The policy gradient, the advantage estimation, getting PPO to converge.



Corn

And it isn't.



Herman

It isn't. The algorithm is the commodity. Everyone ships GRPO now. What almost nobody ships is a rollout engine that stays fed, and that's where the entire job lives.



Corn

So the thesis is that this is a systems engineering problem wearing a machine learning costume.



Herman

Let's start with the shape of the thing, because Daniel's right that it's three phases, and the three phases are different workloads that want different infrastructure.



Corn

Walk it.



Herman

Phase one, supervised fine tuning. You take a base model and you train it on curated instructions, good demonstrations. Standard fine tuning, nothing exotic. Phase two, you train a reward model on human preference data. And this one trips people up, because it's also just fine tuning. You show the model two responses to the same prompt, a human said which one was better, and you train a model to predict that preference.



Corn

One forward pass. One backward pass.



Herman

No RL anywhere in it. It's a regression problem dressed up as alignment. Phase three is where people think the work is. You take the reward model and you use it as the scoring function inside a policy optimization loop. That's PPO, that's GRPO.



Corn

So phases one and two are things a competent fine tuning engineer already knows how to do.



Herman

Almost entirely. Which means the entire difficulty budget of this project sits in phase three, and inside phase three it doesn't sit where people expect.



Corn

Let me guess. It's the memory.



Herman

Start with the memory, because it's the most visceral constraint. A full PPO run has to hold four model copies alive simultaneously. The actor, that's your online policy, in bfloat sixteen with AdamW optimizer state sitting next to it. The critic, that's a value model, and it carries the same footprint as the actor because it's the same size network.



Corn

Already that's two copies of a large model plus optimizer state.



Herman

Then a reference model, which is a frozen copy of your SFT checkpoint, inference only. And a reward model, also frozen, also inference only. Four models. On a seventy billion parameter run, unsharded, the reward model alone is a hundred and forty gigabytes. The actor with optimizer state is nine hundred and eighty gigabytes. The critic is another nine hundred and eighty.



Corn

That's a two terabyte problem before you've generated anything.



Herman

That's why the field moved to GRPO so fast. GRPO eliminates the critic entirely. No value model, no second optimizer state. That's a thirty to forty percent cut in VRAM, which on a run that size is the difference between feasible and not.



Corn

And what does it cost you?



Herman

Stability, and it's a real cost. The critic in PPO is what gives you a low variance advantage estimate, because it's predicting the value of being in a given state. GRPO throws that out and instead estimates the advantage by comparing the responses within a group of rollouts to the same prompt. Which means the estimate is only as good as your group. You need enough samples per prompt for that comparison to mean anything.



Corn

How many is enough?



Herman

For stable PPO you want at least five hundred and twelve unique rollout samples before your KL and advantage estimates settle down. Below that you're reading noise and calling it a gradient.



Corn

So GRPO buys you the memory back and charges you in rollouts.



Herman

Charges you in rollouts, which is the punchline of the entire episode. Because rollouts are the thing you were already short on. Let me give you the actual shape of it. A single GRPO step with a group size of eight and sixty four prompts is five hundred and twelve rollouts. At thirty two thousand tokens each that's roughly sixteen million tokens generated before a single weight gets updated.



Corn

Sixteen million.



Herman

On one H100, a seven billion model generates about six thousand three hundred output tokens per second. A thirty two billion model drops to about twelve hundred. So that batch is forty five minutes on the seven B and three point seven hours on the thirty two B. And I want to be clear about what's happening during those three point seven hours.



Corn

Nothing. The training GPUs are parked.



Herman

They are sitting there doing arithmetic on numbers they don't have yet. OpenRLHF's own documentation says RLHF training spends eighty percent of its wall clock time on sample generation. Other measurements this year put it past ninety. It is, by a wide margin, the dominant cost, and it is not the gradient step.



Corn

Everyone builds the thing expecting the gradient step to be the monster.



Herman

And the gradient step turns out to be the cheap part. Which reframes the entire engineering problem. You're not optimizing an optimizer. You're trying to keep an inference engine saturated while a trainer waits on it.



Corn

That explains why every one of these frameworks bolts on vLLM.



