DagsHub Data Streaming Client

SOTA Meetup, October 2022

WE'RE REVOLUTIONIZING THE WAY DATA SCIENTISTS INTERACT WITH UNSTRUCTURED DATA

October 23, 2022

DagsHub

Outline

1. What's the Challenge?
2. What's the Solution?

Outline

Contextualizing a Data-Heavy Problem

We're *data* scientists.

Contextualizing a Data-Heavy Problem

We're *data* scientists. Working with data is our DNA.

Contextualizing a Data-Heavy Problem

We're *data* scientists. Working with data is our DNA.

This is sometimes tedious – tools don't always work seamlessly - most don't even deal with data!

Contextualizing a Data-Heavy Problem

We're *data* scientists. Working with data is our DNA.

This is sometimes tedious – tools don't always work seamlessly - most don't even deal with data!

Why does that matter?

Contextualizing a Data-Heavy Problem

We're *data* scientists. Working with data is our DNA.

This is sometimes tedious – tools don't always work seamlessly - most don't even deal with data!

Why does that matter? Every new paradigm needs to adapt to a default, mostly set-in-stone standard – oftentimes, that's done hastily and hackily.

Contextualizing a Data-Heavy Problem

Let's take a look at a traditional training pipeline.
URL: `https://colab.research.google.com/drive/1TTsAwism1t0PoTpvKTKkuWywiCxxK0ed/`

Identifying the Bottleneck

There's a couple of challenges that we can identify:

Identifying the Bottleneck

There's a couple of challenges that we can identify:

a. CUDNN is **annoying** to work with.

Identifying the Bottleneck

There's a couple of challenges that we can identify:

a. CUDNN is **annoying** to work with.
   Solution? @ NVIDIA →

Identifying the Bottleneck

There's a couple of challenges that we can identify:

a. CUDNN is **annoying** to work with.
   Solution? @ NVIDIA →

b. Dependency Hell.

Identifying the Bottleneck

There's a couple of challenges that we can identify:

a. CUDNN is **annoying** to work with.
   Solution? @ NVIDIA →

b. Dependency Hell.
   Solution? Semantic Versioning (@ everyone).

Identifying the Bottleneck

There's a couple of challenges that we can identify:

a. CUDNN is **annoying** to work with.
   Solution? @ NVIDIA →

b. Dependency Hell.
   Solution? Semantic Versioning (@ everyone).

c. Data takes too long too load.

Identifying the Bottleneck

There's a couple of challenges that we can identify:

a. CUDNN is **annoying** to work with.
   Solution? @ NVIDIA →

b. Dependency Hell.
   Solution? Semantic Versioning (@ everyone).

c. Data takes too long too load.
   Solution? **Let's talk about it.**

The Problem with Pulling Data

Here's what we know:

- The data **exists**. Somewhere.

The Problem with Pulling Data

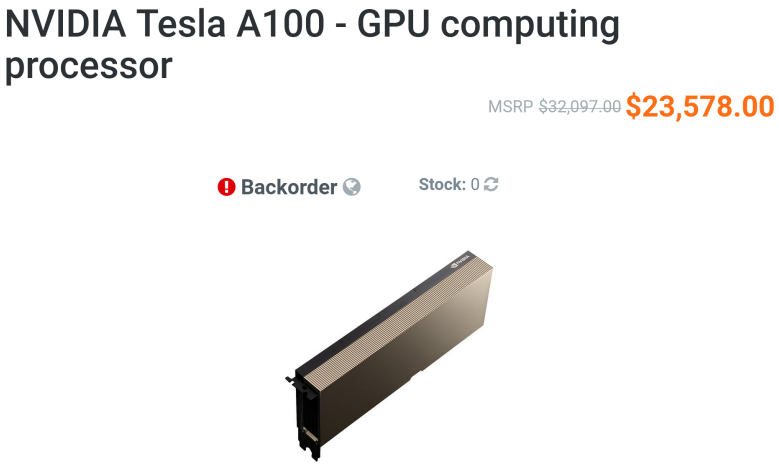
Here's what we know:

- The data **exists**. Somewhere.
- The time sink arises when we transfer that data to a high-compute environment.

