Episode Book

EPISODE #5805

Freeze the Model, Move the Labels

PUBLISHED	9 October 2026 at 10:54 UTC
DURATION	22:31
LISTEN	myweirdprompts.com/episode/dynamic-label-sets-retraining/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.2
SOURCES	16 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:40 UTC

SYNOPSIS

Daniel built a home inventory classifier that sorts items into boxes. Then he bought three new boxes, and the model went stale. His instinct was to retrain weekly — but the real problem isn't the model, it's that his label set keeps changing. This episode digs into the two architectural answers: retrain on a moving label set, or freeze the encoder entirely and put the labels in a vector index instead. Along the way: why SetFit training costs about two and a half cents, why synthetic training data can beat real labels on one benchmark and fail silently in your garage, and why "incremental vs. full retrain" stops being a real decision at small scale.

TRANSCRIPT



Corn

Most people hear "frequent retraining" and immediately start pricing GPUs. That's the frame this whole topic usually arrives in. It's the wrong frame.



Herman

It's wrong because the hardware question was settled years ago and nobody told the people still asking it.



Corn

Which is roughly where Daniel is standing. He's built a home inventory system. Assets, storage units, associations between them, QR codes and printed labels on everything. Every box has a description field explaining what it's for. He trained a small classifier to take an item name and tell him which box it goes in, and it worked.



Herman

And then he bought three new boxes.



Corn

Three new boxes, three new purposes, and now the classifier is stale. So his question is how to set the thing up so it can be retrained often, ideally automatically. He's even got a plan for the training data, generating synthetic item-to-box pairs with AI. He wants to know whether weekly incremental retraining is practical for a model trained on a few hundred boxes, whether it can live in a cron job, and how production systems deal with a label set that's bounded but keeps changing.



Herman

The box list is the interesting part. Not the model.



Corn

Right. And before we answer any of it, it's worth noticing that Daniel has assumed retraining is the answer. That assumption is the first thing to examine.



Herman

Because this isn't a classification problem. It's a problem where the label set moves, and the classification part is almost incidental.



Corn

Say more.



Herman

A normal classifier has a closed world. You decide the classes up front, train, ship it, and the world stays put. Here the world doesn't stay put. Daniel buys a box, the world has a new class. He retires a box, the world has one fewer. The model isn't hard. A few hundred classes over short item names is a tiny problem. The label set is a moving target, and that's a different kind of problem with its own literature.



Corn

Intent classification would be the obvious cousin.



Herman

Exactly that shape. You ship a voice assistant with thirty intents, then the business adds ten more, then retires four. Document classification with a taxonomy that evolves is the same thing at a slower tempo. There's a paper from ACL back in 2019 that states the problem as cleanly as anyone has. Traditional text classifiers predict over a fixed set of labels, but in plenty of real applications the label set is changing all the time, and they propose replacing the fixed output layer with a learned metric space instead of just retraining forever.



Corn

So there's a name for the shape of Daniel's problem.



Herman

There is, and there are two architectural answers to it. Pattern A, you retrain the classifier when the labels change. Pattern B, you don't retrain at all. You freeze the encoder, and you put the label set in a vector index instead of in the model's weights.



Corn

And the tension for the episode is that Daniel is assuming Pattern A, and the research keeps pointing at Pattern B.



Herman

That's the spine of it. Let's take his instinct seriously first, though, because it isn't wrong. It's just cheap in a way he may not have realized.



Corn

So the SetFit architecture. What actually happens when he trains this thing?



Herman

Two pieces. A sentence-transformer encoder that turns text into a vector, and a small logistic head on top that turns that vector into a class. The encoder is doing the semantic lifting, which is the part Daniel actually needs, because his edge case is real. A box for soldering supplies and a box for plug types both look like "electronics" to a keyword matcher. The encoder knows soldering irons and HDMI cables aren't the same kind of object.



Corn

And the reason SetFit matters here is the cost of training it.



Herman

The cost is the whole point. Eight labeled examples per class gets you a model competitive with fine-tuning a large transformer on thousands of examples. And the training time, on an NVIDIA V100 with eight examples per class, is thirty seconds. At a cost of about two and a half cents.



Corn

Two and a half cents.



