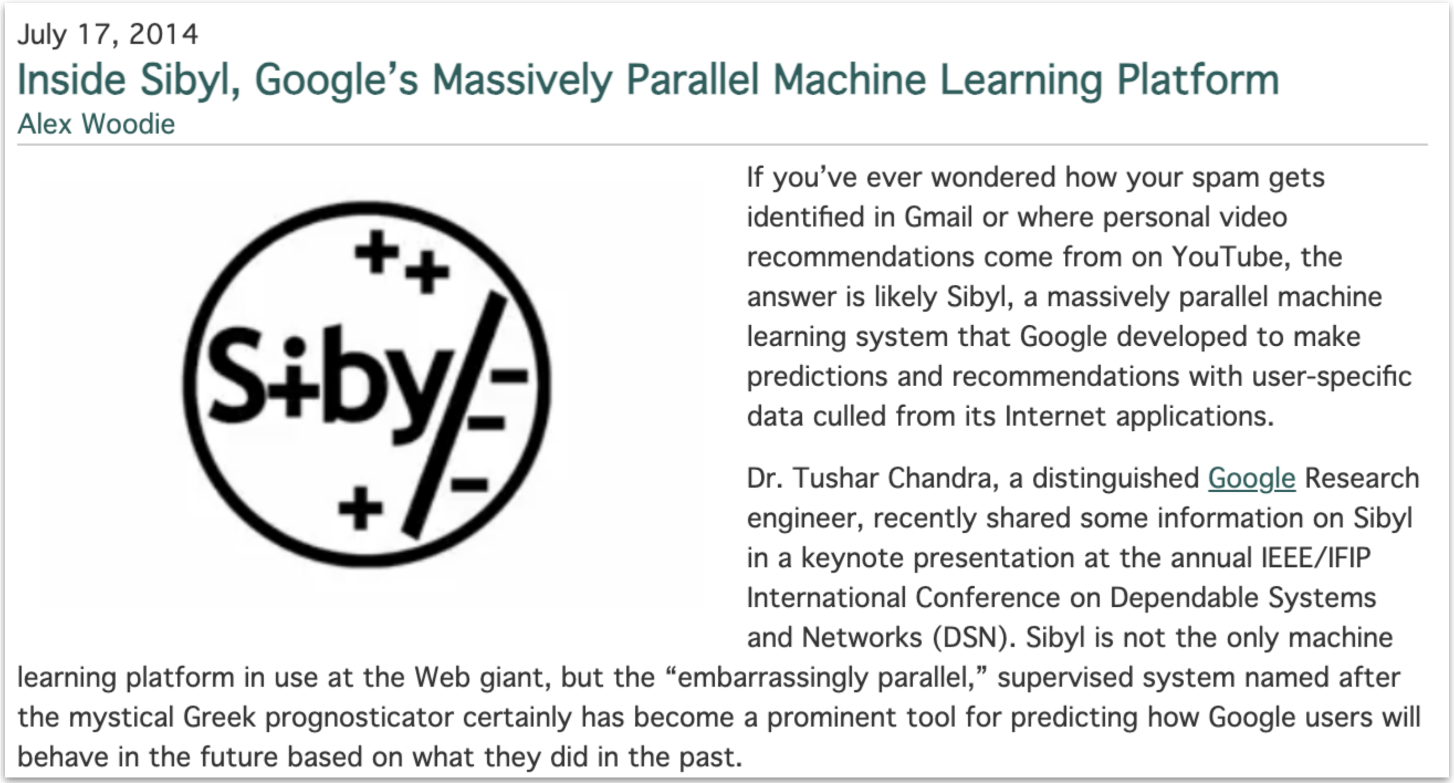
# Kubetorch,

or why ML development ends up on Kubernetes, why is it so hard,
and how we are trying to solve it.

