

# Applied Machine Learning

Rina BUOY

[Source](#)



AMERICAN UNIVERSITY  
OF PHNOM PENH

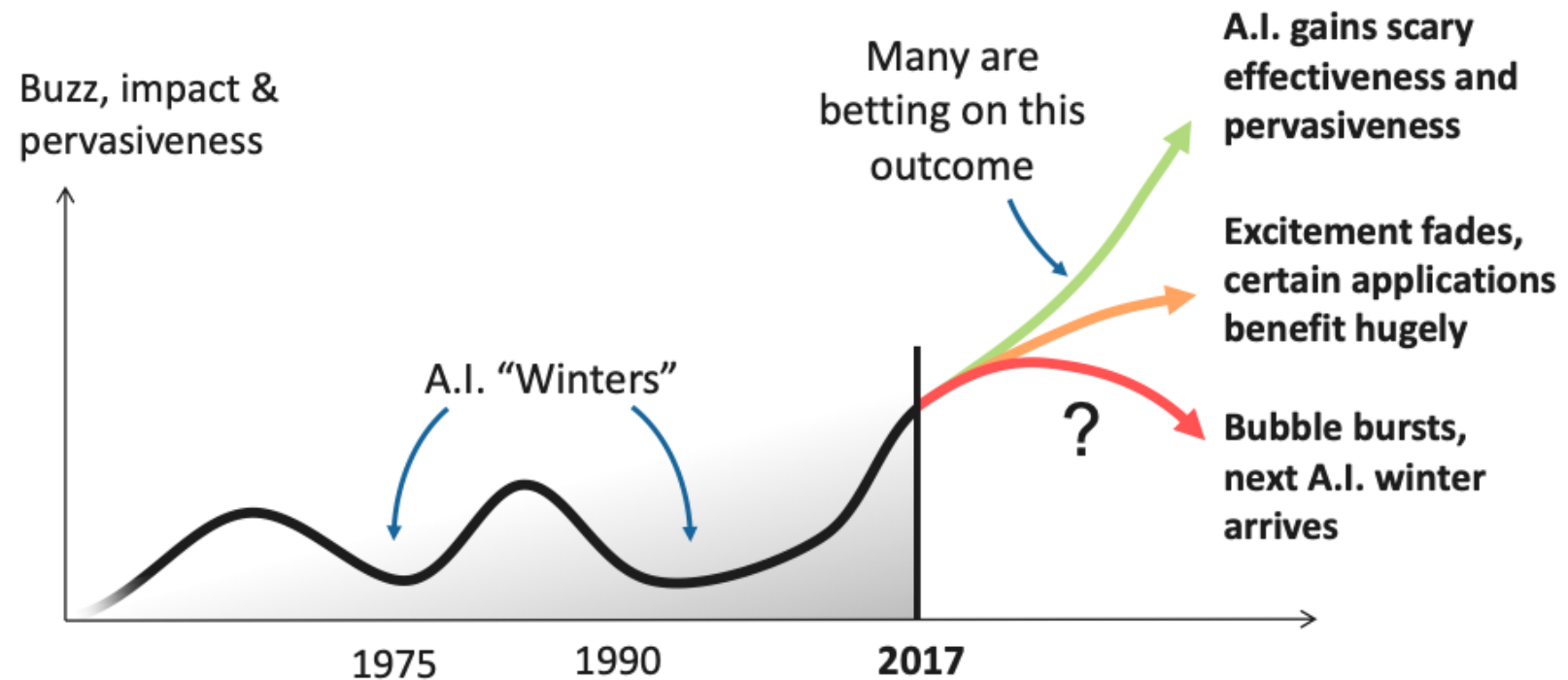
STUDY LOCALLY. LIVE GLOBALLY.

# When to use ML ?

# AI Progress

- The history of ML is characterized by stratospheric rises and meteoric falls of the public perception of the technology.

**AI is enjoying significant hype and investment**



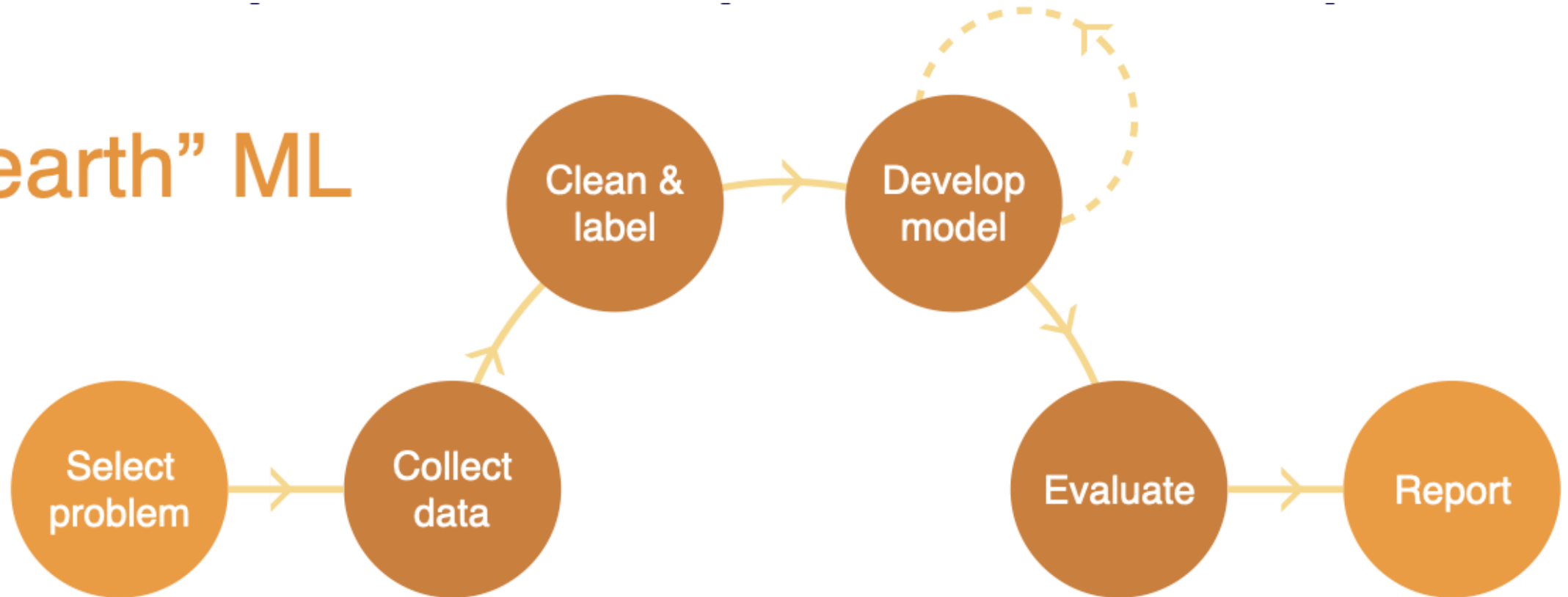
# Possible Outcomes

- Here are the major categories of possible outcomes and our guess about their likelihoods:
  - A true AI winter, where people become skeptical about AI as a technology. We think this is less likely.
  - A slightly more likely outcome is that the overall luster of the technology starts to wear off, but specific applications are getting a ton of value out of it.
  - The upside outcome for the field is that AI continues to accelerate rapidly and becomes pervasive and incredibly effective.
- Our conjecture is that: The way we, as a field, avoid an AI winter is by translating research progress into real-world products. That's how we avoid repeating what has happened in the past.

# ML-Powered Products

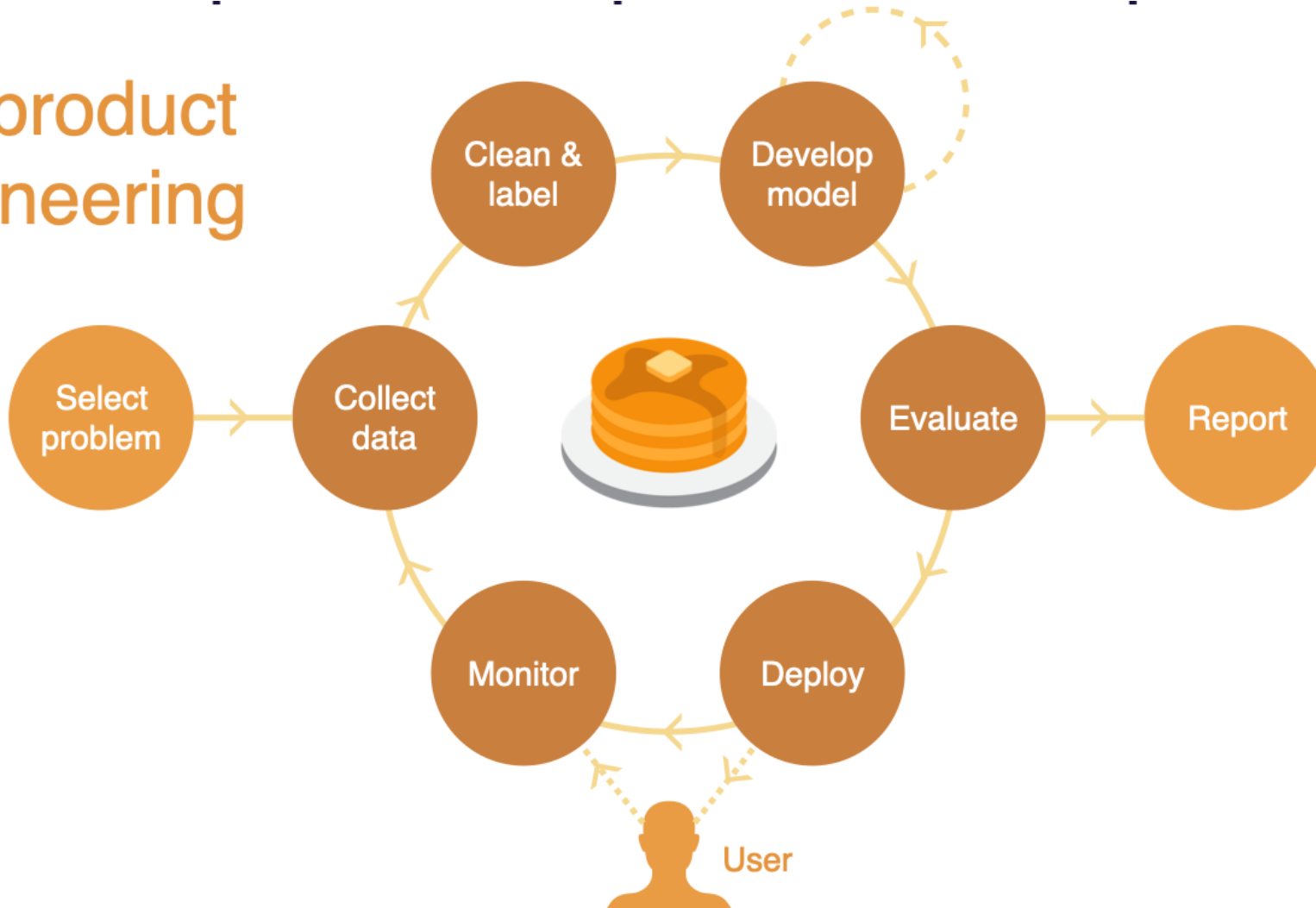
- Building ML-powered products requires a fundamentally different process in many ways than developing ML models in an academic setting.