Herman

The ONNX tutorial describes it as a few seconds on a GPU to minutes on a CPU. There's a walkthrough where someone trains on a CPU in under two and a half minutes and gets ninety-four and a half percent accuracy on sixteen samples.



Corn

So "is weekly incremental retraining practical for a few-hundred-class model" has an answer, and the answer is that it's not merely practical. It's embarrassingly cheap. The compute isn't the constraint.



Herman

It isn't. And here's the part that should make Daniel feel good about his instinct. His synthetic data idea isn't a workaround. It's the documented technique. SetFit's own zero-shot tutorial says the main trick is to create synthetic examples that resemble the classification task and train on them. There's a function that generates N synthetic examples per class from a template, and the docs say eight per class usually works best.



Corn

Which means he could generate the training pairs for three new boxes without touching a single real item.



Herman

And it performs. On the emotion benchmark, purely synthetic training hit zero point five three four five accuracy. The transformer zero-shot pipeline got zero point three seven six five. Training on eight real examples only got zero point four seven zero five. And eight real examples augmented with synthetic ones got zero point six one three.



Corn

Synthetic-only beat real-labels-only.



Herman

On that dataset, yes. And I want to flag the caveat because it matters for Daniel's situation. The docs say the difference depends strongly on the dataset. So for a home inventory system with sparse real labels, synthetic-only is both an opportunity and a trap. It might work beautifully. It might be quietly wrong in ways he won't notice until he's standing in the garage holding a monitor.



Corn

Let me push on that, because "it depends on the dataset" is the kind of hedge that sounds safe and tells you nothing. What's the actual mechanism? Why would synthetic data be great on one task and quietly wrong on another?



Herman

The mechanism is coverage. Synthetic generation works when the generator knows what the real inputs look like. If you're generating examples for a sentiment classifier, the model has read millions of product reviews and movie reviews. It knows the register. It knows what "the battery died after a week" sounds like. So the synthetic examples land close to the real distribution, and the classifier learns the right decision boundary.



Corn

And for Daniel?



Herman

For Daniel, the generator has never been inside his garage. It knows "HDMI cable" and "soldering iron" as general concepts. It does not know that he owns a specific twenty-year-old soldering station with a broken tip, or that his "plug types" box is actually ninety percent travel adapters from a trip in 2016. So the synthetic items will be plausible and slightly generic. The classifier will learn a boundary that's correct for the average household and wrong at the edges of his.



Corn

So the failure isn't catastrophic. It's a bias toward the generic.



Herman

Which is exactly the failure mode you don't catch, because the generic cases are the ones you test with. You type "HDMI cable," it goes to the right box, you feel good. You type "travel adapter," it goes to the right box. Then you type the weird thing, the thing that only exists in your house, and it lands in the wrong place, and you assume you typed it wrong.



Corn

That's a good argument for seeding with real examples even when synthetic is available.



Herman

It's a good argument for a hybrid. Even a handful of real items per box anchors the synthetic generation to your actual distribution. The benchmark number I quoted, eight real plus synthetic at zero point six one three, is the best of the four. That's not a coincidence. Real data tells you where the mass actually is, synthetic data fills in the gaps around it.



Corn

So Daniel's plan is sound, but the version where he generates everything and never labels anything is the version that fails silently.



Herman

Right. Generate the bulk, but hand-label a few real items per box. That's the version that survives contact with the garage.



Corn

There's a cost to incremental retraining he hasn't priced in, though. And it's the reason "just retrain weekly" has a hidden bill even when the compute is free.



Herman

Catastrophic forgetting. If he retrains only on synthetic data for the three new boxes, the model can degrade on the old ones. The weights get overwritten by whatever they saw last. The standard mitigations are experience replay, keeping some old examples in the mix, or regularization approaches like Elastic Weight Consolidation that penalize the model for moving weights that mattered for previous classes.



Corn

For a home system, replay is easy. He's got the old rows sitting in the database.



Herman

He does, and that's actually the cleanest answer at his scale. Don't do incremental at all. Do a full retrain on everything, old and new, every time a box changes. At a few hundred classes and thirty seconds of training, the full retrain is cheaper than the engineering cost of doing incremental safely.



