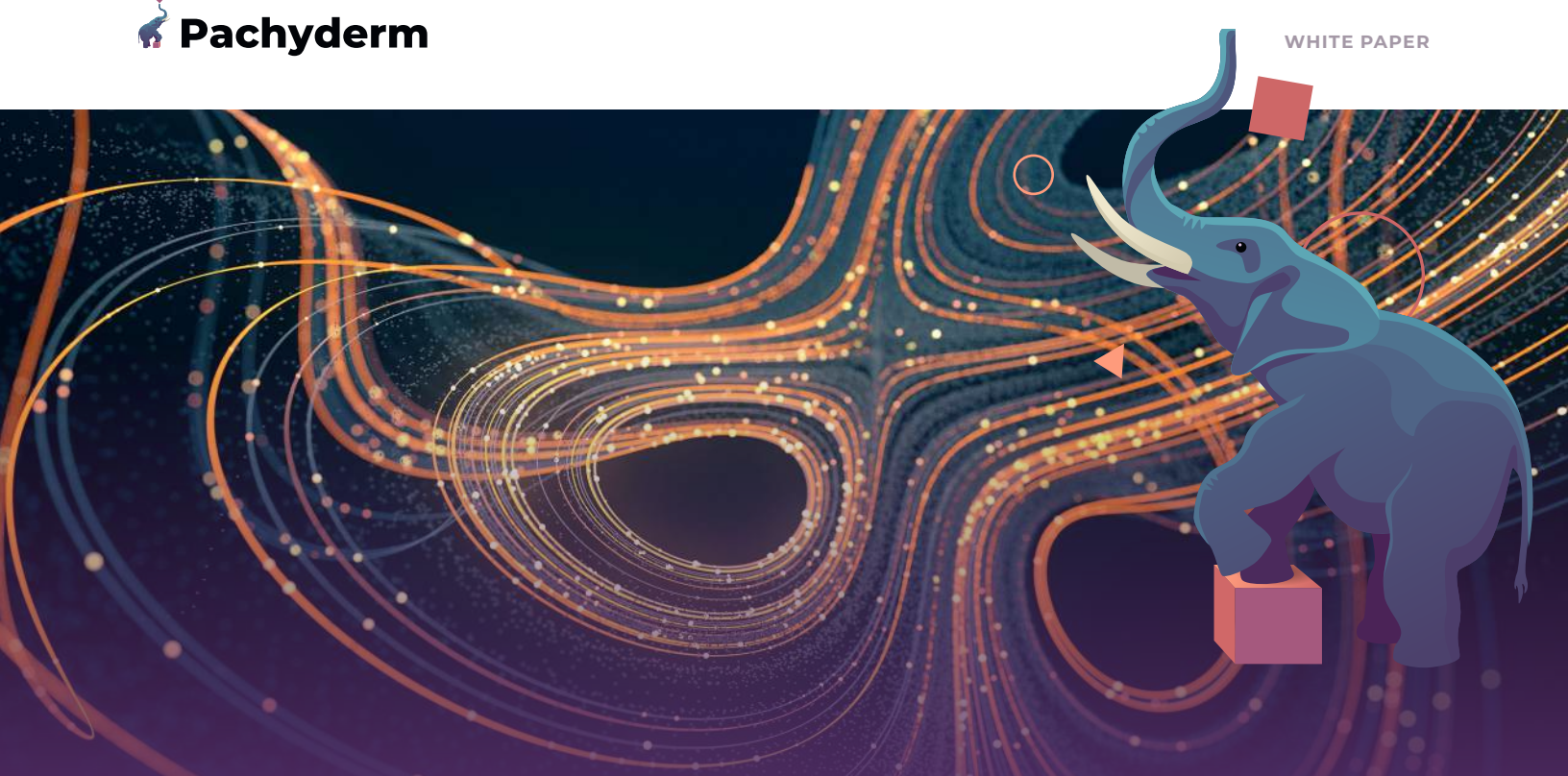
Pachyderm

WHITE PAPER



# Practical Data-Centric AI in the Real World

**Written by Dan Jeffries,**
Chief Technical Evangelist

## Table of Contents



## Introduction

"Data centric AI is the discipline of systematically engineering the data used to build an AI system," according to legendary AI researcher Andrew Ng.