## “Flat-earth” ML



# ML-Powered Products

## ML product engineering

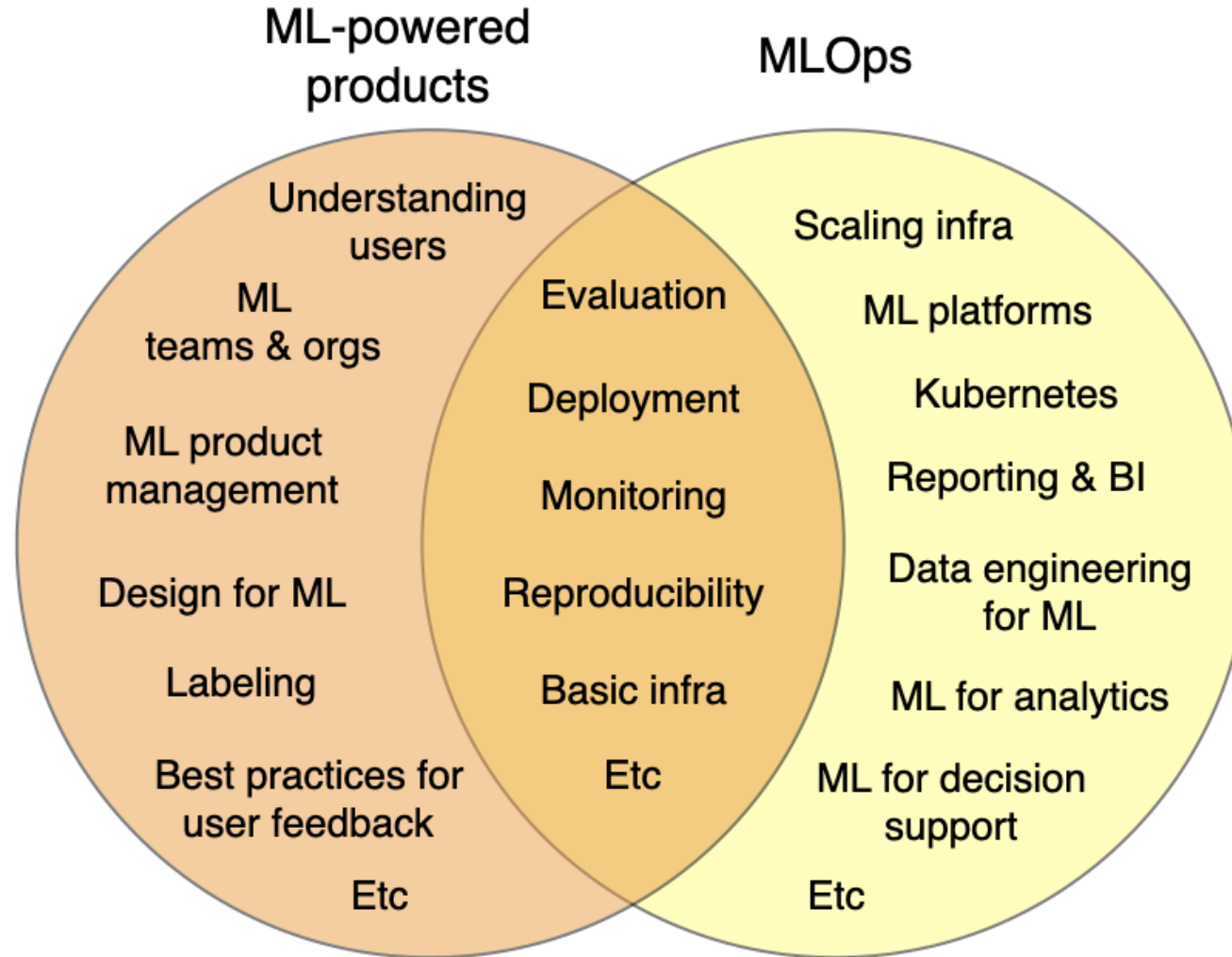


- ML-powered products require **an outer loop** where after you deploy the model into production, you measure how that model performs when it interacts with real users.
- Then, you use real-world data to improve your model, setting up a data flywheel that enables continual improvement.

# ML-Powered Products vs MLOps

- A lot of MLOps is about:
  - How do we put together an infrastructure that allows us to build models in a repeatable and governable way?
  - How can we run ML systems in a potentially high-scale production setting?
  - How can we collaborate on these systems as a team?
- ML product discipline includes:
  - How do you understand how your users are interacting with your model?
  - How do you build a team or an organization that can work together effectively on ML systems?
  - How do you do product management in the context of ML?
  - What are the best practices for designing products that use ML as part of them?

# ML-Powered Products vs MLOps





# When to Use ML At All

- ML projects have a higher failure rate than software projects in general. One reason that's worth acknowledging is that for many applications, ML is fundamentally still research.
- Therefore, we shouldn't aim for 100% success.

# Why ML Projects Fail?

- Additionally, many ML projects are doomed to fail even before they are undertaken due to a variety of reasons:
  - They are technically infeasible or poorly scoped.
  - They never make the leap to a production environment.
  - The broader organization is not all on the same page about what would be considered success criteria for them.
  - They solve the problem that you set out to solve but do not solve a big enough problem to be worth their complexity.

The bar for your ML projects should be that **their value must outweigh not just the cost of developing them but also the additional complexity that these ML systems introduce to your software**

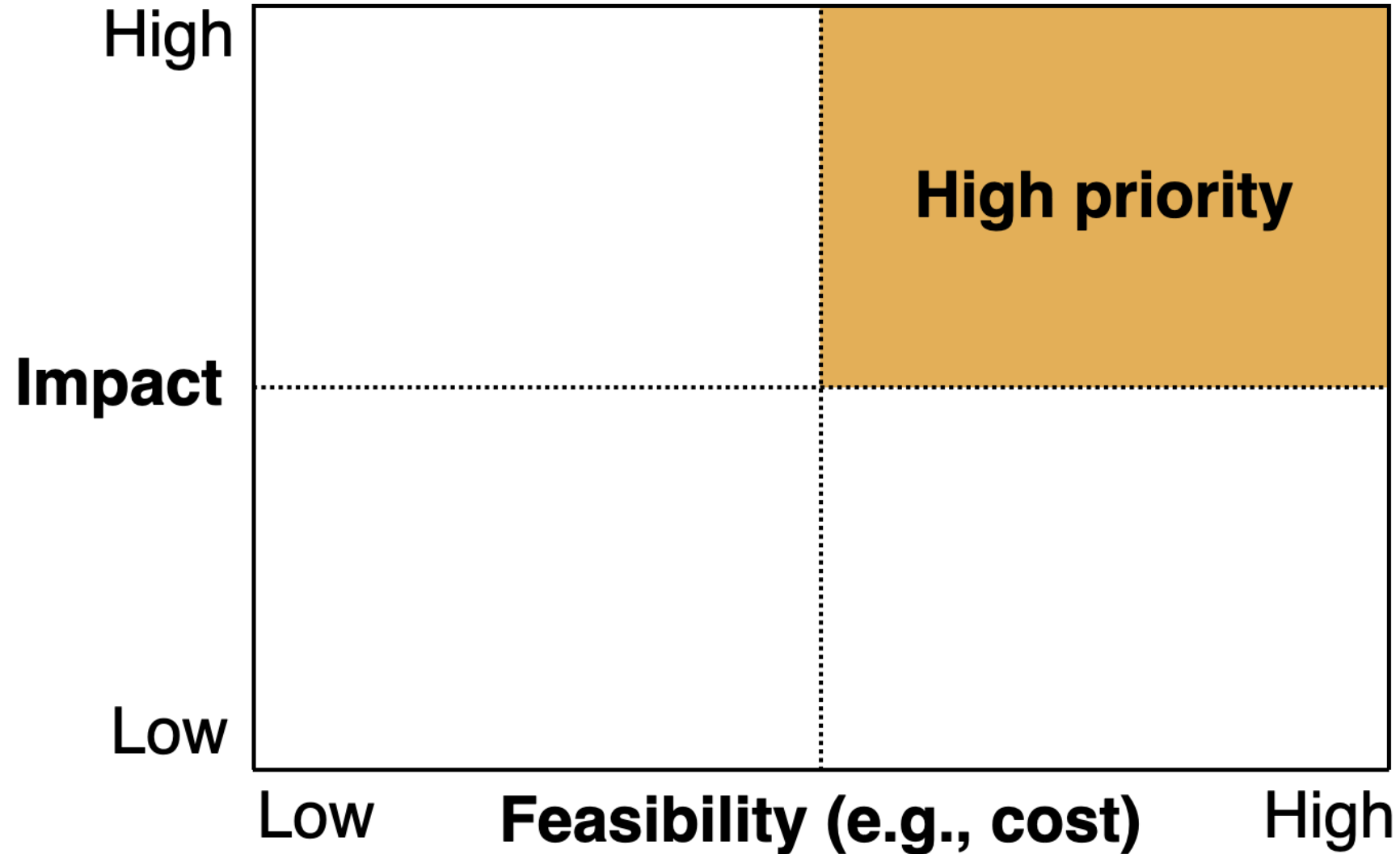
# When to Use ML ?