**paul@run.house**
**🏃‍♀️Runhouse🏠**

## >>> The History and Evolution of AI Platforms

## Prehistoric (~2013-2016): “Algo as a service”



July 17, 2014

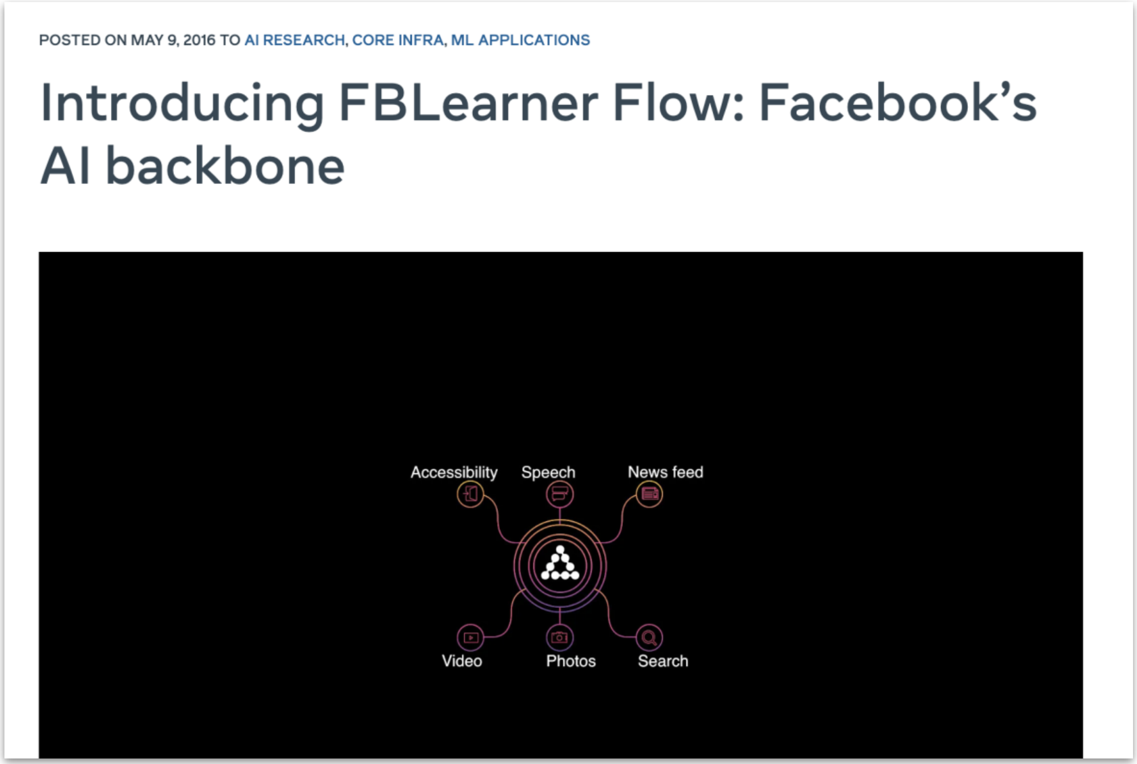
Inside Sibyl, Google’s Massively Parallel Machine Learning Platform

Alex Woodie

If you’ve ever wondered how your spam gets identified in Gmail or where personal video recommendations come from on YouTube, the answer is likely Sibyl, a massively parallel machine learning system that Google developed to make predictions and recommendations with user-specific data culled from its Internet applications.

Dr. Tushar Chandra, a distinguished Google Research engineer, recently shared some information on Sibyl in a keynote presentation at the annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). Sibyl is not the only machine learning platform in use at the Web giant, but the “embarrassingly parallel,” supervised system named after the mystical Greek prognosticator certainly has become a prominent tool for predicting how Google users will behave in the future based on what they did in the past.