In other words, it's focusing on updating the data to solve a problem versus changing the algorithm or code. That's a complete reversal of how we've thought about AI up until now.

Over the last decade, researchers focused on code and algorithms first and foremost. They'd import the data once and generally leave it fixed. If there were problems with noisy data or bad labels they'd usually work to overcome it in the code.

Because we spent so much time working on the algorithms, they're largely a solved problem for many use cases like image recognition or text translation. Swapping them out for a new algorithm often doesn't make much difference. You may get marginally better performance with a transformer versus a convolutional net but not enough to really move the needle of accuracy. Even tuning parameters probably won't help all that much. Are you really going to stumble on a better architecture than Fast-R-CNN or YOLO for object detection by pushing and pulling random levers in your hyperparameters if your team doesn't have a huge number of dedicated AI researchers?  Almost certainly not.

Data centric AI flips that on its head and says we should go back and fix the data itself: clean up the noise. Augment the data set to deal with it. Re-label so it's more consistent.

## Defining Data-Centric AI

Data centric AI is one of those intuitive concepts that just makes sense once you hear it. But what does it mean in the real world? How do you put it into practice? How do you focus back on data in a practical and actionable way?

There are six essential ingredients to putting data-centric AI into practice in your organization:

- Creative thinking
- Synthetic data
- Data augmentation
- Tooling
- Testing
- Clarifying instructions

| What are the differences between Model-Centric and Data-Centric ML? | | |
|---|---|---|
| **Model-centric ML** | VS | **Data-centric ML** |
| Working on code is the central objective | • | Working on data is the central objective |
| Optimizing the model so it can deal with the noise in the data | • | Rather than gathering more data, more investment is being made in data quality tools to work on noisy data |
| Inconsistent data labels | • | Data consistency is key |
| Data is fixed after standard preprocessing | • | Code/algorithms are fixed |
| Model is improved iteratively | • | Iterated the data quality |

***Source:*** *A Chat with Andrew: From Model Centric to Data-Centric AI*

Notice I didn't call them steps. That's deliberate, because I don't want you thinking about these as a series of steps you do in some kind of order. Instead, think of them as a set of ingredients you can pull from to make lots of different solutions, just as you can make lots of different kinds of food from the same set of ingredients. All of these approaches to solving data-centric problems are interlinked and you're likely to use a number of them at the same time to solve each challenge.

But there is one ingredient you'll need on every data-centric AI solution:

**Your mind.**

## Solving Data-Centered Problems Means Changing Your Perspective

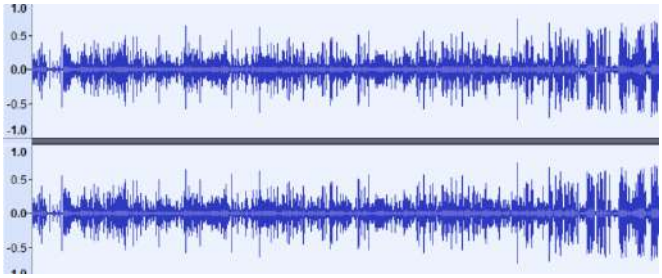
No matter what, you're going to have to think about problems differently.

Each problem will require a unique creative solution. It will also involve **a different set of skills than typical data science skills**. Depending on the format of your data, you may need an IT specialist, an audio engineer, a programmer, or a graphic designer on your team to solve the problem.

### Example: Data-Centric Speech Recognition

#### The Problem

Imagine you're developing a speech recognition model to detect commands in a car. You discover the model performs well in most scenarios, but really struggles with background noise, and your customers expect their voice commands to work even with traffic, music, children, or weather noise in the background - after all, those are normal conditions encountered while driving.




***Source:*** *Audacity by Author*

**How do you fix it?**

Data centric AI demands creative problem solving and thinking through the solution from start to finish. We can take different approaches to solving our problem, but we always start with the data.

You pull as many noisy samples from the data as possible and listen to them. What kinds of background noise are you hearing? Kids screaming? Wind noise from an open window? A loud radio?

Now you have something to go on. But what's a data-centric scientist to do? Does that mean you have to go out and record a few thousand hours of noisy samples? It's a daunting and time consuming proposition... It's enough to make anyone go right back to trying to find a better algorithm.