The Problem with Pulling Data

Here's what we know:

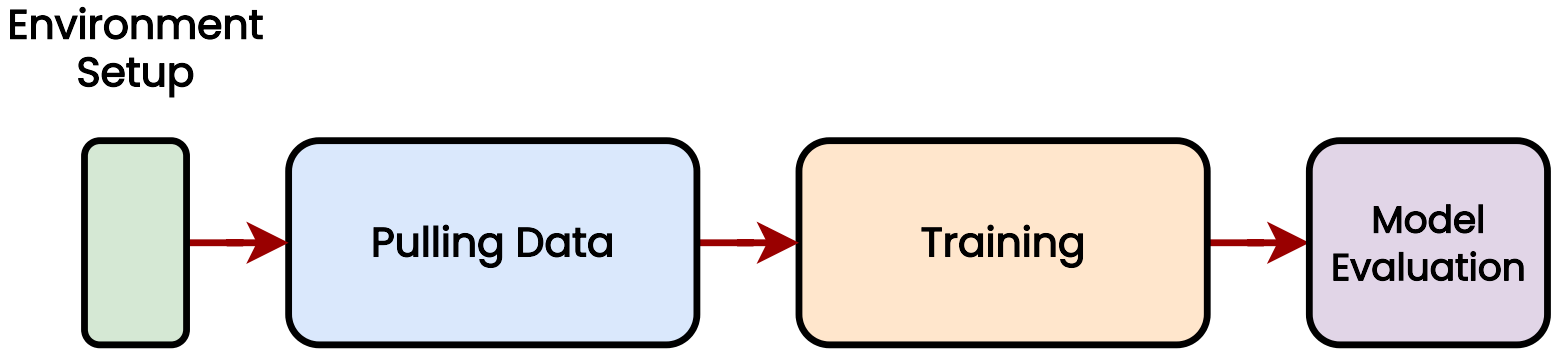
- The data **exists**. Somewhere.
- The time sink arises when we transfer that data to a high-compute environment.
- Buying one NVIDIA Tesla A100 costs $\$23,578$. Also, there's 0 stock.