- Before starting an ML project, ask yourself:
  - Are you ready to use ML? More specifically, do you have a product? Are you collecting data and storing it in a sane way? Do you have the right people?
  - Do you really need ML to solve this problem? More specifically, do you need to solve the problem at all? Have you tried using rules or simple statistics to solve the problem?
  - Is it ethical to use ML to solve this problem? We have a whole lecture about ethics!

# How to Pick Problems to Solve with ML

- Just like any other project prioritization, you want to look for use cases that have **high impact** and **low cost**:
  - **High-impact** problems are likely to be those that address friction in your product, complex parts of your pipeline, places where cheap prediction is valuable, and generally what other people in your industry are doing.
  - **Low-cost** projects are those with available data, where bad predictions are not too harmful.

# How to Pick Problems to Solve with ML



# High-Impact Projects

- Here are some heuristics that you can use to find high-impact ML projects:
  - Find problems that ML takes from economically infeasible to feasible.
  - Think about what your product needs.
  - Think about the types of problems that ML is particularly good at.
  - Look at what other people in the industry are doing.

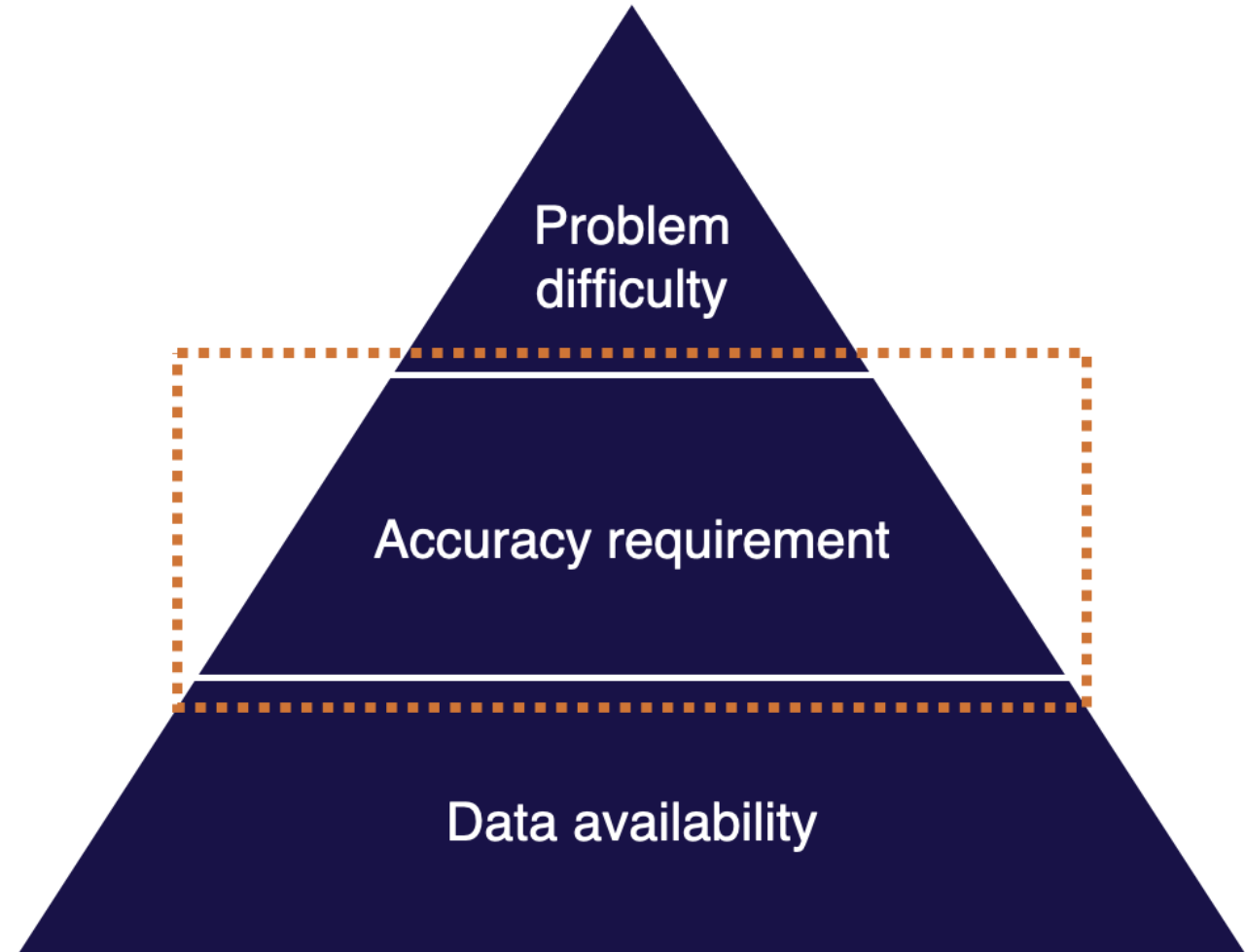
# Low-Cost Projects

- There are three main drivers for how much a project will cost:
  - **Data availability:** How hard is it to acquire data? How expensive is data labeling? How much data will be needed? How stable is the data? What data security requirements do you have?
  - **Accuracy requirement:** How costly are wrong predictions? How frequently does the system need to be right to be useful? What are the ethical implications of your model making wrong predictions? It is noteworthy that ML project costs tend to scale super-linearly in the accuracy requirement.
  - **Problem difficulty:** Is the problem well-defined enough to be solved with ML? Is there good published work on similar problems? How much compute does it take to solve the problem? Generally, it's hard to reason about what's feasible in ML.

# Low-Cost Projects

## Cost drivers

---



## Main considerations

---

- Is the problem well-defined?
  - Good published work on similar problems?  
(newer problems mean more risk & more technical effort)
  - Compute requirements?
  - Can a human do it?
- 
- How costly are wrong predictions?
  - How frequently does the system need to be right to be useful?
  - Ethical implications?
- 
- How hard is it to acquire data?
  - How expensive is data labeling?
  - How much data will be needed?
  - How stable is the data?
  - Data security requirements?



# What's Hard in ML?

- Here are the three types of hard problems:
  - **Output is complex:** The model predictions are ambiguous or in a high-dimensional structure.
  - **Reliability is required:** ML systems tend to fail in unexpected ways, so anywhere you need high precision or high robustness is going to be more difficult to solve with ML.
  - **Generalization is required:** These problems tend to be more in the research domain. They can deal with out-of-distribution data or do tasks such as reasoning, planning, or understanding causality.

# What's Hard in ML?

|                            | Instances   | Examples  |
|----------------------------|---|---|
| Output is complex          | <ul style="list-style-type: none"><li>• High-dimensional output</li><li>• Ambiguous output</li></ul>                | <ul style="list-style-type: none"><li>• 3D reconstruction</li><li>• Video prediction</li><li>• Dialog systems</li><li>• Open-ended recommender systems</li></ul>          |
| Reliability is required    | <ul style="list-style-type: none"><li>• High precision is required</li><li>• Robustness is required</li></ul>       | <ul style="list-style-type: none"><li>• Failing safely out-of-distribution</li><li>• Robustness to adversarial attacks</li><li>• High-precision pose estimation</li></ul> |
| Generalization is required | <ul style="list-style-type: none"><li>• Out of distribution data</li><li>• Reasoning, planning, causality</li></ul> | <ul style="list-style-type: none"><li>• Self-driving: edge cases</li><li>• Self-driving: control</li><li>• Small data</li></ul>   |

# ML Feasibility Assessment

- This is a quick checklist you can use to assess the feasibility of your ML projects:
  - Make sure that you actually need ML.
  - Put in the work upfront to define success criteria with all of the stakeholders.
  - Consider the ethics of using ML.
  - Do a literature review.
  - Try to rapidly build a labeled benchmark dataset.
  - Build a "minimum" viable model using manual rules or simple heuristics.
  - Answer this question again: "Are you sure that you need ML at all?"

# Not All ML Projects Should Be Planned The Same Way

- The three archetypes offered here are defined by how they interact with real-world use cases:
  - **Software 2.0 use cases:** Broadly speaking, this means taking something that software or a product does in an automated fashion today and augmenting its automation with machine learning. An example of this would be improving code completion in the IDE (like Github Copilot).
  - **Human-in-the-loop systems:** Machine learning can be applied for tasks where automation is not currently deployed - but where humans could have their judgment or efficiency augmented. Simply put, helping humans do their jobs better by complementing them with ML-based tools. An example of this would be turning sketches into slides, a process will usually involve humans approving the output of a machine learning model that made the slides.
  - **Autonomous systems:** Systems that apply machine learning to augment existing or implement new processes without human input. An example of this would be full self-driving, where there is no opportunity for a driver to intervene in the functioning of the car.

# Not All ML Projects Should Be Planned The Same Way

## Key questions

---

### Software 2.0

- Do your models truly improve performance?
- Does performance improvement generate business value?
- Do performance improvements lead to a data flywheel?