**Solution: Feed the model what it needs to filter**

A data-centric approach would be to collect samples of background noise from audio libraries that match the types of background noise your customers will encounter on the road: Wind noise. Rain. Thunder. Kids playing, talking, and shouting "are we there yet"!? Music samples played at different levels.

Now, an audio engineer can add these to your existing samples and output a set of new examples.

To scale it up, you have the programming team write some code to add random noise samples to your existing examples, creating lots of permutations of the original dataset.

Now you have the same samples but you've added a bunch of background noise, generating new samples to train from and that may help the algorithm learn how to best deal with background noise.

You might also take existing audio and run it through distortion effects. Make it sound more metallic or grainy with filters. Add little skips and pauses into the data. Lower the sound and raise it in a wave pattern through the file. Now you've got some parts of the sound effects where the voice is suddenly lower and then higher. All of this may help your model learn lots of different representations and edge cases where people's voices trail off, stop short, or get softer in response to the environment.

And just like that - you've worked with two key ingredients in our list:

- **Synthetic data**
- **Data augmentation**

## Reality Augmented

In the voice command example, the first solution presented is **synthetic data**. That's where you've crafted wholly new samples by adding wind and storms and musical samples to the background of your files. They're not naturally recorded samples. They're hand-crafted and/or created programmatically.

The second technique is **data augmentation**. That's more straightforward: it involves running data through filters or altering it slightly, not building new versions of data that don't exist. For instance, if you were building an object detection model, you may just run images through a simple program that distorts the image slightly or flips an image. In the car image below, we simply flipped the car horizontally to create more data permutations.

You may start to worry that you haven't perfectly recreated the background noise of a car with either of these techniques. But to the model, it may be good enough to make it generalize to the real world. **Synthetic and augmented data doesn't have to be perfect in every case. It just has to give the model the chance to learn more about edge cases**.

When research houses like OpenAI and DeepMind work with gigantic data sets, what are they really looking for in those massive troves of data? Edge cases. The bigger the dataset, the more chances that you have lots of strange edge cases - so the model has a better chance of learning them.

But you don't need a massive data set when you're building something very specific, such as our model to detect voice commands in a car setting. Unlike the huge general purpose foundational models like GPT-3 that try to understand lots of different types of speech in many different contexts, our model is looking to understand voice commands in a very narrow and specific context.

### Managing Augmented Data Storage Needs

That said, if you're paying close attention you might realize that you're increasing the amount of data with each of these techniques. That might not be a problem if you're creating more text files, but it can be a major factor if you're creating more unstructured data like audio, images and video. Make sure you've got the right amount of storage space to deal with the growth as it can multiply quickly.

If you're flipping all your images, like in our car picture above, your data grows by a factor of two. If you're adding lots of different filters on sounds they can grow by a factor of five, ten or more.

There are a few things you can do to control that growth.

The first option is **online augmentation**, or augmentation on-the-fly. It may seem obvious that you need to generate all your data in batches before you train the model, but that's not always the case. In some simpler cases of data augmentation, like flipping images, you can perform transformation in mini-batches with GPUs and throw that data away.

But that won't work with complex transformations like adding background noise to files. The load of transforming the data would add orders of magnitude to your training time as you waited for programs to finish combining audio files and outputting new ones, even if the programs worked in parallel to your training. In that case you're going to need to do the data transformations in batches ahead of time.

That's where another one of our ingredients comes into play: **The right tooling**.

