Episode Book

EPISODE #6191

What Actually Happens During Fine-Tuning

PUBLISHED	21 October 2026 at 23:58 UTC
DURATION	21:05
LISTEN	myweirdprompts.com/episode/what-happens-during-fine-tuning-67730/
HOSTS	Corn, Herman, Larry, Hilbert
SCRIPT MODEL	anthropic/claude-sonnet-5.5
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.19
BOOK GENERATED	22 October 2026 at 07:46 UTC

SYNOPSIS

You copy the notebook, change the dataset path, hit run, and hours later a folder appears with new weights that understand your accent. What happened in between? This episode walks the full training step — safetensors files, batches, forward passes, the loss, the backward pass, and Adam's quiet bookkeeping — and explains why a 7B full fine-tune is a memory problem before it's a compute problem. Along the way: why the first five minutes of a fine-tune deliver the cheap gains, how overfitting turns a model into a parrot that can recite training text, and the tricks (mixed precision, gradient checkpointing, gradient accumulation) that let one card do a job that wants a hundred gigabytes.

TRANSCRIPT



Corn

A progress bar crawls across a notebook cell on a rented GPU. Some minutes later a folder appears with one big file in it, and when you point your speech recognizer at the file, it understands your accent better than it did that morning. Nobody in that story did anything you could call understanding. Daniel has lived that story with several Whisper-style models and one language model, mostly on Modal, and his question is about the gap in the middle. You paid for the machine, you handed it weights and a dataset, you got different weights back, and for the stretch in between you were a spectator.



Herman

Which is the normal experience. You copy the notebook, you change the dataset path, you run it, and the first time it exports something that works you feel like a wizard. Then you start wondering what the wizard actually did.



Corn

Daniel's picture is that the training data gets enmeshed into the weights. Not stored as a lookup table, just worked into the numbers that were already there.



Herman

That picture is close, and I want to give it a proper run. Let me start where the run starts, which is a file. The base model is a file on disk, and the format most people meet now is safetensors. It's almost insultingly simple. The first eight bytes say how long the header is. The header is a block of plain JSON listing every tensor by name, with its shape, its data type, and where its bytes begin and end. After that it's raw numbers, packed one after another. No code, nothing clever.



Corn

So the whole brain of the model is a table of contents and then a very long list of numbers.



Herman

A very long list. For a Whisper small model, if I remember right, it's around a quarter of a billion numbers. A seven or eight billion parameter language model is thirty times that or more. Step one of your run is that the script reads the header, finds each tensor, and copies those bytes into the memory on the GPU. At that moment the card is holding a frozen snapshot of what somebody else trained.



Corn

Then the dataset shows up.



Herman

In pieces. The dataset never gets loaded as a whole thing the way you picture it. For your speech models, each audio clip becomes a spectrogram, a grid of numbers describing energy at different frequencies over time, and the transcript becomes a row of token ids. For a language model, the text becomes token ids and nothing else. Then the loader hands the GPU a batch, which is a handful of those examples stacked together. That's the unit of work. Everything that follows happens once per batch.



Corn

And what happens to a batch?



Herman

A forward pass. The numbers go in at the bottom and flow through every layer, each layer multiplying them by its weights, and at the top the model produces a guess. For a language model the guess is a probability for every possible next token. For Whisper it's the same thing, one text token at a time, conditioned on the audio. And here's the part that makes training different from just using the model. You know what the right answer was, because it's sitting in your dataset. So you can score the guess. That score is the loss. One number per batch that says how surprised the model was by the truth.



Corn

One number. Forty minutes of electricity for one number.



Herman

For a lot of numbers, one per batch, thousands of batches. But the loss alone isn't useful yet. The magic is the next step, the backward pass. Starting from that loss, the framework walks back through the model layer by layer and works out, for every single weight, which direction would have made the loss smaller and by how much. That's the gradient. One gradient value per weight. So after a backward pass the GPU is holding a second set of numbers the same size as the model.



Corn

The model and its report card.