Outline

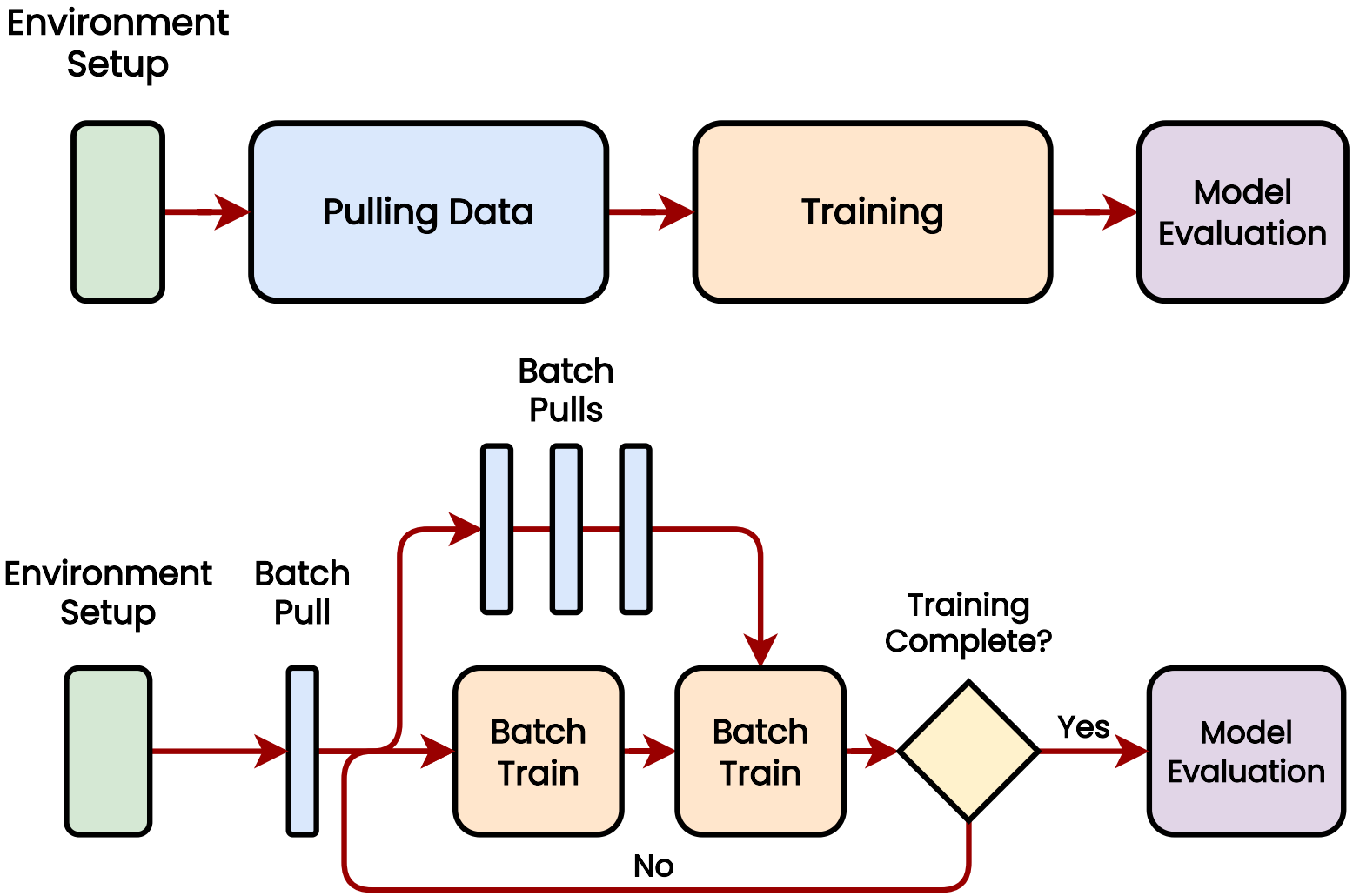
Streaming the Pain Away

What if, instead of pulling and training sequentially, we **merge** the processes?



Streaming the Pain Away

What if, instead of pulling and training sequentially, we **merge** the processes?



Streaming the Pain Away

So, how do we do this, while:

Streaming the Pain Away

So, how do we do this, while:

- Maintaining versioning,

Streaming the Pain Away

So, how do we do this, while:

- Maintaining versioning,
- Allowing Dataset Shuffling,

Streaming the Pain Away

So, how do we do this, while:

- Maintaining versioning,
- Allowing Dataset Shuffling,
- Preventing a full dataloader rewrite, and

Streaming the Pain Away

So, how do we do this, while:

- Maintaining versioning,
- Allowing Dataset Shuffling,
- Preventing a full dataloader rewrite, and
- **Keeping it user friendly!**

Flexible Data

We have not one, but TWO solutions to show you today!

Flexible Data

We have not one, but TWO solutions to show you today!

a. A custom filesystem for big-data interfacing.

Flexible Data

We have not one, but TWO solutions to show you today!

a. A custom filesystem for big-data interfacing.

b. A broad monkeypatched solution for universal remote data interfacing.

Implementation Idea

Hello, **FUSE!**

FUSE, or *Filesystem in Userspace*, is *nix software that allows us to create our filesystem without editing kernel code.

Implementation Idea

Hello, **FUSE!**

FUSE, or *Filesystem in Userspace*, is *nix software that allows us to create our filesystem without editing kernel code.

Using FUSE, we can build our own filesystem to handle file interfacing.

Implementation Idea

Hello, **FUSE!**

FUSE, or *Filesystem in Userspace*, is *nix software that allows us to create our filesystem without editing kernel code.

Using FUSE, we can build our own filesystem to handle file interfacing.