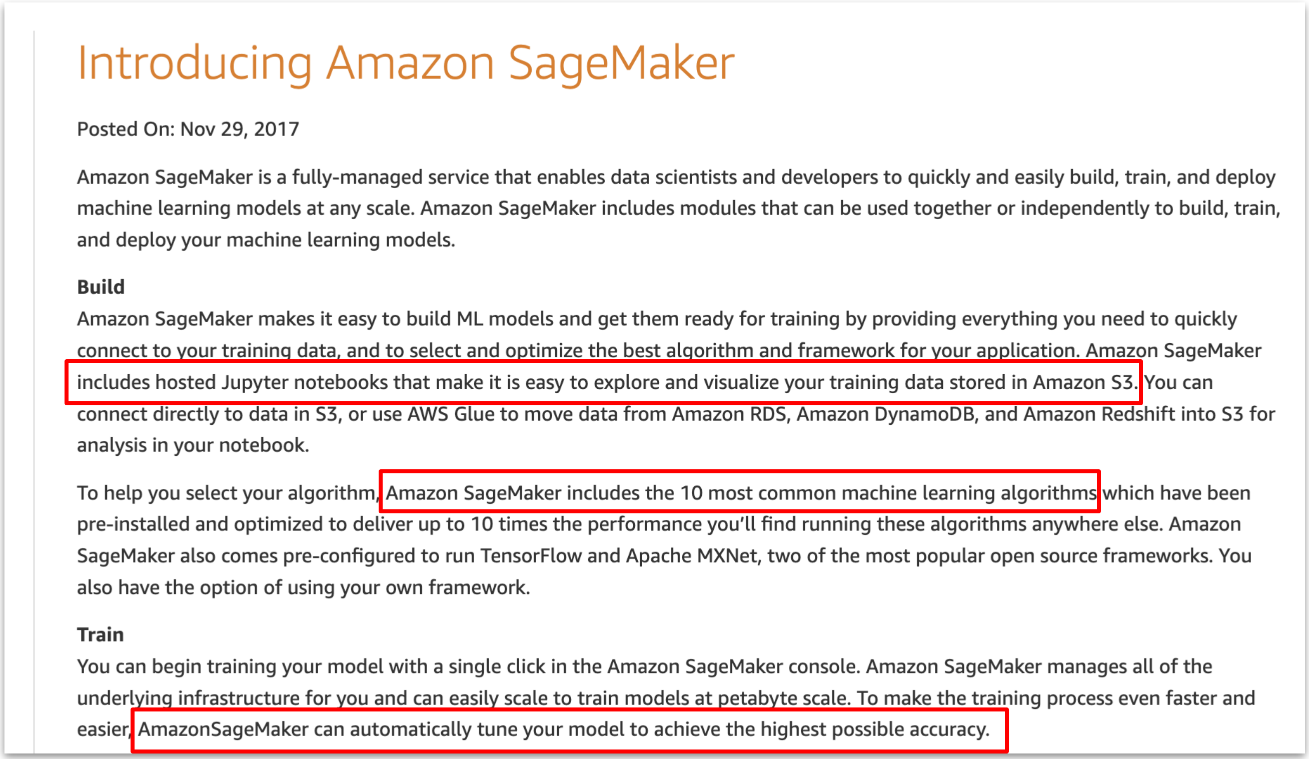
## Introduction of pipelines (and notebooks)


POSTED ON MAY 9, 2016 TO AI RESEARCH, CORE INFRA, ML APPLICATIONS

Introducing FBLearner Flow: Facebook's AI backbone

## Experimentation (2017-2020): Platform in a box

Notebooks, pipelines, and model endpoints, what more could you need?