Corn

Which is a useful thing to say out loud, because "incremental versus full" dominates a lot of production machine learning as if it's a real fork in the road. At this scale it isn't a fork. It's a rounding error.



Herman

The decision only becomes real when a full retrain takes hours or days. Daniel's takes less than the time to walk to the garage.



Corn

So the honest answer to his first question is yes. Weekly retraining is practical. A cron job is fine. And if he wants to be careful, he retrains on everything rather than just the new boxes.



Herman

That's Pattern A, fully validated. And it still leaves the question of whether he should be doing it at all.



Corn

Which is where the freezing approach comes in. Explain the move.



Herman

The move is to stop treating the label set as something that lives in the model. Freeze the encoder. Take every box description, run it through the encoder once, and store the resulting vectors in an index. When an item comes in, encode the item name, find the nearest box vector, done.



Corn

So the model never changes. Adding a box is adding a row.



Herman

Adding a box is a database insert. No training run, no synthetic data pipeline, no forgetting, no retraining schedule. The thing that changes is data, not weights.



Corn

There's an open-source implementation of exactly this.



Herman

The Adaptive Classifier, published on Hugging Face last year. New classes can be added seamlessly without retraining existing knowledge. It keeps a prototype memory, which is a representative vector per class, and does similarity search over it, with a light neural layer on top. The prototypes update on an exponentially weighted moving average, and the index only gets rebuilt once accumulated updates cross a threshold. Defaults are a thousand examples per class and a prototype update every hundred examples.



Corn

Interesting that it still carries Elastic Weight Consolidation in reserve.



Herman

It does, and that's worth noticing. Even the architecture built specifically to avoid retraining keeps a forgetting mitigation for when new classes arrive. The difference is that it's a safety net rather than the primary mechanism. The primary mechanism is that the class was never in the weights to begin with.



Corn

What's the research lineage behind it?



Herman

The ACL paper from 2019 is the conceptual ancestor, replacing the fixed output layer with a metric space. There's a more recent method, KLDA, that freezes the foundation model entirely. When a new task arrives, it computes the mean for each class and updates a shared covariance matrix. No replay data at all, and accuracy comparable to training on everything jointly. Adding a box means computing one new class mean.



Corn

One vector.



Herman

One vector and a covariance update.



Corn

And the retrieval-augmented version, for completeness?



Herman

Retrieval-augmented classification uses relevant stored memories to guide predictions and allows behavior updates in real time without retraining. In their evaluation, updating the memories recovered five percent accuracy and came within half a percent of a fully retrained traditional classifier. So the gap between "retrained" and "never retrained" is half a point.



Corn

Half a point is the kind of number that ends an argument.



Herman

It should. Now, the cron job question. Daniel asked whether this can live in a back-end cron job.



Corn

And the answer is yes, but the interesting part isn't the answer.



Herman

The interesting part is that cron-based retraining is the standard starting point, and the guidance on it is blunt. Scheduled retraining fires on a cron schedule regardless of observed drift. It's the safest starting point because it's predictable and easy to test. That's the argument for it. The argument against it is in the next sentence. It can retrain unnecessarily when data is stable, or miss a sudden drift event between scheduled runs.



Corn

For Daniel, "retrain unnecessarily" is a two-and-a-half-cent problem. He doesn't care.



Herman

At his scale the scheduled-versus-triggered question is almost philosophical. But it's worth laying out the trigger taxonomy because it's what production systems actually use. Scheduled, which is the clock. New-data volume threshold, which fires when enough new labeled records accumulate. Performance degradation, which fires when the model's measured accuracy drops. Distribution shift, which fires when the inputs start looking different from what the model was trained on.



Corn

Give the thresholds.



Herman

A common drift alert is a population stability index above zero point two on a key feature. A common volume trigger is ten thousand new labeled records.



Corn

Ten thousand new labeled records. Daniel buys three boxes a year.



Herman

Daniel is roughly four orders of magnitude away from needing a volume trigger. Which is the point. The trigger machinery exists for systems where retraining costs real money. At his scale, you could retrain on every single write and not notice.



Corn

So the real payload here isn't the infrastructure. It's the part about governance.