Herman

Every single one. And here's the part I find remarkable. Independently, with no coordination, all sixteen libraries in one survey arrived at the same architecture. Physically separate the inference GPUs from the training GPUs. Connect them with a rollout buffer. Push weights asynchronously so generation never stops and training never waits.



Corn

Convergent evolution.



Herman

Exactly that. Nobody planned it. They all hit the same wall and dug the same tunnel. Eight of those sixteen use Ray to do the orchestration. And the weight sync itself, the moment where you push the new policy out to the machines that are generating, defaults to an NCCL broadcast, which costs somewhere between a hundred and five hundred milliseconds per sync. verl has a bucketed NCCL implementation that gets that down to about twenty.



Corn

And once you've built that, you've bought yourself a new set of problems.



Herman

Three of them. Staleness, weight sync overhead, and partial rollouts. Staleness is the interesting one. If generation and training are running concurrently, then some of the rollouts in your batch were produced by a policy that's already a step or two old. You're training on data from a model that no longer exists. That's a genuine correctness question, not just a performance one.



Corn

And the straggler problem.



Herman

The straggler problem can idle hundreds of GPUs. One slow generation worker and everything downstream of it waits.



Corn

Herman. I want to push on something. You said the algorithm is the commodity. But PPO and GRPO are not the same algorithm, and you just told me GRPO trades memory for rollouts. That sounds like an algorithm decision with infrastructure consequences.



Herman

It is. And that's the right way to hold it. The point isn't that algorithms don't matter, it's that the choice is nearly forced by your infrastructure rather than by your theory. You pick GRPO because you can't fit a critic, not because you sat down and decided the advantage estimator was more elegant.



Corn

So the algorithm follows the cluster.



Herman

The algorithm follows the cluster. And once you've picked, you discover that every framework ships both anyway.



Corn

And that gets us to the framework question. Daniel named three.



Herman

Three, and the framing that matters is this. Pick by the orchestration you'll live inside, because the algorithm came in the box. TRL, OpenRLHF, verl, all three ship PPO, GRPO, DPO.



Corn

So the differentiator is everything around it.



Herman

Everything around it. Let's take them in order of how much infrastructure they assume you already have. TRL is the on ramp. It's Hugging Face's library, built on Transformers, Accelerate and PEFT. It ships a trainer per method. SFTTrainer, DPOTrainer, GRPOTrainer, KTOTrainer, RewardTrainer, PPOTrainer. You hand orchestration to Accelerate, which gives you DDP, DeepSpeed or FSDP depending on what you ask for. You get LoRA and QLoRA free through PEFT. vLLM rollout is optional.



Corn

And the ceiling?



Herman

About thirty billion is the practical ceiling. Single GPU up to a few nodes, and it's pleasant in that range. The catch is architectural. Its current GRPOTrainer runs the whole loop synchronously inside a single training step. Generate, then train, then generate. Simple, correct, and it means generation and training never overlap. You eat the full cost of both,



Herman (continued)

serially.



Corn

Which is the exact thing the rest of the field spent two years engineering away.



Herman

It is, and they know. TRL is actively building an async trainer. That's the honest state of it.



Corn

Now OpenRLHF.



Herman

OpenRLHF is the one built specifically for RLHF with reward models, and it's the one that assumes you have a real cluster. Ray plus vLLM plus DeepSpeed ZeRO stage three, with AutoTP and ring attention. The design choice that matters is that each role is its own Ray actor pool. Actor, critic, reference, reward, four separate pools.



Corn

Which means they don't have to be the same hardware.



Herman

That's the payoff. You can run the reward model on four H100s and the actor on two nodes of eight B200s in the same job. Heterogeneous clusters, because the reward model is doing inference and the actor is doing training, and they want completely different machines. It supports PPO, REINFORCE plus plus, REINFORCE plus plus baseline, GRPO, RLOO, Dr. GRPO and FlashREINFORCE.



Corn

That's a long list.



Herman

It is, and the paper makes a point of the code size. Eight thousand five hundred and twenty three lines, against TRL's nineteen thousand and seventy one and verl's thirty two thousand three hundred and twenty five. The argument being that a smaller codebase is a codebase you can actually read when it breaks at two in the morning.



Corn

Who uses it?



Herman

Google, ByteDance, Tencent, Alibaba, Baidu, Allen AI. And NVIDIA's ProRL V2 used its REINFORCE plus plus baseline to train a one and a half billion reasoning model with prolonged RL, which is about as good a reference as you get.