Introducing Amazon SageMaker

Posted On: Nov 29, 2017

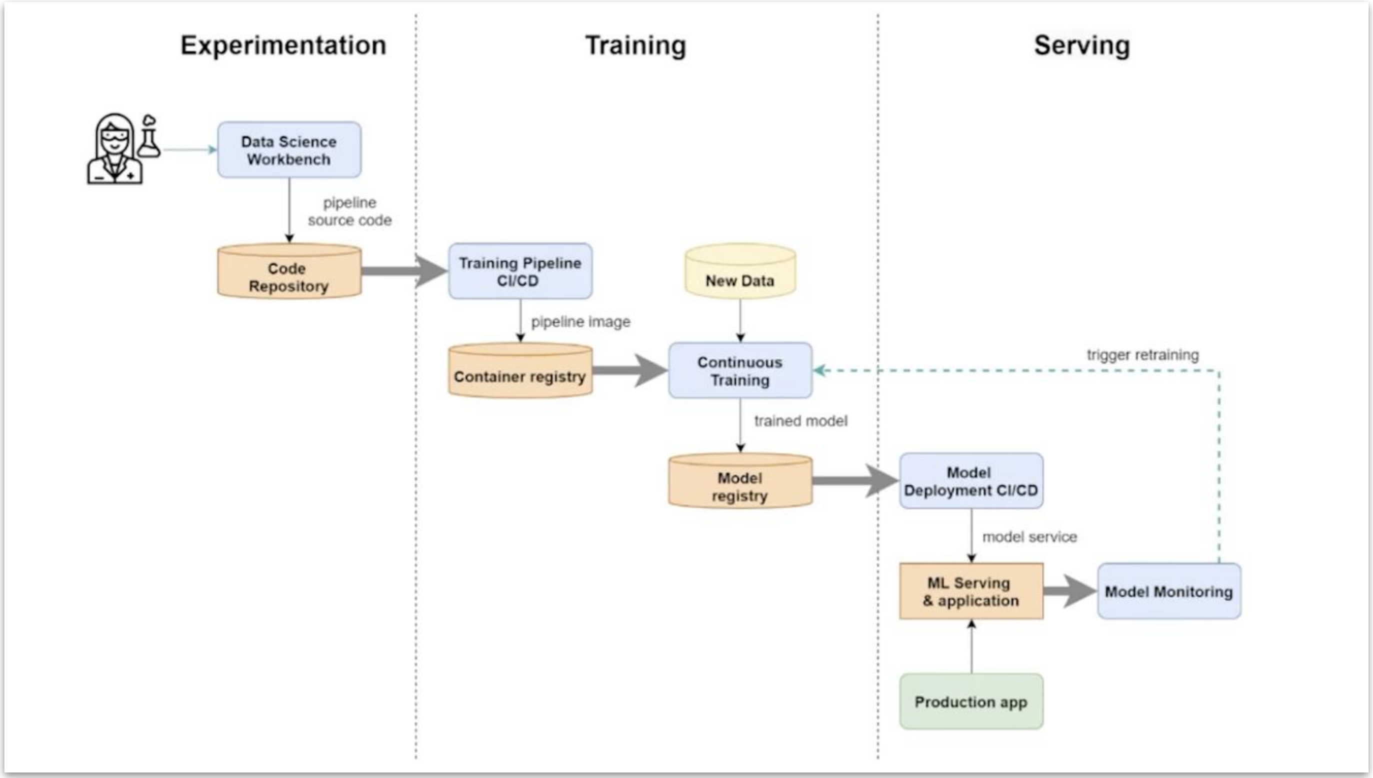
Amazon SageMaker is a fully-managed service that enables data scientists and developers to quickly and easily build, train, and deploy machine learning models at any scale. Amazon SageMaker includes modules that can be used together or independently to build, train, and deploy your machine learning models.

**Build**
Amazon SageMaker makes it easy to build ML models and get them ready for training by providing everything you need to quickly connect to your training data, and to select and optimize the best algorithm and framework for your application. Amazon SageMaker includes hosted Jupyter notebooks that make it is easy to explore and visualize your training data stored in Amazon S3. You can connect directly to data in S3, or use AWS Glue to move data from Amazon RDS, Amazon DynamoDB, and Amazon Redshift into S3 for analysis in your notebook.

To help you select your algorithm, Amazon SageMaker includes the 10 most common machine learning algorithms which have been pre-installed and optimized to deliver up to 10 times the performance you'll find running these algorithms anywhere else. Amazon SageMaker also comes pre-configured to run TensorFlow and Apache MXNet, two of the most popular open source frameworks. You also have the option of using your own framework.

**Train**
You can begin training your model with a single click in the Amazon SageMaker console. Amazon SageMaker manages all of the underlying infrastructure for you and can easily scale to train models at petabyte scale. To make the training process even faster and easier, AmazonSageMaker can automatically tune your model to achieve the highest possible accuracy.