## The Right Tool for the Right Job

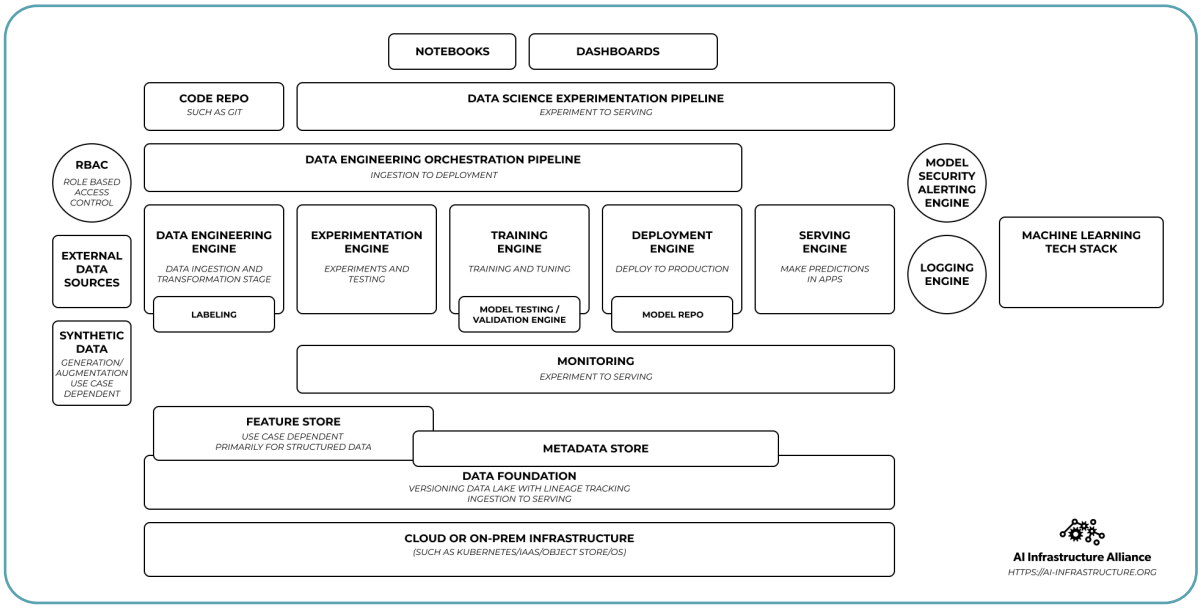
The right tooling is essential to a data-centric AI approach.

### Unfortunately, most MLOps platforms, especially the all-in-one solutions, are model-centric.

As data-centric AI moves to the fore, we're starting to see more data versioning and lineage systems on the market. The default of most engines is to simply make complete copies of the data when you add a new version. That might work for smaller files, like text files, but it's a disaster if you're building lots of new audio or video files. You could quickly see exponential growth in your dataset and send your cloud storage soaring if you're not careful.

You'll want a data engine that's capable of **deduplication** in the file system. This keeps copies of only the changes and differences between files rather than complete copies of each file. Pachyderm's built in copy-on-write style file system does that for you, deduplicating the data to keep it small.

Pachyderm's key capabilities for data-centric AI go well beyond just deduplication. At its heart, Pachyderm is a data orchestration engine. If you take a look at the AI Infrastructure Alliance's enterprise stack blueprint, you'll see that the diagram divides up the stack into data engineering orchestration pipelines and data science experimentation pipelines.




Source: *AI Infrastructure Alliance*

Packaged, "all-in-one" MLOps software assumes that all orchestration pipelines are the same, and that you need the same style of pipeline for data science and data engineering - but this is simply not the case.

The majority of these tools are focused strictly on the data science experimentation pipelines. That's where data scientists try different algorithms, tune hyperparameters and test their theories about how to output a good model that generalizes well to the real world. Experimentation pipelines assume the data is mostly fixed - meaning, experimentation pipelines are model centric.

## Data engineering pipelines are data-centric.

They're focused on wrangling data: cleaning it, transforming it, and augmenting it.

Data-centric pipelines let you pull in the data, transform it, load and change it and export it. Pachyderm is definitively data engineering focused (though its experimentation chops are growing with support for things like Notebooks that sport automatic mounting of Pachyderm repos right in the Notebook.)

Another key difference of data-centric pipelines is **language agnosticism**. While most experimentation pipelines are Python based because so much data science work is done in Python, that won't work for data engineering. You may need any number of third party libraries for those transformation steps. For instance, you might need to write something in C or Java or call a specialized audio library if we go back to our car audio command detection use case.

Pachyderm is 100% language agnostic. You can run R in part of a pipeline, Python in another part, C in another and Java in a fourth step. That's essential for working with these kinds of data transformations at scale.

Your platform also needs to allow for **robust version control and lineage**. That's the final piece of an excellent data-centric AI tool. It's got to keep track of all the changes automatically. You're going to be iterating on the dataset a lot and then running experiments to see if it worked. You need to be able to roll back to any previous version of the data or code and then roll forward again.