Herman

This is the line I'd put on a wall. Most teams treat continuous training as a tooling problem. The hard part is governance. In practice, automated pipelines tend to fail not because the infrastructure is wrong but because nobody defined the promotion criteria, nobody owns the monitoring alerts, and nobody has a clear mandate to roll back a bad model under time pressure.



Corn

For a solo home system, "governance" is one person and a garage.



Herman

It is, and that's the easier case. But the failure pattern still apply, just smaller. A cron job that silently fails for three weeks while Daniel keeps trusting the routing suggestion. Synthetic training data that drifts away from the real distribution because the generator was told to produce example items and it produced plausible-sounding ones that don't resemble anything he owns. A feedback loop where the model's own predictions get logged as training data and it gradually confirms its own mistakes.



Corn

That last one is the one I'd worry about. If the system records "I put the thing in box 100 because the model said so" as a training example, the model is teaching itself.



Herman

Which is why the standard advice is a holdout set and a promotion gate. There's a simpler version of it that fits a home system perfectly well. If three consecutive retrains fail an evaluation against a fixed set of items you know the right answers for, stop the automation and go look at it.



Corn

Three consecutive failures and the pipeline halts.



Herman

That's a circuit breaker. It sounds like overkill for a garage. It costs one table of known item-box pairs.



Corn

Let me steelman the other side for a second, because I think there's a version of this where the circuit breaker is worse than nothing. If Daniel's system halts and he doesn't notice for a month, he's been routing items by hand the whole time and he's fine. The circuit breaker only helps if he's checking the alert.



Herman

That's fair. The circuit breaker is only as good as the notification. But the alternative, a pipeline that fails silently and keeps serving stale predictions, is worse, because he trusts it. The failure pattern you don't know about is the one that costs you. A halted pipeline is loud. A quietly degrading model is quiet.



Corn

So the rule is that the automation has to fail visibly.



Herman

The automation has to fail visibly, and the evaluation set has to be small enough that he actually maintains it. Ten items. One row each. If a retrain can't route ten items he's confident about, that's the signal.



Corn

So back to Daniel's actual architectural question. Production systems with a bounded but frequently changing label set. What's the honest answer?



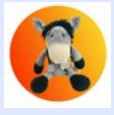
Herman

The honest answer is that the mature pattern separates the encoder from the label store, so label churn never touches model weights. Retraining pipelines still exist and they're cheap, but the systems that handle frequent label change gracefully are the ones where adding a label is a data operation rather than a training operation.



Corn

And the research has moved toward that position over the last couple of years.



Herman

It's converged. There's a paper arguing that continual learning has closed the gap, that with foundation models it's now possible to reach upper-bound accuracy and that the field is ready for real applications. And there's work on dynamic label hierarchies from ICML this year, where the taxonomy evolves across granularities, which is about as direct an analogue to boxes being added and split as you could ask for.



Corn

A box that splits into two boxes is a label hierarchy changing granularity.



Herman

A box splits into two, or two boxes get consolidated into one, and the labels move up and down a level. That's exactly the problem that paper is about.



Corn

So Daniel has two answers and they're both correct. Retrain everything on every change, which costs seconds, or don't retrain at all and put the boxes in an index.



Herman

And the second one is the one I'd build. The first one is the one I'd build if I already had the app and didn't want to change the architecture.



Corn

Because they don't conflict. He could retrain weekly and it would work fine. He could also never retrain and it would work fine.



Herman

The failure pattern is the middle. Retraining only on the new data, on a schedule, with no holdout and no evaluation, which is the version most people build first.



Corn

Which is the version where he buys three boxes, the model gets three new classes, and quietly gets worse at the four hundred old ones.



Herman

And he finds out when the monitor goes in the box with the plug adapters.



Hilbert

Forty-two dollars.



Corn

...Go on.



Hilbert

A case of them. Forty-two dollars and change, but the invoice said forty-two ninety-five, and I only counted the marker budget, which was separate. This is the eighties. Late eighties. I was at a returns warehouse outside of Newark, and the job was routing. Everything that came back had to go to a bin, and there were a few hundred bins. Electronics in one run, small appliances in another, and the bins kept moving. A line would take off and they'd split a bin into three.



Corn

So the bins were the labels.