### Human-in-the-loop

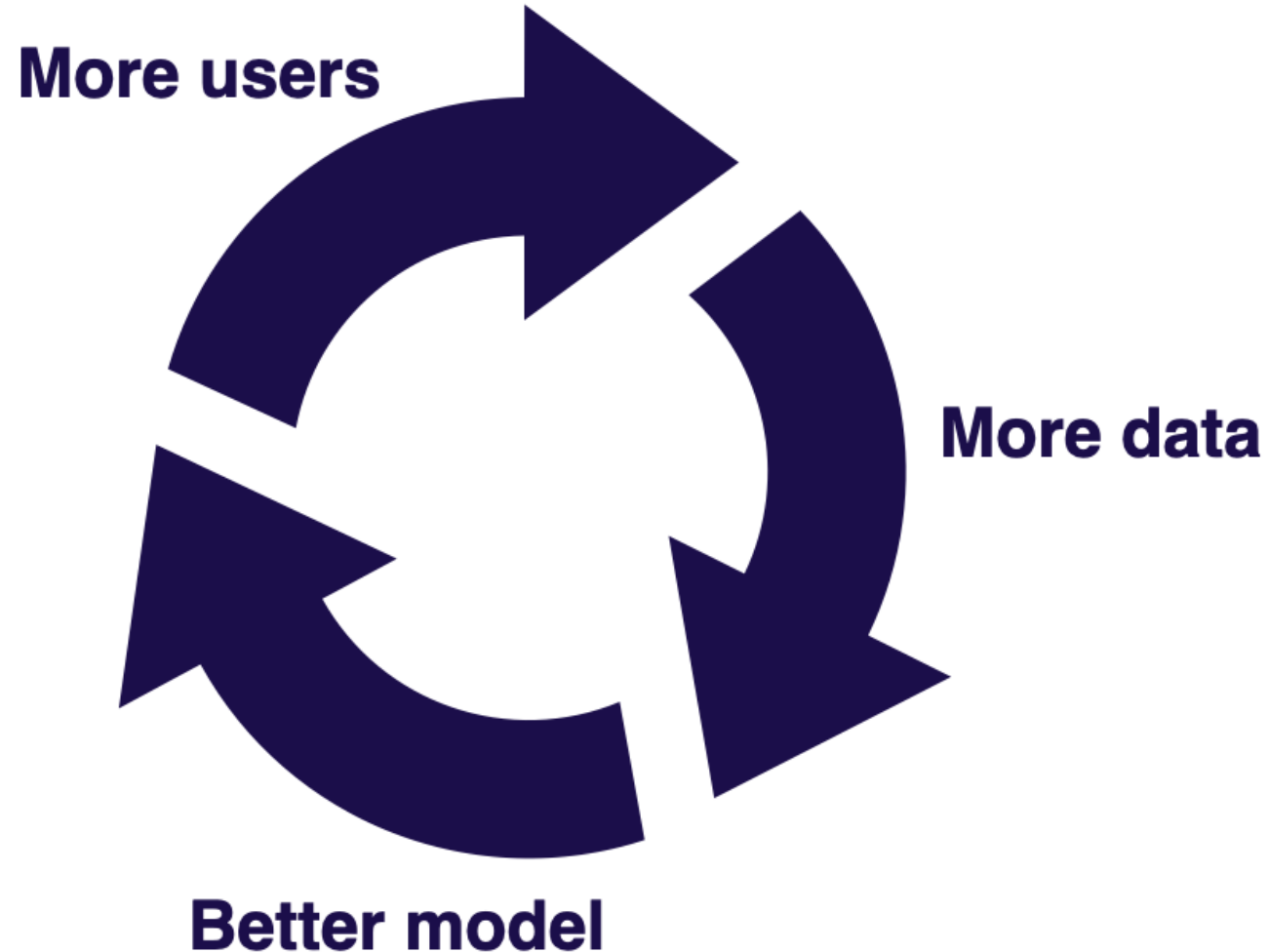
- How good does the system need to be to be useful?
- How can you collect enough data to make it that good?

### Autonomous systems

- What is an acceptable failure rate for the system?
- How can you guarantee that it won't exceed that failure rate?
- How inexpensively can you label data from the system?

# Data Flywheels

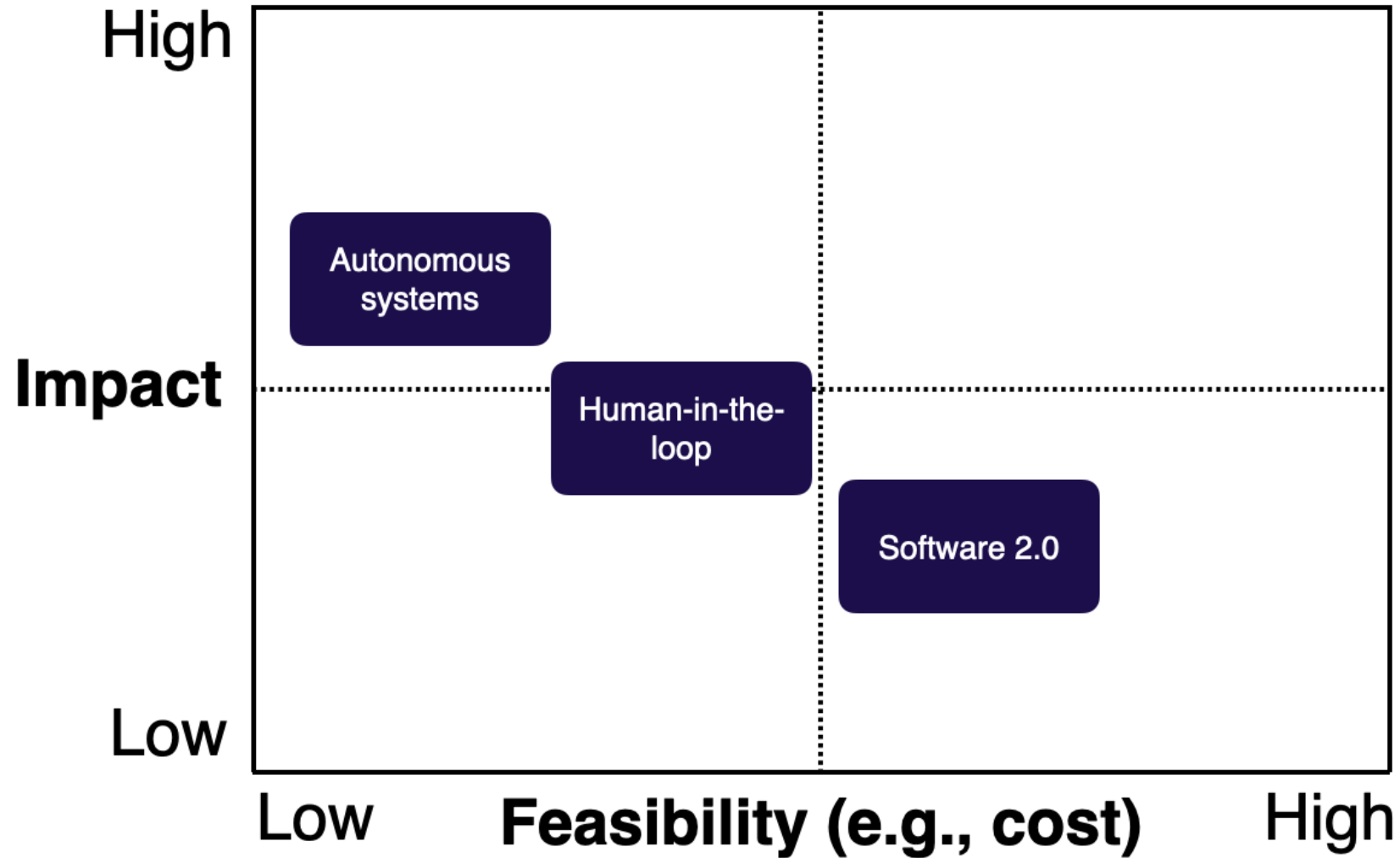
- As you build a software 2.0 project, strongly consider the concept of the data flywheel.



# Data Flywheels

- **Do you have a data loop?** To build a data flywheel, you crucially need to be able to get labeled data from users in a scalable fashion. This helps increase access to high-quality data and define a data loop.
- **Can you turn more data into a better model?** This somewhat falls onto you as the modeling expert, but it may also not be the case that more data leads to significantly better performance. Make sure you can actually translate data scale into better model performance.
- **Does better model performance lead to better product use?** You need to verify that improvements with models are actually tied to users enjoying the product more and benefiting from it!