Herman

Roughly. And then the optimizer takes the report card and actually changes the weights. Plain gradient descent would just subtract a small multiple of the gradient. What most people run is Adam, and Adam is a little hoarder. For each weight it keeps two more numbers. A running average of the recent gradients, which is a kind of momentum, and a running average of how large and noisy those gradients have been, which it uses to scale the step for that particular weight. So every parameter in the model has three companions during training. Gradient, momentum, variance.



Corn

That's a lot of baggage for one number.



Herman

Add it up with the common mixed precision recipe and it's about sixteen bytes per parameter. Two for the weight, two for the gradient, and twelve for the optimizer's bookkeeping, which includes a full precision master copy of the weight. For a Whisper small that's a few gigabytes, which fits anywhere. For a Whisper medium it's around twelve, fine on a decent card. For a seven billion parameter model fully fine-tuned it's well over a hundred gigabytes before you've stored a single activation. So a full fine-tune of a big model is a memory problem long before it's a compute problem.



Corn

Which is why the language model run was a different experience from the speech ones, I'd guess. I'd want to hear Daniel's side of that, but that's the arithmetic.



Herman

It all comes down to that bookkeeping being scaffolding. The gradients, the momentum, the variance, the master copy, all of it exists only while training is running, and when the last step finishes it gets thrown away. And that is why the exported weights are the only thing you carry into inference.

L

Larry

Friends, are your weights feeling unloaded? Do your forward passes leave you flat and your losses piling up at home? Introducing Fine Tune Tonight, the first nightly remedy that reaches deep into your backpropagation and gently optimizes the whole you. Just one epoch before bed and you will wake up with fully exported weights, a lighter loss, and an inference so smooth your neighbors will ask what changed. Fine Tune Tonight is not for everyone. Side effects may include sudden epoch elongation, optimizer step slippage, unexpected gradient vanishing during dinner parties, forward pass repetition, the sensation that your loss is being watched, mild weight decay, an audible hum from your backpropagation, exporting weights you do not recognize, and in rare cases, remembering the entire training set. Ask your doctor if Fine Tune Tonight is right for you.



Corn

Sixteen bytes per parameter on the way in, two on the way out. I'd want the other fourteen refunded.



Herman

Nobody's refunding the electricity. Let me finish the step, though, because I skipped the interesting part. The optimizer changes each weight by a small amount, and the size of that amount is the learning rate. It usually starts near zero, ramps up over the first stretch of steps, then shrinks as training goes on. I'm describing the typical shape, not any one recipe. Too big and the loss bounces around or blows up. Too small and you've paid for a lot of GPU time to change almost nothing.



Corn

This is where I'd have said the data gets enmeshed. Every step moves every weight a hair in whatever direction made that batch less surprising.



Herman

Every weight, yes, and that's why enmeshed is a good word. There's no spot where example number four hundred is filed. A batch tugs on millions of weights at once, the next batch tugs on the same millions in a slightly different direction, and what survives is whatever the tugs have in common. One pass through the whole dataset is an epoch. Run several and the model sees every example several times.



Corn

Meanwhile the notebook prints a line every so often with a loss in it, and I've always treated those like weather reports.



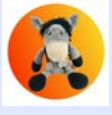
Herman

They're the only window you get. Every few dozen steps it averages the recent batch losses and prints one. The batches are shuffled each epoch, so the model can't lean on the order, and the number is noisy, it jumps around even when things are going well. You're looking for the trend. Early on it falls fast, because the base model is already close and the first few hundred steps fix the obvious gaps, like your accent or your jargon. Later it flattens, and that flat stretch is where the careful part of the run lives. Also, every so often it saves a checkpoint, a copy of the weights mid-run, so a crash or a preempted machine doesn't cost you the whole job.



Corn

You're saying the exciting part of a fine-tune is the first five minutes.



Herman

The cheap gains, yes. After that you're paying for polish, and the polish is where the overfitting risk starts. Which is a nice setup for your question.



Corn

Why would you run several? If it's seen the clip once, it's seen the clip.



Herman

Because one tug at a small learning rate barely moves anything. It's the same reason you don't learn a language from one listen to each sentence. Here's a question for you, Corn, since you're the one who tells people to stop practising. What would you expect to happen if you run too many?