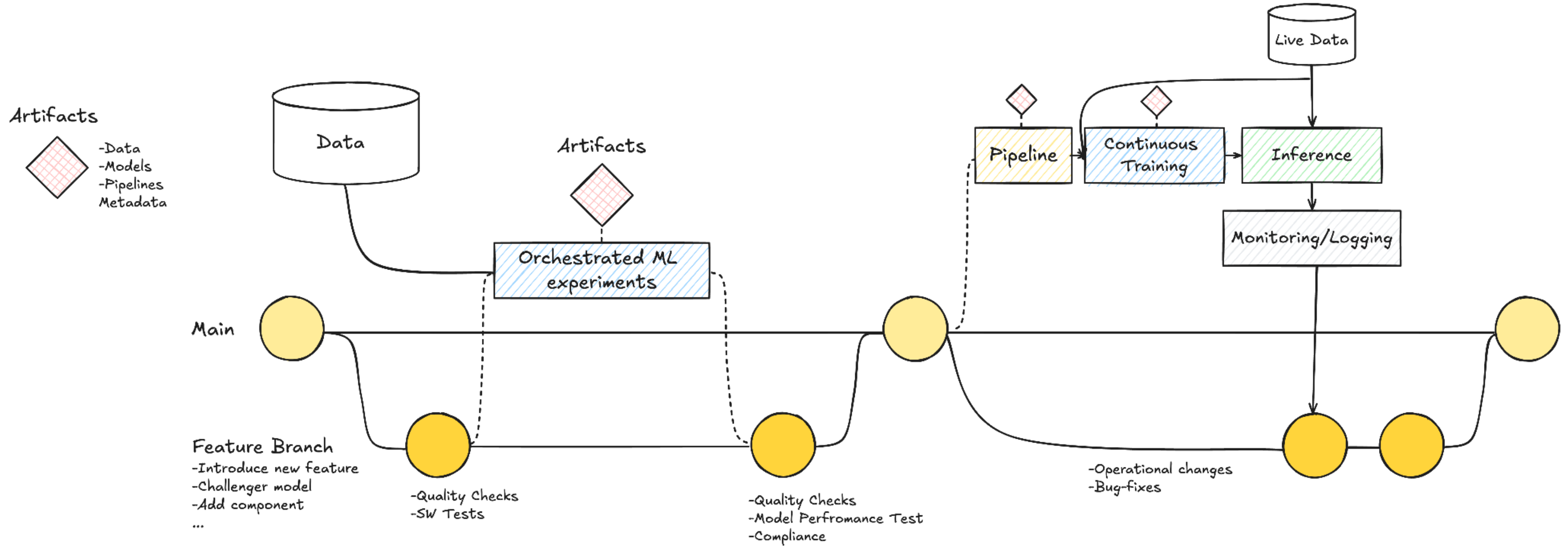
## The trap of these systems is a slow, human-centric process of reaching production