# Impact and Feasibility of ML Product Archetypes

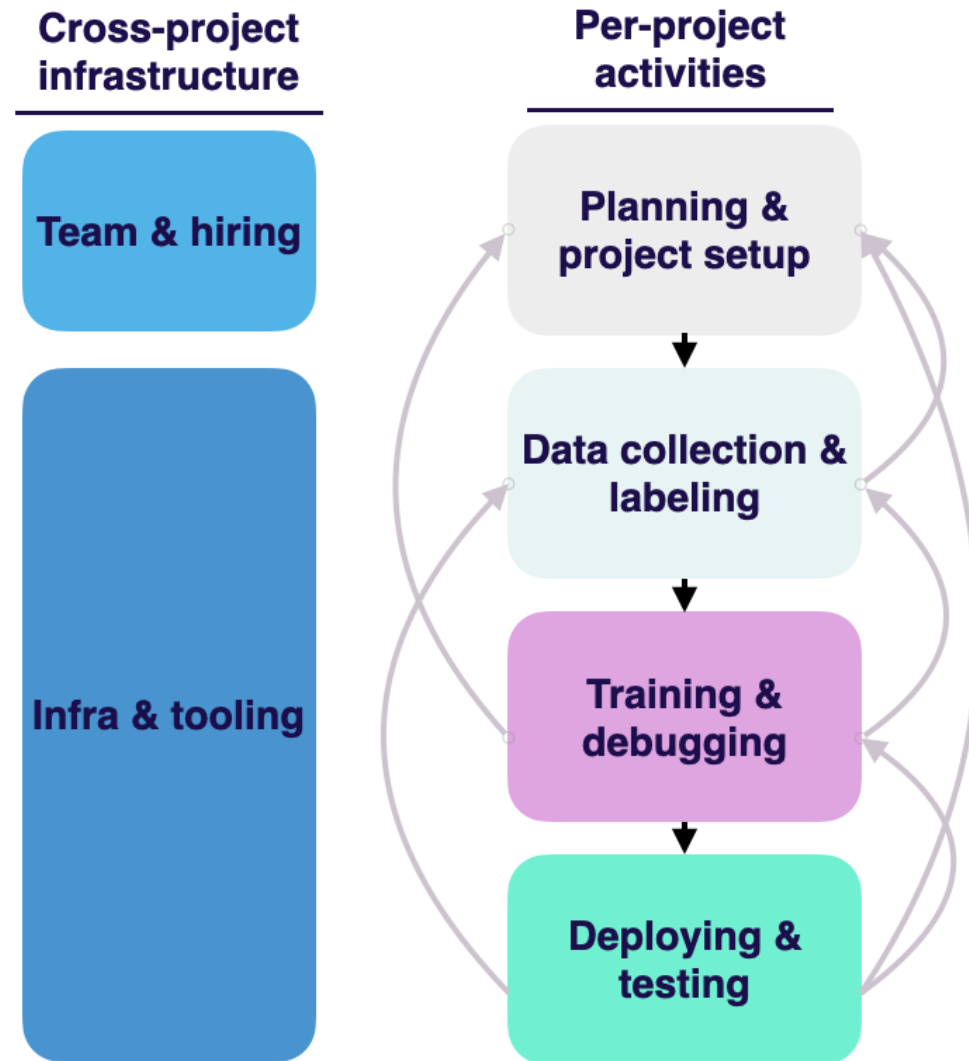




# Just Get Started!

- With all this discussion about archetypes and impact matrices, don't forget the most important component of engineering:
  - Actually building!
  - Dive in and get started.
  - Start solving problems and iterate on solutions.

# Lifecycle



# Development Infrastructure & Tooling

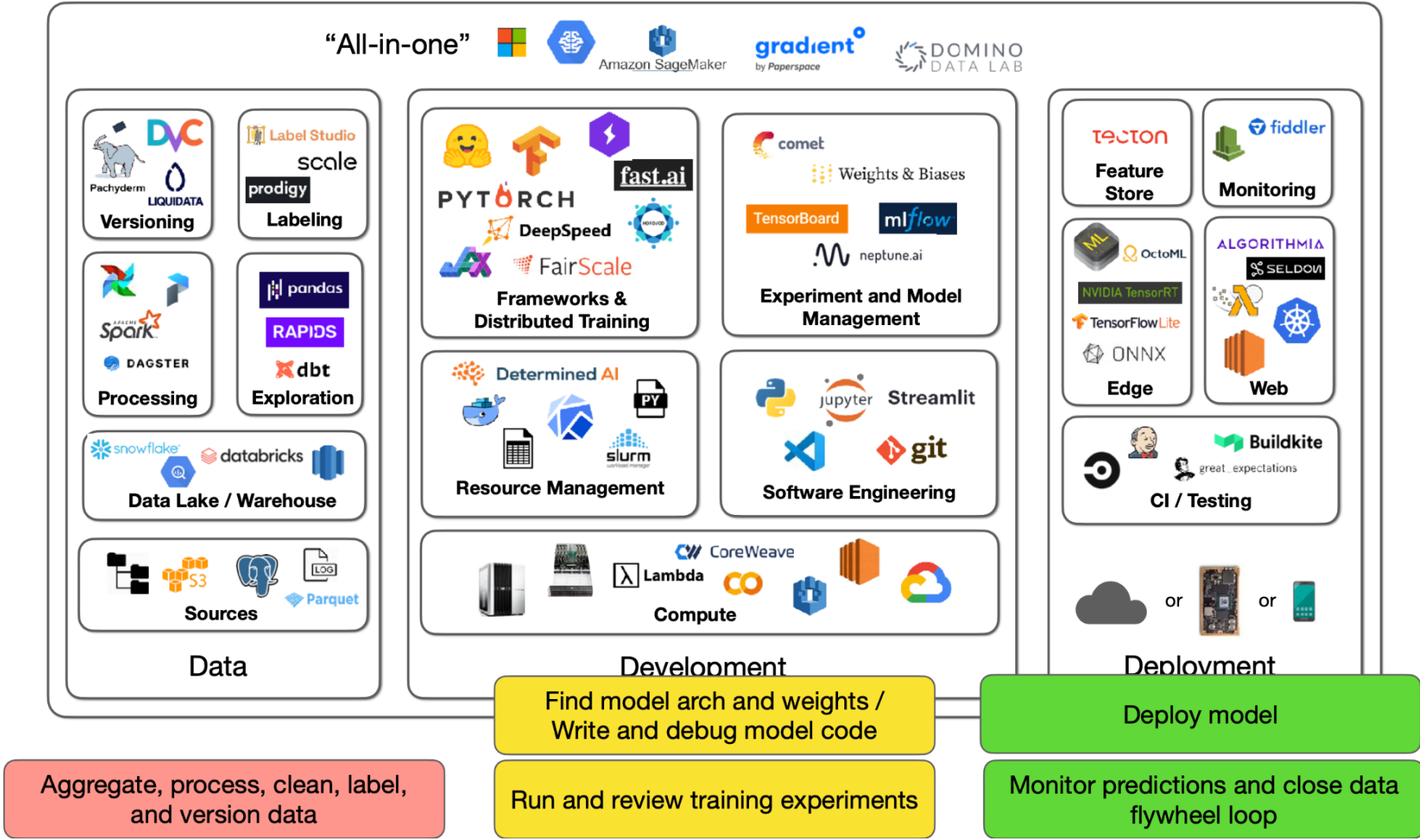
# Dream ML Project

- The dream of ML development is that given a project spec and some sample data, you get a continually improving prediction system deployed at scale.

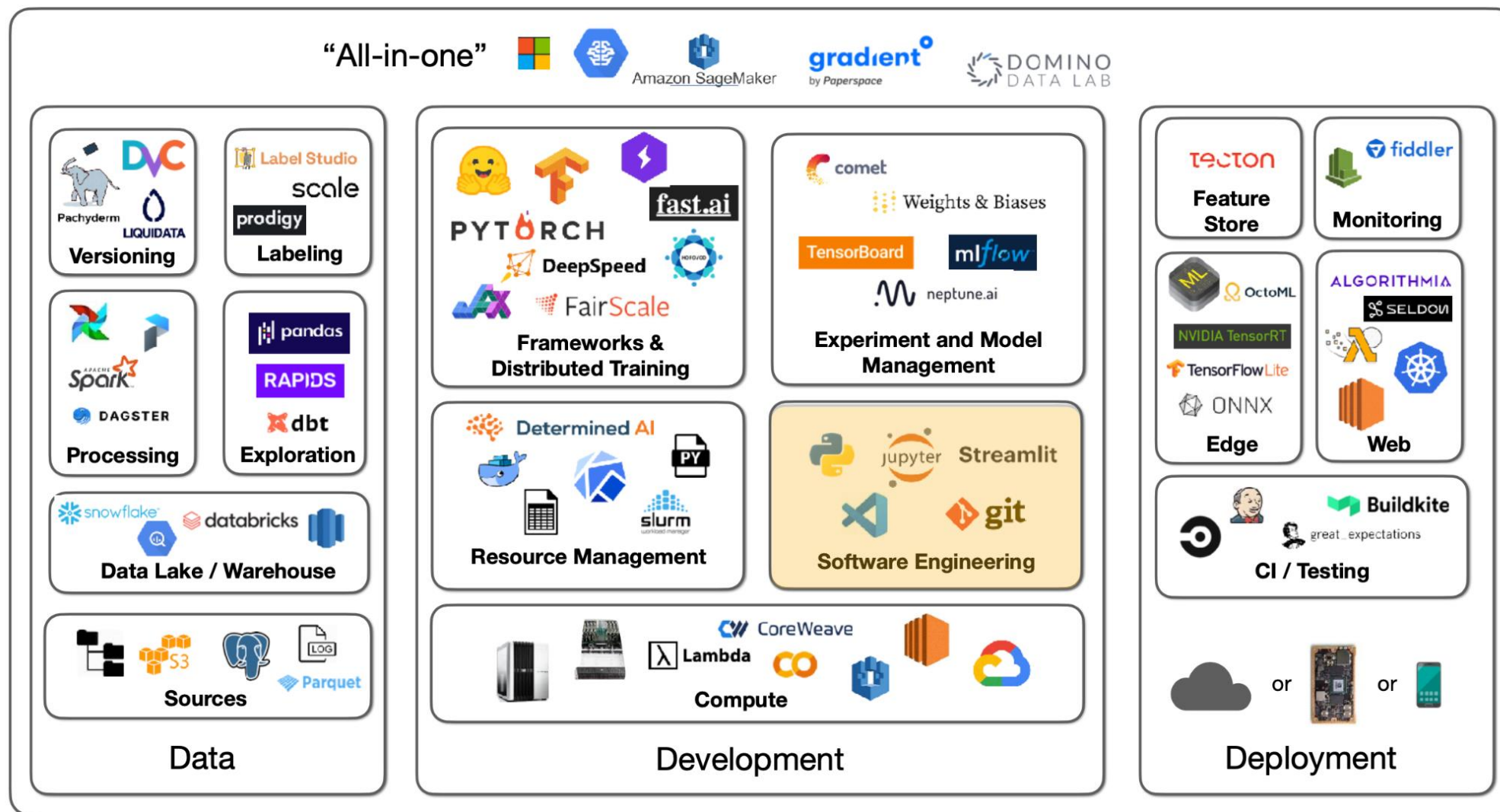
# Reality

- The reality is starkly different:
  - You have to collect, aggregate, process, clean, label, and version the data.
  - You have to find the model architecture and their pre-trained weights and then write and debug the model code.
  - You run training experiments and review the results, which will be fed back into the process of trying out new architectures and debugging more code.
  - You can now deploy the model.
  - After model deployment, you have to monitor model predictions and close the data flywheel loop. Basically, your users generate fresh data for you, which needs to be added to the training set.