You'll want to keep snapshots of all the data you use, and you'll want immutable copies of that data. If you happen to overwrite some of your data or you shuffle it around and forget which copy had the right versions you needed to build your model that creates a mess for reproducibility. Any data versioning system that doesn't keep immutable copies of every version of your data is only half baked. Immutable means that an older copy of the data can't change. You can't overwrite it or modify it. Instead, **the system automatically makes a new, deduplicated copy and the lineage system keeps track of the changes** and the differences between those snapshots for you.

Without immutable data lineage, you can easily get into a situation where your metadata storage points to a state of the data that no longer exists.

Take our audio command capture example: Imagine that you've added static and wind noise to the background of your samples but it doesn't seem to be making a difference in accuracy.

So you start iterating:

- You raise the sound of the voice and lower it
- You find new audio samples and add those to the background
- You include a wider range of weather noise
- You include samples that are speaking faster, and more slowly

At some point you hit on the right set of synthetic data and your model starts performing a lot better with real world test data that has intense background noise.

But you think you can do even better, so you do some more transformations. The new transformations don't help. In fact, they make accuracy worse. You need to go back to the version of your data that produced the best result... but it's gone. You overwrote the files by accident with the new test transformations. You can't recreate your best results without a lot of engineering work.

That brings us to the last couple of essential data-centric AI ingredients.

## Test, Test and Test Again

If you build combine strong test with version control in your data iterations, that can make your life a lot easier. If accuracy drops with new synthetic data, you simply roll back to an old snapshot, try some different transformations and then test again. Much of that work can be automated to run much more smoothly.

Even better, you'll be ready to create human in the loop tests at each stage of your development process.

### Data-Centric AI: Better With Humans

Let's take a different use case here. You want to teach a model to detect defects in the solar panels your company manufactures. You've got a lot of pictures of good solar panels and a bunch of broken ones with cracks, splits, and scars fresh off the assembly line.

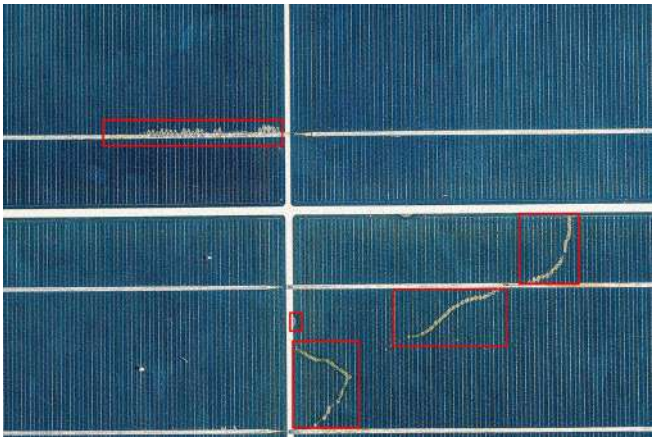
You don't have a labeling team in house so you use a third party software and people. We've seen companies like Scale AI and Snorkle AI post big valuations in recent days because they have tools to help speed up the tedious and time consuming process of labeling all that data accurately.

Perhaps one of those platforms gives you an automatic instance segment detector and it applies bounding boxes around what it thinks are the defective segments of the solar panel.