Corn

It learns the sentences instead of the language. Parrots them.



Herman

Close to exactly that. This is overfitting, and it's the thing I'd like Daniel's picture to carry with it. The data isn't stored as records, but the weights can encode specific strings well enough to reproduce them. Researchers pulled verbatim training text out of an early GPT model, phone numbers and chat logs included, and some of those strings had appeared in only one document. A small dataset run for many epochs is the exact setup where a model starts to recite. So when a run goes well you'd want to hold some of your data back, never train on it, and watch the loss on that held-out set. When the training loss keeps falling and the held-out loss turns around and rises, the model has stopped generalising and started memorising.



Corn

So the number you actually care about is the one the model never trained on.



Herman

That's the one a good notebook plots on the same chart. And the training loss alone going down tells you almost nothing, since you could get that by memorising.



Corn

I notice the thing taking sixteen bytes a parameter doesn't scale to the laptop on my desk. How does anyone do this on one card?



Herman

A few tricks, and each one is a trade. Mixed precision first. The forward and backward passes run in sixteen-bit numbers, which are half the size and much faster on modern cards. But the update to a weight can be tiny, smaller than what a sixteen-bit number can register next to a big weight, so it would round to nothing. So the optimizer keeps a full precision master copy, adds the tiny update there, and rounds down to sixteen bits only for the next forward pass. There's also loss scaling. Gradients in sixteen-bit can underflow to zero, so the original recipe multiplies the loss up before the backward pass and divides it back down before the update. That was worked out for the older sixteen-bit float. What current frameworks do with bfloat sixteen, I'm less sure about, and I'd check before repeating anything I said about it.



Corn

So the model runs in a cheap number format and the real weights live in a safe at the back.



Herman

That's fair, and it's the one image I'll allow. Second trick is gradient checkpointing. A backward pass needs the intermediate values from the forward pass, called activations, and keeping all of them eats memory. So the framework keeps only a few and recomputes the rest when the backward pass needs them. The classic result is a huge drop in memory for roughly a third more running time. The card forgets things on purpose and does the work twice.



Corn

A GPU that takes notes and then bins them.



Herman

Third is gradient accumulation. You want a batch of sixty-four but only eight fit. So you run eight, keep the gradients, run eight more, add them up, and only after eight rounds do you take the optimizer step. Same maths, spread over time.



Corn

That one I like. Cheating at arithmetic, legally.



Herman

Would you have guessed that before today, Corn? That the card is mostly holding things it will never export?



Corn

I'd have guessed the model was the big part. I'd have been wrong, and slowly.



Herman

The last trick is the one that changes what your output is. LoRA. With full fine-tuning, every existing weight tensor gets updated, so what you export has the same shapes as what you loaded and every number is slightly different. With LoRA you freeze the base. The original weights don't move. Next to selected weight matrices you attach two small thin matrices whose product has the same shape as the big one, and only those get gradients and optimizer state. The paper's headline was up to ten thousand times fewer trainable parameters than full fine-tuning of a huge model, and about a third of the memory.



Corn

Hold on. If the base doesn't move, where did Daniel's enmeshing happen?



Herman

In the small matrices. All the tugging lands there. At inference, the model's output is the frozen weights plus the adapter's contribution, and the adapter file is a separate small object. You can ship it alone and load it onto the base. Or you can merge, where the adapter's contribution is added into the original weight tensors and the adapter is dropped. After a merge you're back to a normal-shaped model, and it looks exactly like a full fine-tune from the outside.



Corn

So there are two different things Daniel might have been holding. The small adapter file, or a merged model that's just a big file again.



Herman

Which one he got depends on the script, and that's worth looking at. One more wrinkle, and this isn't from the library docs I trust most, so take it as likely rather than certain. If the base was loaded in four-bit to save memory, the advice I've seen is to merge into a sixteen-bit copy of the base, not into the four-bit weights.



Corn

Fine, so training is over. The loss stopped doing anything interesting. What is the export actually doing? Is that more training?