# 3D – Data, Development, Deployment



# Software Engineering

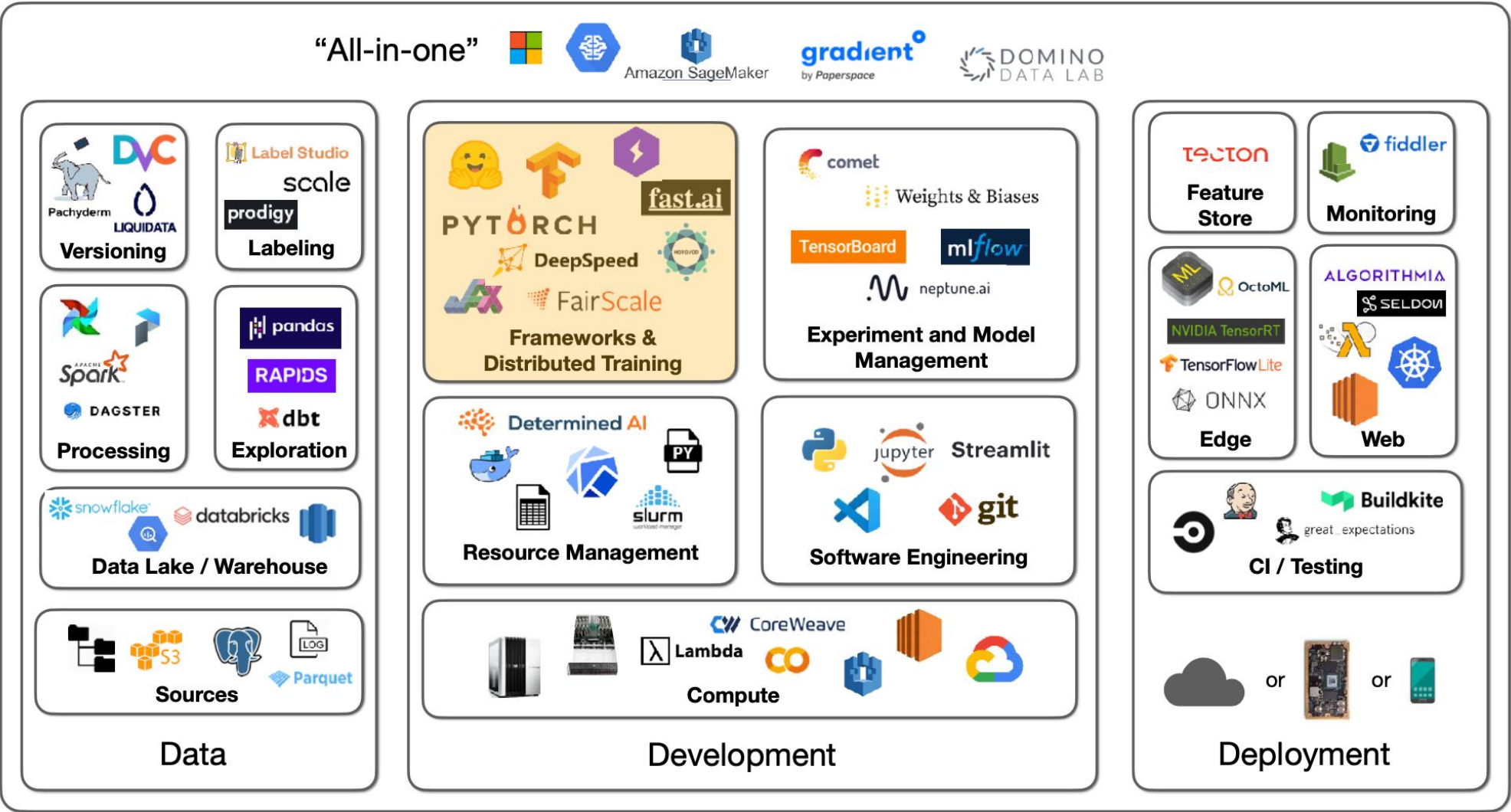


# Software Engineering

- Language
  - For your choice of programming language, Python is the clear winner in scientific and data computing because of all the libraries that have been developed. There have been some contenders like Julia and C/C++, but Python has really won out.
- Editors
  - To write Python code, you need an editor. You have many options, such as Vim, Emacs, Jupyter Notebook/Lab, VS Code, PyCharm, etc.



# Deep Learning Frameworks



# Deep Learning Frameworks

- There are various frameworks, such as PyTorch, TensorFlow, and Jax. They are all similar in that you first define your model by running Python code and then collect an optimized execution graph for different deployment patterns (CPU, GPU, TPU, mobile).
  - We prefer PyTorch because it is absolutely dominant by measures such as the number of models, the number of papers, and the number of competition winners. For instance, about 77% of 2021 ML competition winners used PyTorch.
  - With TensorFlow, you have TensorFlow.js (that lets you run deep learning models in your browser) and Keras (an unmatched developer experience for easy model development).
  - Jax is a meta-framework for deep learning.

# Deep Learning Frameworks





**Josh Tobin**

@josh\_tobin\_



Why do people always ask what ML framework to use?  
It's easy:

- jax is for researchers
-  - pytorch is for engineers 
- tensorflow is for boomers