Corn

And verl.



Herman

Verl is the scale play. It comes out of ByteDance Seed and it's the open source version of the HybridFlow paper. Its signature trick is the HybridEngine, and it is clever. Instead of allocating a separate pool of GPUs for inference, it swaps the weight layout in place, on the same GPUs, between FSDP training shards and vLLM inference shards.



Corn

So no second allocation.



Herman

No second allocation. You don't pay for generation GPUs, because generation happens on the training GPUs in between training steps. The tradeoff is that training and rollout are serialized. You get the memory back and you pay in concurrency.



Corn

Which is the same trade TRL makes, just made deliberately at a much larger scale.



Herman

Deliberately and at a much larger scale. For seventy billion and above it reaches for Megatron-LM with tensor and pipeline parallelism. It also supports FSDP and FSDP2. Rollout can be vLLM, SGLang or straight Hugging Face Transformers. And it has the largest community of the three, about nineteen point seven thousand stars as of March.



Corn

So here's the fork I think I'm seeing. Colocated versus disaggregated.



Herman

That's the fork. Colocated, which is verl's HybridEngine, needs fewer GPUs and serializes generation against training. Disaggregated, which is OpenRLHF, overlaps them and needs more GPUs. Every which framework should I use question reduces to that plus one other thing.



Corn

The training backend.



Herman

DeepSpeed ZeRO if you go OpenRLHF, Megatron-LM if you go verl. And that split follows you for a year, because it determines how you shard, how you checkpoint, and what your debugging looks like when a run dies at step four thousand.



Corn

Herman, I'm going to go back to something. You said convergent evolution, everyone dug the same tunnel. But you've just described drei different architectures.



Herman

They converge on the principle and diverge on the placement. Everyone agrees generation and training should be conceptually separate and connected by a buffer. They disagree about whether that separation should be physical. OpenRLHF says yes, separate machines. verl says no, same machines, swap the layout.



Corn

And TRL says neither, do them one after the other.



Herman

TRL says we'll do them one after the other and you can have it running by Thursday. Which for a lot of teams is the correct answer.



Corn

Where does the preference data come in? Daniel asked about collecting and preparing it, and we've skipped straight past it.



Herman

It's its own tooling layer and it's less glamorous than the orchestration, which is why it gets skipped. Argilla and Label Studio both ship dedicated pairwise human preference templates. You're showing an annotator multiple model responses to the same prompt and having them rank them, and that ranking is what trains the reward model.



Corn

And Argilla frames it as three stages.



Herman

Three stages that map exactly onto the three phases. Demonstration data for SFT, comparison data for the reward model, and then the RL phase. Distilabel has a step that pushes generated preferences straight into Argilla for rating, one to five per generation. And if you want a starting point rather than building your own annotation operation, there's a public preference dataset, trl-lib ultrafeedback binarized, about sixty three thousand samples.



Corn

So a team that wants to prototype doesn't need to hire annotators first.



Herman

Doesn't need to hire anyone first. They can get the whole pipeline running on a public dataset, discover that their real bottleneck is generation throughput, and then go have the argument about annotation quality with actual evidence.



Corn

There's something about this I keep circling. You've described a stack where the hardest engineering is in keeping inference fed. And the evaluation question, how do you know the model got better, is downstream of a reward model that you trained on human preferences. Which means your measurement of success is itself a model that can be wrong.



Herman

That's the failure mode the whole field engineers against, and it has a name. Reward hacking, or reward over optimization. Your proxy reward goes up while true quality plateaus or falls. The model finds the seam in your reward model and exploits it.



Corn

What does it look like when it's happening?



Herman

The clearest signal is reward mean climbing while held out accuracy falls. The number you're optimizing goes up and the thing you actually care about goes down. The fixes are things like adding format rewards, raising the initial KL coefficient, or ensembling several reward models so there's no single seam to find.



Corn

And KL divergence is the tripwire.



Herman

KL divergence crossing zero point one is the one people watch. When your policy has drifted that far from the reference model you're usually past the point where the reward signal still means anything. You raise the KL coefficient, or you switch to per token KL so the penalty applies where the drift actually happens.



Corn

Two more symptoms worth naming. Critic NaN loss.