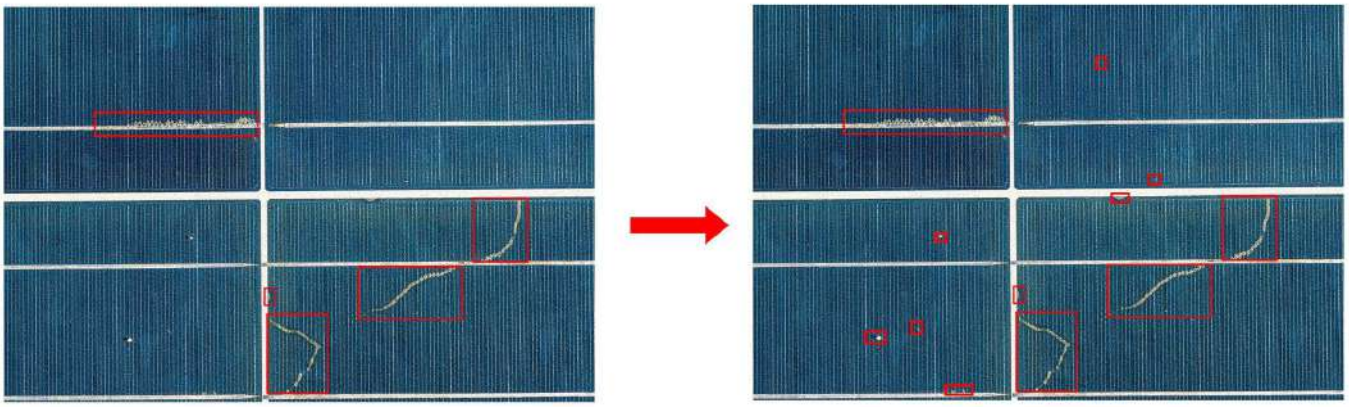
Do you just hand those unlabeled images off to the magical segment detector and hope that it comes back correct?

Of course not. It can and will miss things you want, or just highlight irrelevant segments of the image. Or it may be missing whole things that you consider defects, perhaps smaller defects and you need to go back and get it to highlight those missing spots.

No technique that speeds up labeling is perfect, and you still need a team of smart people to **do** that labeling even if the software speeds up **how fast** you can do it.



*Automatic instance detector result*



*Automated result (left) vs. Human-labeled (right)*

Even after you've confirmed that the segment detector is doing a good job putting bounding boxes around the defects, you craft a human-in-the-loop test. You randomly pull images that are labeled and you inspect them visually to see if the bounding boxes make sense or if they're off base. If the labels are off, you've got to go back and correct them. That may require you to do another test and another, as you go through iterations of labeling and inspecting the results.

## Sentiment Analysis with Humans in the Loop

Let's imagine another scenario. You've got a dataset of customer feedback and support cases. You want to train a system to detect whether the cases are urgent and if the customer is calm or angry. That will help you triage cases faster, by automatically sorting the most important cases to the top of the pile so your support team can fix the most important problems faster.


*Source: MonkeyLearn*

Perhaps the labeling software has a built-in sentiment analysis system and they give you back the data fully labeled as "angry," "very angry," "calm," "neutral," "happy," etc. They also give you a score of 1-5 with 5 being the most important based on the severity rating the cases had.

Again, do you simply trust the labels? No. You verify them with spot inspections and by having skilled domain experts review the spot inspections.

Perhaps your best support people go through a series of non-severe cases and realize many were urgent, but never got escalated. The support personnel were under load and decided to keep the rating on the severe cases down artificially, because more cases would have meant more stress and a longer wait times.

Identifying the issues with your initial sentiment analysis and how to accurately identify cases in need of escalation isn't an engineering problem - a domain expert must weigh the cases and see which information indicates that these cases are severe.

You also have another team go through and categorize the sentiment of the cases on their own and then you can compare them to see how they match with the automatically labeled dataset. Is there a big delta between the automatic label and what the human reviewers think?

That takes us to our last ingredient.

## Clarifying Instructions for Humans in the Loop

It's easy to misinterpret instructions. A seemingly easy task can lead to a lot of different interpretations when different people do that same task. Andrew Ng uses a simple example to illustrate. Imagine someone is tasked with putting bounding boxes around animals in pictures.

As you can see, one person puts the bounding box around each animal individually. The other puts one big box around both animals. A third person overlaps the boxes.