6:24 PM · Mar 11, 2021 · Twitter Web App

---

**263** Retweets

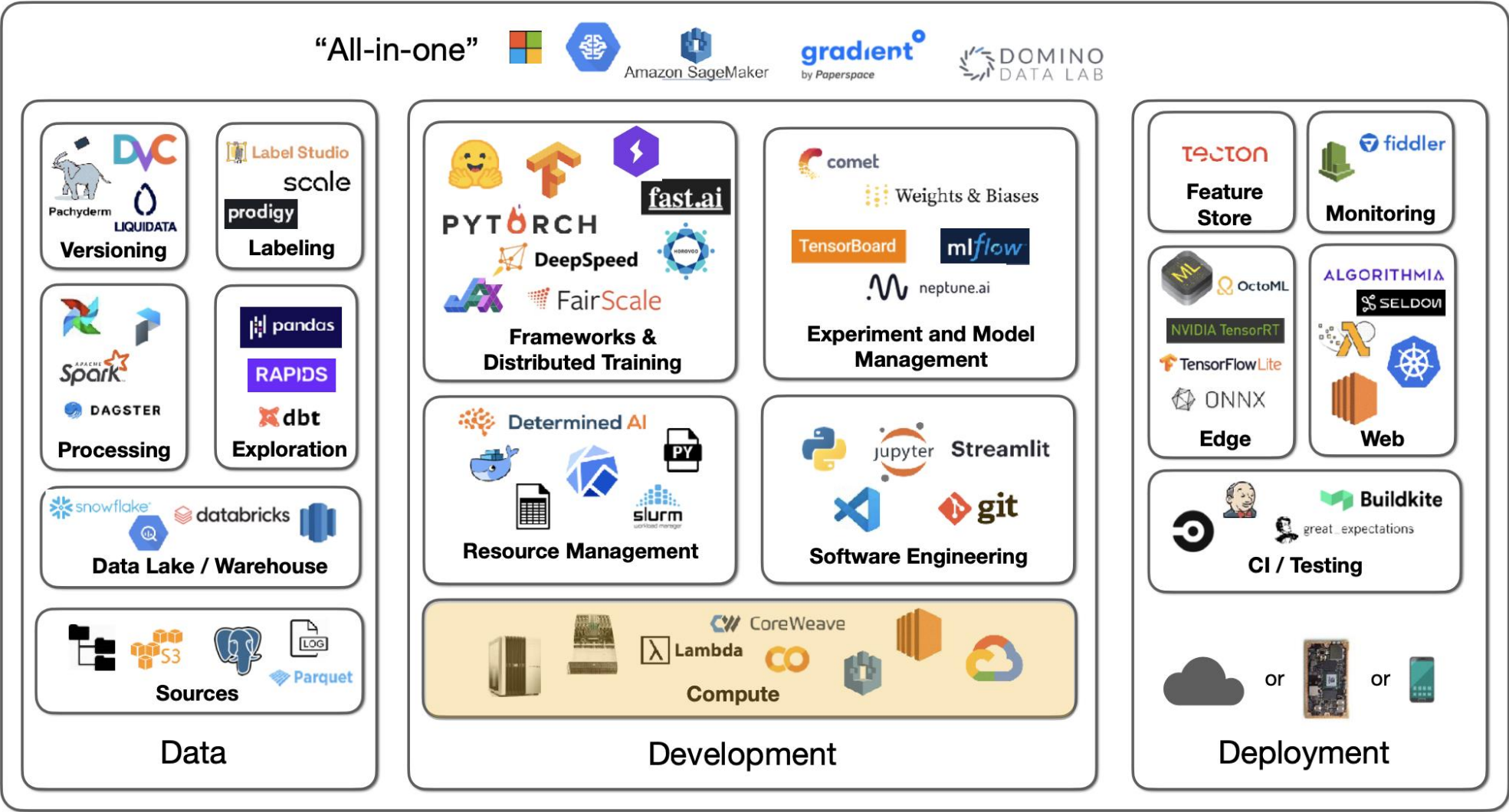
**65** Quote Tweets

**2,884** Likes

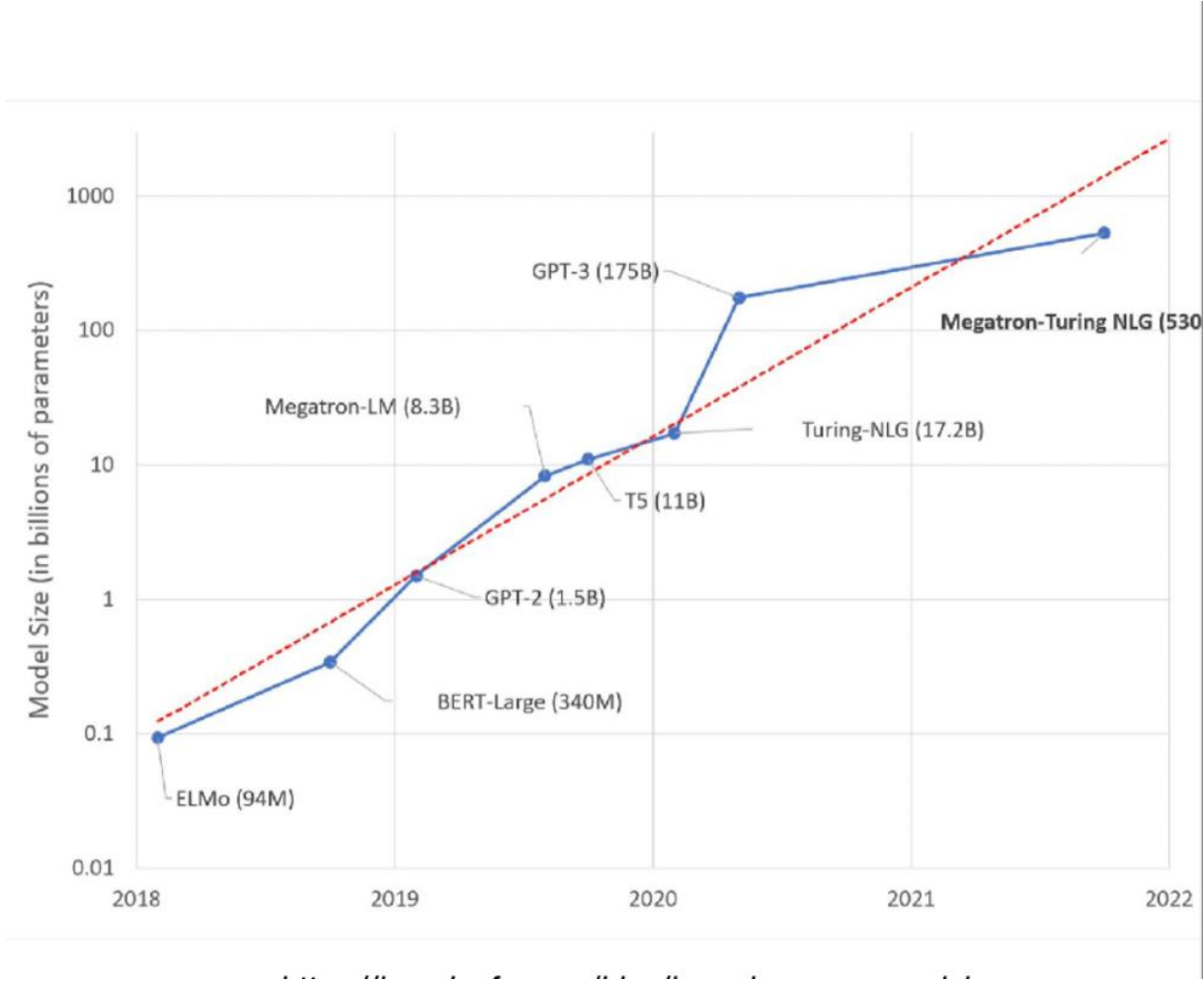
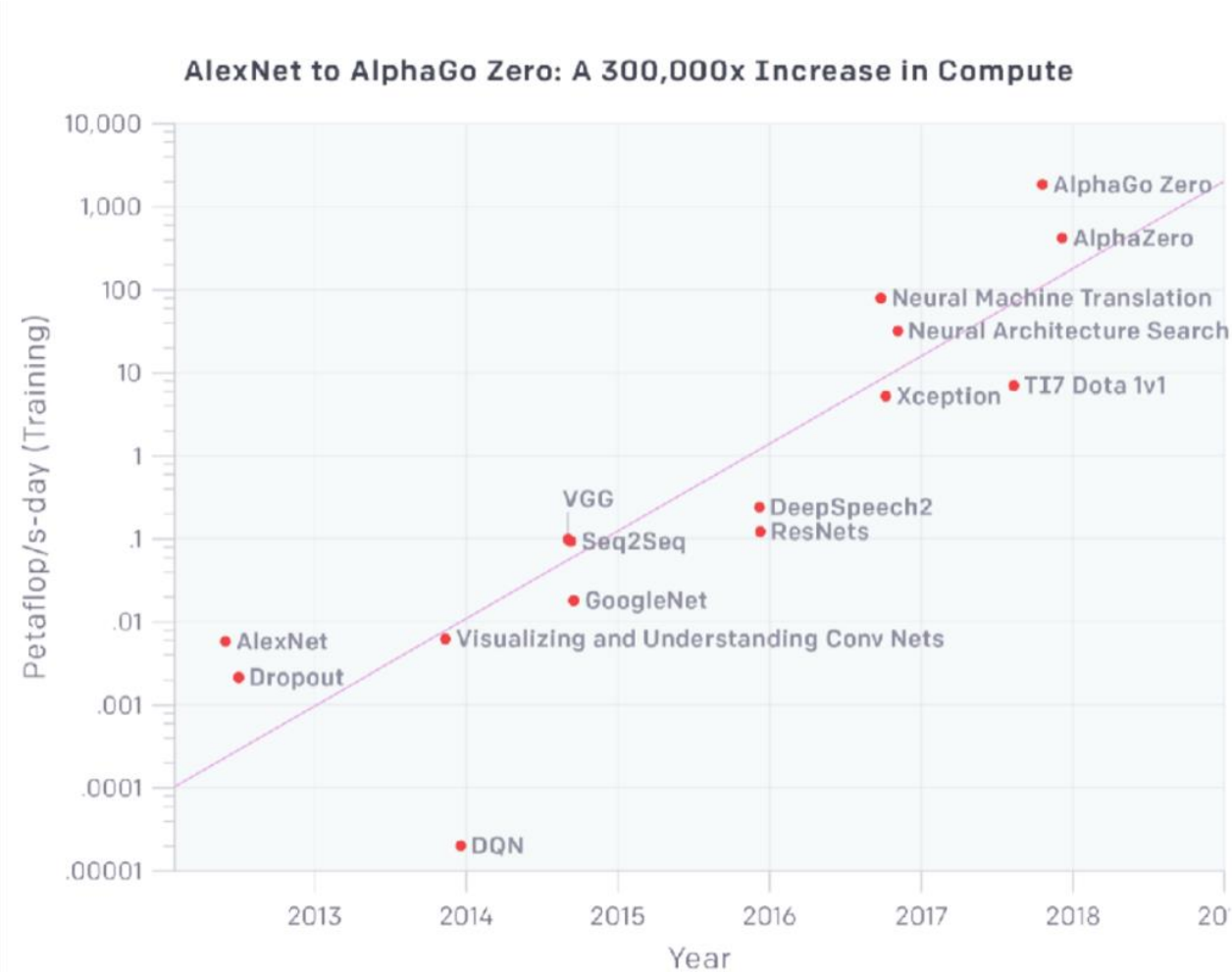
# Meta-Frameworks and Model Zoos

- Most of the time, you will start with at least a model architecture that someone has developed or published. You will use a specific architecture (trained on specific data with pre-trained weights) on a model hub.
  - **ONNX** is an open standard for saving deep learning models and lets you convert from one type of format to another. It can work well but can also run into some edge cases.
  - **HuggingFace** has become an absolutely stellar repository of models. It started with NLP tasks but has then expanded into all kinds of tasks (audio classification, image classification, object detection, etc.). There are 60,000 pre-trained models for all these tasks. There is a Transformers library that works with PyTorch, TensorFlow, and Jax. There are 7,500 datasets uploaded by people. There's also a community aspect to it with a Q&A forum.
  - **TIMM** is a collection of state-of-the-art computer vision models and related code that looks cool.

# Compute



# Compute



# GPUs

- To effectively train deep learning models, GPUs are required. NVIDIA has been the superior choice for GPU vendors, though Google has introduced TPUs (Tensor Processing Units) that are effective but are only available via Google Cloud. There are three primary considerations when choosing GPUs:
  - How much data fits on the GPU?
  - How fast can the GPU crunch through data? To evaluate this, is your data 16-bit or 32-bit? The latter is more resource intensive.
  - How fast can you communicate between the CPU and the GPU and between GPUs?