Then, using DagsHub's Web APIs, we can fetch files within the backend and pipe it as if it was locally present all the while!

Demonstrating the Training Pipeline

All put together, let's check out the new training pipeline!
URL: `https://colab.research.google.com/drive/1vTaZfOzchKxeC6nNVZvHxjFqWmXXmrJ6/`

Let's Monkeypatch!

Well, what if I don't have a *nix system?

Let's Monkeypatch!

Well, what if I don't have a *nix system? We got you covered. :)

Let's Monkeypatch!

Well, what if I don't have a *nix system? We got you covered. :)

Another way you can lazily accessing files – while running a python script only – is monkeypatching.

Let's Monkeypatch!

Well, what if I don't have a *nix system? We got you covered. :)

Another way you can lazily accessing files – while running a python script only – is monkeypatching.

Effectively, we take out the filesystem manipulation, and extrapolate that MVP to a single script.

Let's Monkeypatch!

Well, what if I don't have a *nix system? We got you covered. :)

Another way you can lazily accessing files – while running a python script only – is monkeypatching.

Effectively, we take out the filesystem manipulation, and extrapolate that MVP to a single script.

Here's an example: `https://colab.research.google.com/drive/1eKt2TIXj9wgFhA0Nu3G6ZW5IDC_9fdFU/`

Note: Monkeypatching currently has the widest range of support. When in doubt, **use monkeypatching!**

Direct Data Access

The **perfect** data pipeline can seamlessly manage data and codefiles. So, let's abstract DVC interfacing!

Direct Data Access

The **perfect** data pipeline can seamlessly manage data and codefiles. So, let's abstract DVC interfacing!

The new Direct Data Access can:

Direct Data Access

The **perfect** data pipeline can seamlessly manage data and codefiles. So, let's abstract DVC interfacing!

The new Direct Data Access can:

- Append to a directory

Direct Data Access

The **perfect** data pipeline can seamlessly manage data and codefiles. So, let's abstract DVC interfacing!

The new Direct Data Access can:

- Append to a directory
- Create DVC tracked files from scratch

Direct Data Access

The **perfect** data pipeline can seamlessly manage data and codefiles. So, let's abstract DVC interfacing!

The new Direct Data Access can:

- Append to a directory
- Create DVC tracked files from scratch
- Create a DVC directory from scratch

Direct Data Access

The **perfect** data pipeline can seamlessly manage data and codefiles. So, let's abstract DVC interfacing!

The new Direct Data Access can:

- Append to a directory
- Create DVC tracked files from scratch
- Create a DVC directory from scratch
- Access Git and DVC tracked files

Direct Data Access

The **perfect** data pipeline can seamlessly manage data and codefiles. So, let's abstract DVC interfacing!

The new Direct Data Access can:

- Append to a directory
- Create DVC tracked files from scratch
- Create a DVC directory from scratch
- Access Git and DVC tracked files
- List repository structure and metadata

Direct Data Access

The **perfect** data pipeline can seamlessly manage data and codefiles. So, let's abstract DVC interfacing!

The new Direct Data Access can:

- Append to a directory
- Create DVC tracked files from scratch
- Create a DVC directory from scratch
- Access Git and DVC tracked files
- List repository structure and metadata

Documentation dropping soon!

Use Cases

- Instantaneous Datasets!

Use Cases

- Instantaneous Datasets!
- Lazy Dataset Subsetting

Use Cases

- Instantaneous Datasets!
- Lazy Dataset Subsetting
- Support for ML on Edge Devices

Use Cases

- Instantaneous Datasets!
- Lazy Dataset Subsetting
- Support for ML on Edge Devices
- **Unlocking Active Learning**

Use Cases

- Instantaneous Datasets!
- Lazy Dataset Subsetting
- Support for ML on Edge Devices
- **Unlocking Active Learning**

(use cases proportional to imagination)

Thank you!

Have an awesome rest of your day!

Slides: `https://cs.purdue.edu/homes/jsetpal/dsc.pdf`