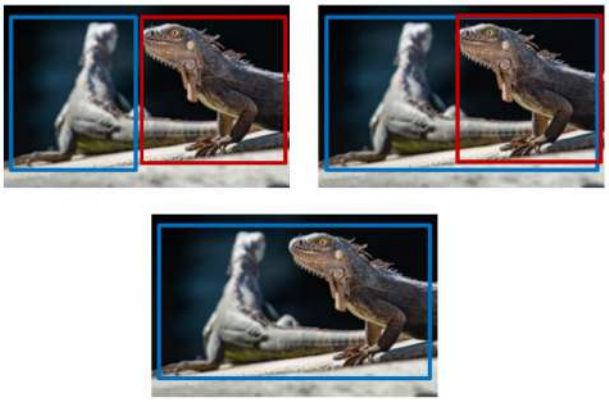
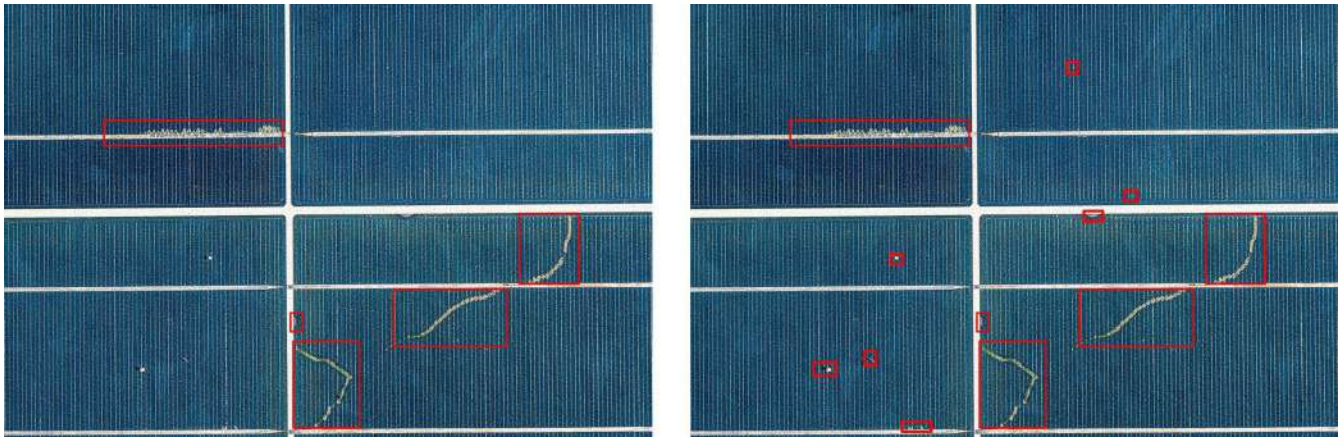


*Photo courtesy of Andrew Ng's Data-Centric AI presentation found here*

At this point your first response might be "Why didn't they just do what I asked? They must not get it!" Instead, this is a signal to rethink your approach and ask, **"How can I clarify my instructions to better explain what we need?"** You may have to go through many rounds of clarifying the rules and instructions before your labelers provide exactly the results you're looking to get.

Let's go back to our solar panel defects use case. You notice that some of the labelers are missing cracks of a certain size. Some are labeling anything that even looks like a minor scratch as a defect.



***Source:*** *Author via images from Shutterstock*

At this point, you have to think clearly about what you know about your domain, or tap an expert to judge your results so you can better define your labeling. You'll want to clarify factors like:

- Are minor scratches a defect, or will the solar panels still do the job with those scratches and therefore you don't want them labeled as defects?
- Is a divot in the panel a real defect or can the section that's dented still function?
- Are these performance defects, or cosmetic ones?
- Should you have unique labels for cosmetic defects vs performance?

Maybe those defects don't really affect the performance of the unit, but customers will still perceive them as broken, resulting in a higher rate of customer returns. In that case, you do want those minor defects labeled, and you need to adjust your instructions accordingly.

Then again, maybe there is an acceptable risk level with minor defects. You have the team classify them as minor or major defects. This distinction could even change how your model is used in production. Perhaps major defects are automatically pulled off the line and scrapped, but minor defects are sent to an inspection team to judge whether the product is discarded, or safe to ship.

This scenario brings us full circle: versioning your data as you go through many iterations of labeling, clarifying, and labeling again. Labeled data is good but versioned, labeled data is even better.

By combining human-in-the-loop spot testing, clarifying instructions, re-labeling problem data, and versioning the different iterations, you can efficiently build a smooth, data-centric pipeline that drives major improvements in your model accuracy and how well it generalizes to real world conditions.

## Focus Back on What Matters Now

For the longest time we've thought of ML code as the 'cool' part of data science. And in many ways, it is. Some of the biggest breakthroughs in the last decade, like Alphafold, came from fantastic algorithmic breakthroughs. But most of us aren't doing AI research and trying to crack the protein folding challenge that stumped researchers for decades.