# NVIDIA GPUs

| Card        | Release | Arch   | Use-case | RAM (Gb) | 32bit TFlops | Tensor TFlops | 16bit |
|-------------|---------|--------|----------|----------|--------------|---------------|-------|
| K80         | 2014    | Kepler | Server   | 24       | 5            | N/A           | No    |
| P100        | 2016    | Pascal | Server   | 16       | 10           | N/A           | Yes   |
| V100        | 2017    | Volta  | Server   | 16 or 32 | 14           | 120           | Yes   |
| RTX 2080 Ti | 2018    | Turing | PC       | 11       | 13           | 107           | Yes   |
| RTX 3090    | 2021    | Ampere | PC       | 24       | 35           | 285           | Yes   |
| A100        | 2020    | Ampere | Server   | 40 or 80 | 19.5         | 312           | Yes   |
| A6000       | 2022    | Ampere | Server   | 48       | 38           | ?             | Yes   |

- **RAM:** should fit your model + meaningful batch of data
- **32bit vs Tensor TFlops**
  - Tensor Cores are specifically for deep learning operations (mixed precision)
- **16bit** ~doubles your effective RAM capacity



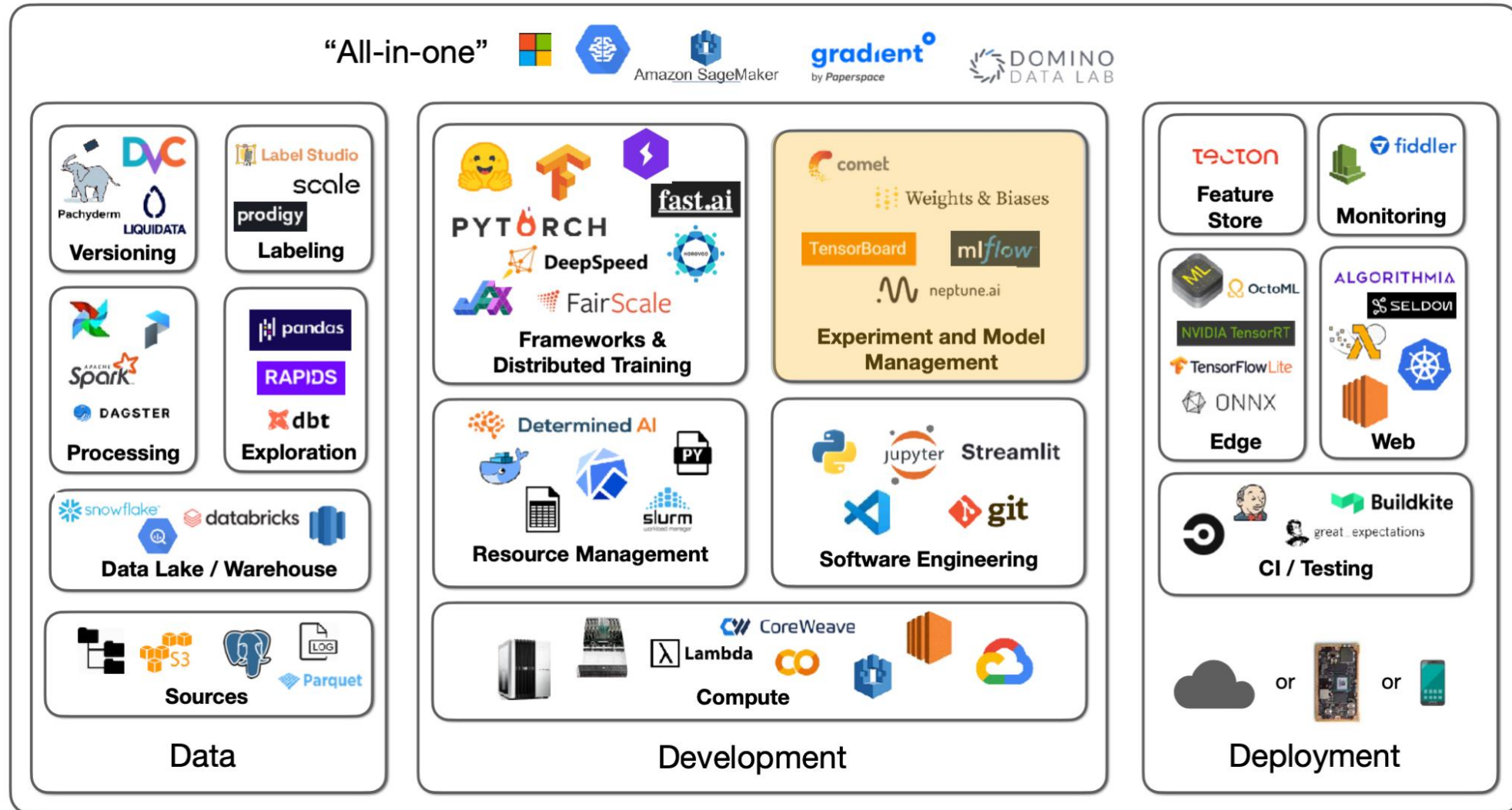
# On-Prem vs. Cloud

- For on-prem use cases, you can build your own pretty easily or opt for a pre-built computer from a company like NVIDIA.
- You can build a good, quiet PC with 128 GB RAM and 2 RTX 3090s for about \$7000 and set it up in a day.
- Going beyond this can start to get far more expensive and complicated.

# On-Prem vs. Cloud

- Some tips on on-prem vs. cloud use:
  - It can be useful to have your own GPU machine to shift your mindset from minimizing cost to maximizing utility.
  - To truly scale-out experiments, you should probably just use the most expensive machines in the least expensive cloud.
  - TPUs are worth experimenting with for large-scale training, given their performance.

# Experiment and Model Management



# Experiment and Model Management

- Experiment management refers to tools and processes that help us keep track of code, model parameters, and data sets that are iterated on during the model development lifecycle.
- Such tools are essential to effective model development.

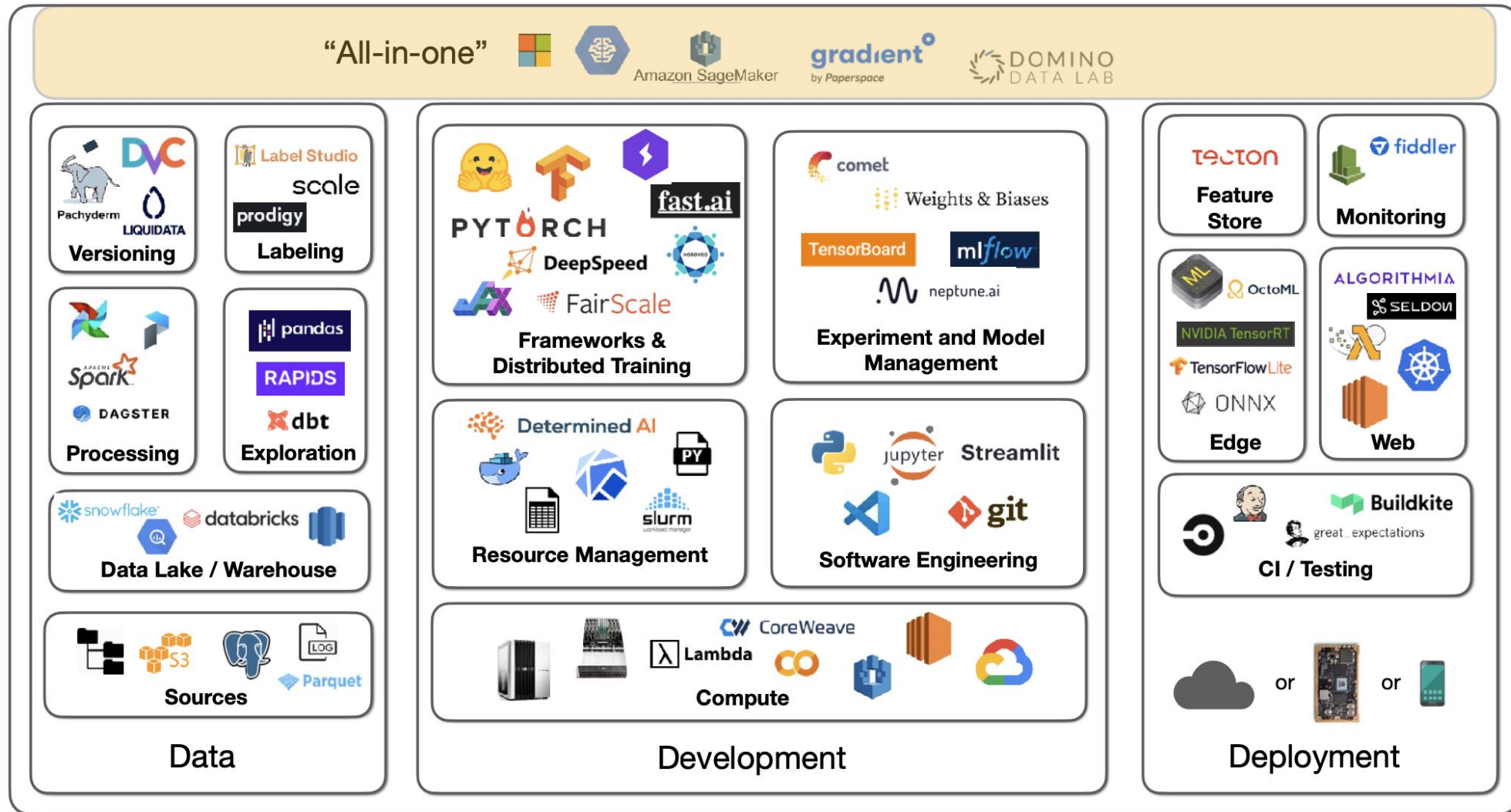
# Experiment and Model Management

- **TensorBoard:** A non-exclusive Google solution effective at one-off experiment tracking. It is difficult to manage many experiments.
- **MLflow:** A non-exclusive Databricks project that includes model packaging and more, in addition to experiment management. It must be self-hosted.
- **Weights and Biases:** An easy-to-use solution that is free for personal and academic projects! Logging starts simply with an "experiment config" command.
- Other options include Neptune AI, Comet ML, and Determined AI, all of which have solid experiment tracking options.

# Experiment and Model Management

- **TensorBoard:** A non-exclusive Google solution effective at one-off experiment tracking. It is difficult to manage many experiments.
- **MLflow:** A non-exclusive Databricks project that includes model packaging and more, in addition to experiment management. It must be self-hosted.
- **Weights and Biases:** An easy-to-use solution that is free for personal and academic projects! Logging starts simply with an "experiment config" command.
- Other options include Neptune AI, Comet ML, and Determined AI, all of which have solid experiment tracking options.
- Many of these platforms also offer intelligent hyperparameter optimization, which allows us to control the cost of searching for the right parameters for a model.

# "All-In-One"



# "All-In-One"

- There are machine learning infrastructure solutions that offer everything-- training, experiment tracking, scaling out, deployment, etc.
- These "all-in-one" platforms simplify things but don't come cheap! Examples include Gradient by Paperspace, Domino Data Lab, AWS Sagemaker, etc.