## Notebooks + DAGs does not scale

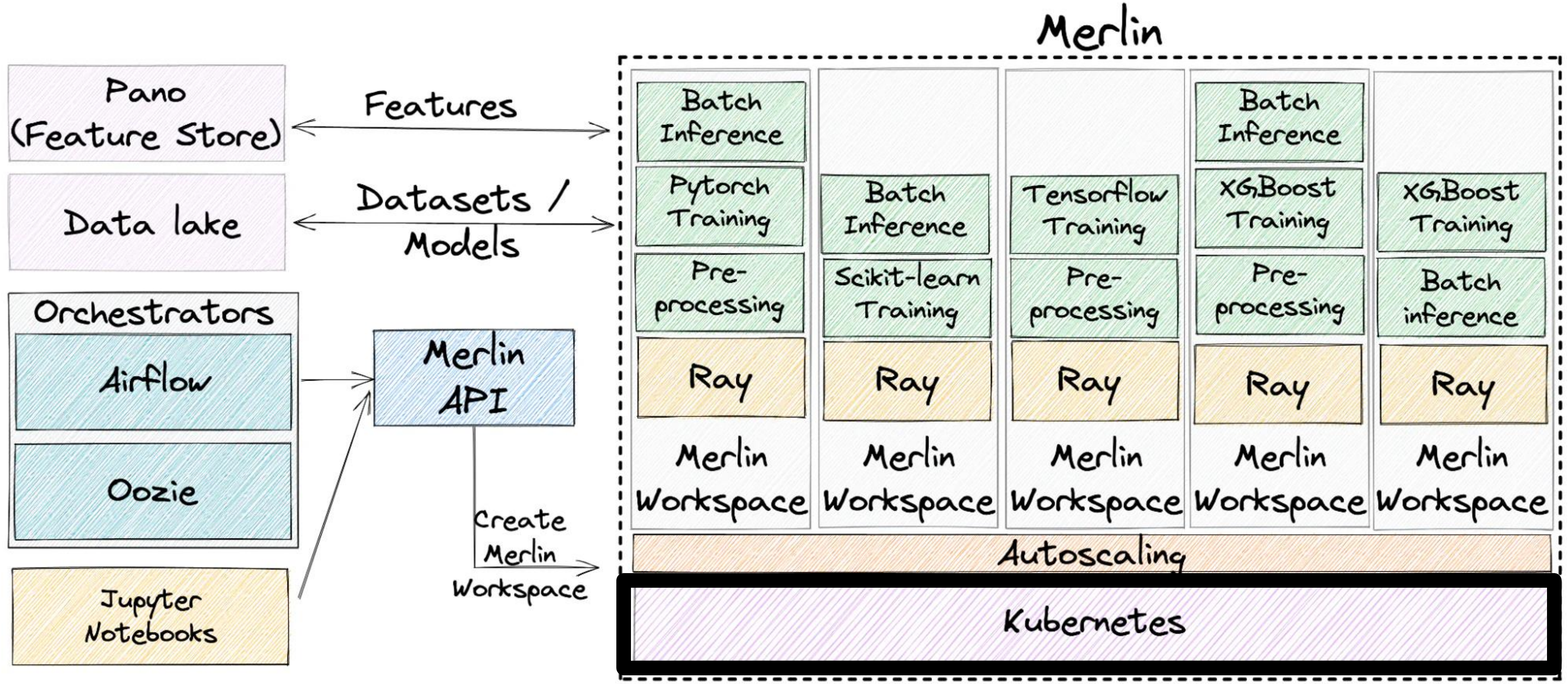
- Notebooks are fine for exploration, but the code cannot be version controlled, modules cannot be reused, and likely execution cannot reproduce.
- To translate notebooks for "production", a very long process is required to create a Airflow/Kubeflow pipeline; typically 3 weeks to 3 months.
- Debugging Airflow is a waste of time for your highest paid ML Engineers

# Centralization (2020-2022): Moving towards platforms



*Inspired by Volvo – Credit to Leonard Aukea*

## An ML platform enables self-service



*https://shopify.engineering/merlin-shopify-machine-learning-platform*

## The Scramble (2022-2024): 🤷♂

*<All my friends working on AI platforms pivot to GenAI apps>*

*<Many of them are now richer than I am>*

## >>> Why Kubernetes

## Reproducibility

- Setup, package version, driver version, and overall environment are extremely impactful to results and outputs

- Containerized applications are reproducible (notebooks, local scripts, devboxes are not), and Kubernetes exists to orchestrate containers.

- Declarative infrastructure for managing runtime compute

## Scalability

- Kubernetes lets you launch and scale replicas seamlessly while orchestrating the containers. There's demand for scaling across ML:
    - Distributed training
    - Hyperparameter optimization
    - Autoscaling inference
    - Data processing

- Scheduling and fair sharing of resources

- Bin-packing jobs effectively saves a ton of money
    - Azure sells 1 A10 with 24GB of VRAM with 440GB of RAM and 36 CPUs

## Portability

- Stages of research, dev, production are identical, and could just be separated by a namespace instead of requiring reconfiguration.

- Take your containers, launch a cluster anywhere you need to go (where the H200s are), and run your application.

- Idempotency and isolation

## Ecosystem and Integration

- Easily connects to cloud services using standard Kubernetes primitives and plugs into existing auth.

- Plug into the same monitoring, logging, and delivery pipelines as the rest of engineering.