Herman

No training. Export is bookkeeping. The framework writes the weights out as a safetensors file, same simple layout we started with, header and raw bytes. For your speech models there's often a second conversion. If you want faster-whisper to run it, you run a converter to the CTranslate two format, and that's also where you pick a stored precision, sixteen-bit or eight-bit. Then at load time you can choose a compute precision separately from how it's stored. For llama.cpp style tools the equivalent is a conversion to GGUF, but I haven't walked through that one, so I won't pretend to.



Corn

So all that mattered, the gradients, the momentum, the safe at the back, is gone, and you're holding a file that looks like the one you started with.



Herman

The same layout, the same shapes if you merged, different numbers. That's the whole trick. The first file was what somebody else's data tugged into the model. The second file is that plus the tugs from yours. Nothing else survives the run.



Corn

And everything the loop was doing in the middle is gone. All we have is the arithmetic. After all that compute, what you get out is a long list of...



Hilbert

Fan speeds. Teodor's got his as a list of fan speeds.



Herman

Numbers, but I'll bite. Who's Teodor?



Hilbert

Met him on the night ferry out of Ancona. He lent me his reading glasses for the timetable and has them still. I rang him about twenty minutes ago, while you were on the sixteen bytes.



Corn

He's who you called about the glasses.



Hilbert

About the glasses, yes. He runs the cooling for a rack of rented GPUs in a disused funicular station outside Kranj. Rented by the hour, same as you were talking about. His day starts at four. He walks the length of the aisle once an hour and it takes him nineteen minutes, because he stops at every fourth card and hums at it.



Herman

Hums, to check the fan bearing?



Hilbert

He says a bad bearing hums back a different note. Herman, you said nudge eleven times. Teodor has never nudged anything. He puts a hand flat on the chassis and he knows.



Corn

Eleven. I wasn't counting.



Hilbert

At eleven in the morning he stands at the end of the aisle and reads out every customer's loss number to the room. He doesn't trust the dashboard. Last Tuesday it was point four one to point three eight and he was very pleased, and he said so to the goat.



Herman

There's a goat.



Hilbert

Radovan. On the payroll. Fire warden. Teodor's pay is six hundred and ten euros a month and Radovan's is four hundred, which Teodor says is fair, since Radovan has never once let the room go above twenty-six degrees.



Corn

That's a better temperature record than most data centres keep.



Hilbert

He sent me a fan. Retired from slot nine. It's on top of my fridge now, plugged into the hall extension, blowing at the kettle. Hannah wants to know why the kettle boils so fast.



Herman

A rack fan will move a lot of air, and a kitchen outlet will keep it spinning. The bearing wasn't designed for a draught full of steam, though, so listen for that note he mentioned.



Hilbert

It's not humming yet. He said it would take eleven weeks. Postage was eleven euros forty, and he wants the glasses sent back first.



Corn

Forty minutes of electricity, one number. Teodor reads it aloud at eleven, and it sounds like the only line on the dashboard he believes.



Herman

That held-out loss line from before is the one I'd put in front of him. The one the model never trained on. If that one goes down, the run did something. If only the training line goes down, you paid to teach a model to recite your dataset.



Corn

So the one thing to carry out of this. The run is a loop of guess, score, assign blame, take a small step, and the output file is the same kind of object you loaded with different numbers in it. There's no new part, only different tugs on every existing one. Or on a small attached pair, if it was LoRA.



Herman

And the file doesn't remember how it got there. It carries the result of the tugs and, at the edges, sometimes more of your data than you meant to give it.



Corn

That's what I'd check next time, Daniel. Not whether the bar finished, but which loss line went down and what kind of file landed in the folder. For me the open question is the one about the middle of the run. If the held-out number is the only one anyone trusts, who's actually reading it while it goes? A man in a funicular station, apparently, out loud, to a



Corn (continued)

goat.



Herman

Thanks to Hilbert Flumingtop for producing, and for keeping the whole thing running while we talk about the part nobody sees.



Corn

This has been My Weird Prompts. You can find every episode at my weird prompts dot com.



Herman

See you tomorrow.