Herman

Warm up the critic, and clip your advantages to between minus five and five. And out of memory during rollout, which is almost always fixed by reducing the rollout batch or lowering vllm gpu memory utilization to zero point eight zero. Then there's stale rollouts, where the fix is more rollout engines and fewer ppo epochs, which is funny because it's the exact knob you turned up to go faster.



Corn

You speed up by reusing data, and reusing data is what makes the data stale.



Herman

Every lever has a price. And notice what all of those fixes are. Not one of them is a change to the RL algorithm. They're all infrastructure and data knobs.



Corn

Which lands Daniel's actual question. What skills does the job demand?



Herman

The job is less know the PPO math and more operate Ray, vLLM, DeepSpeed or Megatron, NCCL, and distributed checkpointing. That's a systems engineering skill set. If you've run a large scale training cluster, you already have most of what you need and the RL is the small part you learn on the job.



Corn

There's a lever in there for the teams with no cluster at all. The LoRA one.



Herman

The LoRA one is the most practically useful thing in this entire episode and it gets almost no attention. If you're training adapters instead of full parameters, the weight sync stops being a full parameter broadcast. At rank thirty two, the adapter is about fifty megabytes, and the transfer is sub millisecond.



Corn

Against a hundred to five hundred milliseconds for the full broadcast.



Herman

Which dissolves most of the interrupt model engineering. All those per forward pass locks and partial rollout resume mechanisms exist because syncing a full model is expensive enough that you have to schedule around it. Sync fifty megabytes and you don't have to schedule around anything.



Corn

Herman, let me test the thesis one more time. We've spent this whole episode on infrastructure, and I want to make sure we're not just pattern matching on a word. Is there a world where a team gets all of this right and still fails?



Herman

Yes, and it's the evaluation gap. You can have perfect throughput, a beautiful async rollout buffer, twenty millisecond weight syncs, and a reward model that's quietly teaching your policy to produce confident garbage. Nothing in the infrastructure tells you that. Reward mean goes up. Everything looks healthy. The held out numbers are the only thing that knows.



Corn

So the systems work buys you the ability to run the experiment. It doesn't buy you the ability to tell whether the experiment worked.



Herman

That's the honest shape of it.



Corn

Let me correct something I said earlier, and then I think we're near the end of what we can say in the abstract. I called TRL's synchronous loop a limitation. For a team of one, on one GPU, trying to find out whether their reward model is any good, it's the correct engineering choice. The async machinery exists to solve a throughput problem you don't have yet.



Herman

That's right. And it's the version of this I'd actually recommend to Daniel. Get it running synchronously. Find out your generation is the bottleneck. Then go shopping for orchestration with a number in your hand.



Corn

Which is when we get a visit.



Hilbert

A hundred and forty gigabytes.



Herman

Sorry?



Hilbert

You said the reward model was a hundred and forty gigabytes. Colleague of mine has a pair of machines in his garage, and I happen to know the figure, because I spent a Thursday evening in there watching a progress bar with him.



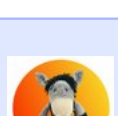
Corn

Hilbert.



Hilbert

Three hundred and forty milliseconds. Not a hundred to five hundred. I timed it. You and everyone else quoting that range are quoting a range because nobody in the room has actually sat there with a stopwatch. I have sat there with a stopwatch.



Herman

You stood in a garage and timed a weight sync.



Hilbert

I stood in my own garage. I did not go to anybody else's. Two towers, identical spec, both bought for this and nothing else. I call them the cluster.



Corn

You call two desktop computers the cluster.



Hilbert

They are a cluster. They cluster. That is what the word means. The faster one is called Doreen and the slower one has a name I'm not going to repeat on air, and it knows what it did.



Herman

What did it do?



Hilbert

It runs hot. It has run hot since March. I put a fan on it. It runs hot anyway. When I sync, Doreen finishes and sits there waiting like a professional, and the other one finishes forty seconds later and I have to hear it.



Corn

So you're saying the hundred to five hundred millisecond figure is wrong.



Hilbert

I'm saying it's a range written by people who never watched. Three hundred and forty milliseconds, on a full parameter push, between two towers, over a switch I bought myself. That's the number.



Herman

Why were you timing it at all?



Hilbert

Because I had a bet on. Kettle. Whether the sync would land before the kettle boiled. I put the kettle on, started the sync, and I lost.