Hilbert

The bins were the labels, and the bins were alive. You'd learn the layout on Monday and by Thursday there were two new bins where one used to be, and the old bin number meant something different. The first month I was there, I tried to keep a written map. Little index card, updated every morning. By week three the card was fiction.



Herman

How did you keep up?



Hilbert

You didn't keep up with the map. You kept up with the things. You learned what a thing was and what it was for, and then you asked where that kind of thing went today. The bin was a fact about the warehouse that week. The thing was a fact about the thing.



Corn

That's the whole architecture, isn't it. The thing is stable. The bin is data.



Hilbert

That's why I'm telling you. The forty-two dollars was the marker budget, because every time the bins moved, someone had to relabel them, and I was the someone. I spent forty-two dollars of my own money on markers and labels before anyone told me there was a reimbursement form.



Herman

The form was probably in a bin that had moved.



Hilbert

The form was in a bin that had moved twice. I found it eventually. But here's the part that matters for your Daniel. The warehouse never retrained me. They never sat me down and re-taught me the routing. They moved the bins and I adapted, because I'd learned the things and not the map. If I'd memorized bin numbers, I'd have been useless by week four.



Corn

The humans were running Pattern B the whole time.



Hilbert

The humans were running Pattern B because Pattern A would have meant retraining every picker every time a line took off. Nobody had the budget for that. So they moved the labels and left the people alone.



Herman

The failure pattern was the new guy who memorized the map.



Hilbert

The new guy who memorized the map sent a pallet of returned blenders to the bin that used to be small appliances and was now something else entirely. Nobody died. But the pallet sat there for two weeks because nobody was looking for blenders in that bin.



Corn

That's the monitor in the plug adapter box.



Hilbert

The model isn't wrong about blenders. It's wrong about where blenders live this month.

SOURCES

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

1. **Hugging Face SetFit repo (commit d70dad8), README + docs [primary]**
   https://github.com/huggingface/setfit
2. **HF SetFit zero-shot tutorial (synthetic data) [primary]**
   https://huggingface.co/docs/setfit/main/tutorials/zero_shot
3. **Adaptive Classifier: Dynamic Text Classification with Continuous Learning, Asankhaya Sharma, 2025-06-20 [primary]**
   https://huggingface.co/blog/codelion/adaptive-classifier
4. **SetFit: Efficient Few-Shot Learning Without Prompts (30s/$0.025 training figure)**
   https://huggingface.co/blog/setfit
5. **HF SetFit ONNX tutorial (CPU training time)**
   https://huggingface.co/docs/setfit/main/en/tutorials/onnx
6. **MLflow, Continuous Training in ML: A Practical MLOps Guide, 2026-08-04**
   https://mlflow.org/articles/what-is-continuous-training-ml/
7. **Snowflake continuous-training triggers**
   https://www.snowflake.com/en/artificial-intelligence/machine-learning/mlops/continuous-training/
8. **Metric Learning for Dynamic Text Classification, ACL 2019**
   https://aclanthology.org/D19-6116.pdf
9. **Orca RAC blog**
   https://orcadb.ai/blog/retrieval-augmented-text-classifiers-adapt-to-changing-conditions-in-real-time

10. **Online Continual Learning with Dynamic Label Hierarchies, ICML 2026**
https://arxiv.org/abs/2605.11742

11. **Task-Oriented Rank Adaptation for Continual Learning in Text Classification, 2026-10-01**
https://arxiv.org/abs/2610.01702

12. **Momeni & Liu, Achieving Upper Bound Accuracy of Joint Training in Continual Learning**
https://arxiv.org/abs/2502.12388

13. **Continual Learning Using a Kernel-Based Method Over Foundation Models (KLDA)**
https://arxiv.org/abs/2412.15571

14. **Resilio Tech on cron anti-patterns**
https://resiliotech.com/blog/replacing-cron-jobs-with-proper-ml-pipeline-orchestration

15. **ZenML on Twitter Cortex retraining history**
https://www.zenml.io/blog/the-hidden-complexity-of-ml-pipeline-schedules

16. **AWS Prescriptive Guidance**
https://docs.aws.amazon.com/pdfs/prescriptive-guidance/latest/creating-production-ready-ml-pipelines-on-aws/creating-production-ready-ml-pipelines-on-aws.pdf