- Many people at your organization already know how to run Kubernetes clusters, and it is easy to hire people too.

## What about Slurm?

- Popular with researchers, everyone’s first system in academia.

- Can somewhat enables rapid iteration loops while working with regular code on powerful infrastructure.

- But it’s best for research teams who typically “manually push enter”
    - CLI-based job submission has poor fault tolerance and debuggability properties
    - No online inference capabilities
    - Poor integration with broader modern platform engineering teams

## >>> What’s the Catch?

## Why Is ML Development Different from Software?

- **No local development** – A need for GPU acceleration, distributed compute, large datasets that won’t fit locally, and data restrictions.
- **Availability and efficiency matters** – My compute is multi-region and multi-cloud so I can access the GPU types I need for the lowest price.
- **Online and offline activities** – A single project spans batch data processing, research training, recurring training, and inference.

## Why does this matter? Development == Deployment

- It breaks a fundamental assumption for dev workflows on Kubernetes!
- There is no ability to locally run or test code *before* containerizing it for production deployment.
- As a result, there is a terrible dev loop: “Push and Pray” to run while iterating through deployment
  - 30-60 minutes to push to CI, build a container, have it get picked up and run

## Also, everyone hates Kubernetes and YAML

- People **loathe** debugging Kubernetes and YAML.
  - *Imagine asking your front-end engineers to dig through Kubernetes logs to figure out why an element was misaligned*

- Python is the language of choice for 95% of data science and ML developers.

## >>> So What Makes a Great AI/ML Platform?

## OLAP (e.g. Snowflake) Abstracts Infra Complexity

**Before Snowflake:**

- SQL worked fine for single-node queries
- For anything larger, you need to *do infrastructure* (Hadoop)

**After Snowflake (and OLAP databases generally):**

- Write regular SQL queries, completely non-optimized for scale
- Magic
- Executes over ephemerally allocated cloud compute that is entirely opaque to you

**<u>This is better</u>**

## So What Makes a Great AI Platform?

1. Makes heavy computation accessible and reproducible without opinionation

2. Introduces a standard which can be shared by both research (velocity) and production (robustness)

3. Makes it simple to propagate wins throughout the team

## Introducing Kubetorch

- Kubetorch is for infra people who love Kubernetes and the DS/MLEs who love PyTorch

- But it’s not just about PyTorch — *Kubetorch is to Kubeflow as PyTorch is to Tensorflow*:
    - Pythonic
    - Clean modular APIs
    - Eager and debuggable
    - A rich ecosystem

## Introducing Kubetorch

- Use Python to define the compute and distributed framework, and dispatch code for execution.
- For research iteration, <2s to rerun locally developed code (in regular Python).
- All APIs are in Python; no YAML, no kubectl.
- In production, execution happens identically because your orchestrator / CI does identical dispatch and remote execution.
- Completely agnostic to your methods, tools, cloud, etc.

## Distributed Training API

```
gpus = kt.Compute(
       gpus=1,
       image=kt.Image(image_id="nvcr.io/nvidia/pytorch:23.10-py3"),
   ).distribute("pytorch", workers=4)

train_ddp = kt.fn(train).to(gpus)
```

## Inference API

```python
img = kt.Image(image_id="nvcr.io/nvidia/pytorch:23.10-py3").pip_install(
    ["vllm"]
)
# Deploys with CLI command `kubetorch deploy `
@kt.compute(gpus="L4:8", image=img, name="deepseek_llama")
@kt.distribute(num_replicas(1,4))
class DeepSeekDistillLlama70BvLLM:
...

if __name__ == "__main__":
    # Load the deployed model service
    deepseek = DeepSeekDistillLlama70BvLLM.from_name("deepseek_llama")
```

## Code Demo

## Who are we?

**Runhouse** is a NYC-based **team of AI/ML infrastructure builders** who want to liberate ML teams from MLOps and infra work. We allows researchers and engineers to seamlessly develop, debug, and deploy **powerful distributed** ML workloads within their **own clouds and hardware**.

If you want to give us feedback or chat: [paul@run.house](mailto:paul@run.house)
It would be much appreciated!