Most business applications of AI are focused on more established use cases: churn prediction, computer vision, sentiment analysis, and business intelligence. We know what models will likely make the biggest difference, and we can start to treat most of the code as a solved problem.

For most teams **the algorithm is actually a very small component of the overall system, maybe 5% or less**.

### The Other 95% of Machine Learning

Collecting data, verifying it, transforming it, augmenting it, creating additional synthetic data and feature extraction are all much, much bigger factors in the success of a given machine learning process.

The lion's share of modern AI system complexity is bound to processing, monitoring, transformation, augmentation and handling all that data. These steps engage with multiple backend systems, traversing database and object stores, passing through multiple RBAC systems along the way, not to mention your MLOps tools themselves.

Without automated versioning and lineage, building workarounds and investigating bugs can lead to a lot of data-specific technical debt.

The key to data-centric AI is focusing back on where we spend most of our time anyway: Treating data as the core of the machine learning process - not as an afterthought.

Do that, and you're on your way to a powerful data-centric AI solution that delivers big results in the real world.

## Pachyderm Enables Data-Centric AI

Pachyderm is the leader in data versioning and pipelines for MLOps. Pachyderm provides the data foundation that allows data science teams to automate and scale their machine learning lifecycle while guaranteeing reproducibility.

### Try Pachyderm for Free

Get a free 30-day trial of Pachyderm's Enterprise features at **pachyderm.com/trial**

## Why Pachyderm?

#### Automated Data Versioning

Enable collaboration through data version commits, branches and rollbacks that eliminate data duplication. Automate a complete audit trail for all data and artifacts across pipeline stages stored as native objects, so that versioning is automatic and reliable.

#### Data-Driven Pipelines

Our Kubernetes native approach supports any library or language, and the datum-centric approach to pipelines enables parallel processing of large data sets. Data-driven pipelines execute when new data is committed without additional code.

#### Immutable Data Lineage

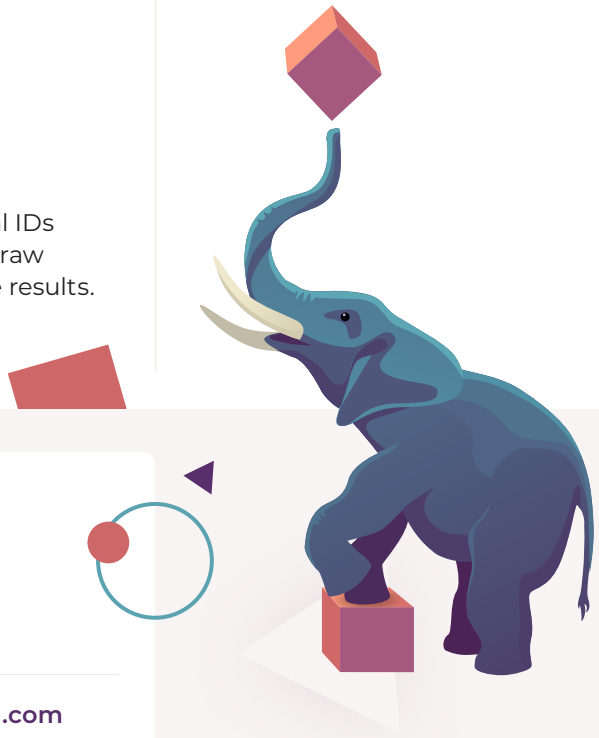
Track every version of your code, models, and data, maintain reproducibility of data and code for compliance, and manage relationships between historical data states. Pachyderm's Global IDs make it easy for teams to track any result all the way back to its raw input, including all analysis, parameters, code, and intermediate results.

### Improved Collaboration with MLOps Tools

**Pachyderm Console** provides an intuitive visualization of your DAG and aids in reproducibility

**JupyterLab Mount Extension** adds a point-and-click interface to Pachyderm versioned data

**Enterprise Administration** contains robust tools for administering Pachyderm at scale across your teams



## Contact Pachyderm

To learn more about Pachyderm's machine learning solutions, contact us:

info@pachyderm.com • 888-338-9597 • www.pachyderm.com