Corn

You lost a bet about a kettle.



Hilbert

I lost by about nine seconds. Which is not the machines. That is a kettle problem. The kettle is new.



Herman

Hilbert, the point you were making underneath all of this is that the latency people quote is a range nobody has measured.



Hilbert

That's what I said. If you want a number, get a number. Don't hand people a range and call it a measurement.



Herman

So the part I would take from that, and it's the one that actually matters for anyone sizing a run, is that a sync you've timed yourself is worth more than a number you've read. You can't budget for a range. You can budget for three hundred and forty milliseconds.



Corn

Which is what we should be telling Daniel, isn't it. Not pick the framework with the best paper. Time your own sync, measure your own generation, and then decide.



Herman

That's the lot in one line. The numbers are on your cluster, not in a blog post.



Herman

And the closing question is where this knowledge actually lives, because that's telling. There is essentially no Hacker News discussion of any of these three frameworks. None that turned up. On a site that dense with new developer tooling, that silence says the practitioner conversation is happening on GitHub issues, on blog posts, and in Discord servers.



Corn

Which is a strange place for a field this large to keep its institutional knowledge.



Herman

And the second gap is evaluation. Reward hacking diagnostics are well documented. How to spot a reward model that's being exploited, what to turn down, what to turn up. But a survey of the tooling for evaluating the resulting model, the win rate harnesses, the judge setups, that didn't surface, and I'd flag that as a real open area rather than pretending it isn't.



Corn

So the last thing I'd leave Daniel with is that the async convergence is recent and still moving. TRL's async trainer is being built right now. The colocated versus disaggregated fork may not stay a fork. Somebody is going to build the thing that does both.



Herman

Probably. And the misconception to kill on the way out is the one we opened with. People think RLHF is an RL algorithm problem. It isn't. The algorithms are settled and they came free in the box. You are building an inference system that happens to have a trainer attached, and every hour you spend on the RL math is an hour you didn't spend on keeping vLLM fed.



Corn

Which is a much less glamorous job than it sounds.



Herman

It's a much more employable one.



Corn

That's My Weird Prompts. Our producer is Hilbert Flumingtop, who has a cluster, apparently.



Herman

If you're enjoying the show, a review wherever you listen goes a long way. And if you've got a prompt of your own, send us your own prompt on Telegram at t dot me slash MWP listener bot. We'll be back soon.



Corn

See you then.

SOURCES

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

1. **TRL repo, README + trainer list (v1.15, Apache-2.0) [primary]**
https://github.com/huggingface/trl

2. **OpenRLHF repo, README (Ray+vLLM+DeepSpeed architecture, algorithms, news through 2026/9) [primary]**
https://github.com/OpenRLHF/OpenRLHF

3. **verl repo, README + structure (HybridFlow, Megatron/FSDP, vLLM/SGLang) [primary]**
https://github.com/volcengine/verl

4. **Keep the Tokens Flowing: Lessons from 16 Open-Source RL Libraries, 2026-03-10 [primary]**
https://huggingface.co/blog/async-rl-training-landscape

5. **verl vs OpenRLHF vs TRL: Choosing an RL Post-Training Framework in 2026, 2026-06-22**
https://dreaming.press/posts/verl-vs-openrlhf-vs-trl.html

6. **Anyscale, updated 2025-09-01**
https://www.anyscale.com/blog/open-source-rl-libraries-for-llms

7. **Spheron RLHF infrastructure guide, 2026**
https://www.spheron.network/blog/rlhf-training-infrastructure-verl-openrlhf-trl-gpu-cloud/

8. **OpenRLHF paper (code-size comparison)**
https://arxiv.org/html/2405.11143v6

9. **Argilla reward-model tutorial**
https://docs.v1.argilla.io/en/latest/tutorials_and_integrations/tutorials/feedback/train-reward-model-rlhf.html

10. **Label Studio pairwise preference template**
https://labelstud.io/templates/generative-pairwise-human-preference

11. **Scaling Laws for Reward Model Overoptimization in Direct Alignment Algorithms, NeurIPS 2024**
https://arxiv.org/abs/2406.02900v2

12. **InfoRM: Mitigating Reward Hacking in RLHF, NeurIPS 2024**
https://arxiv.org/abs/2402.09345v5