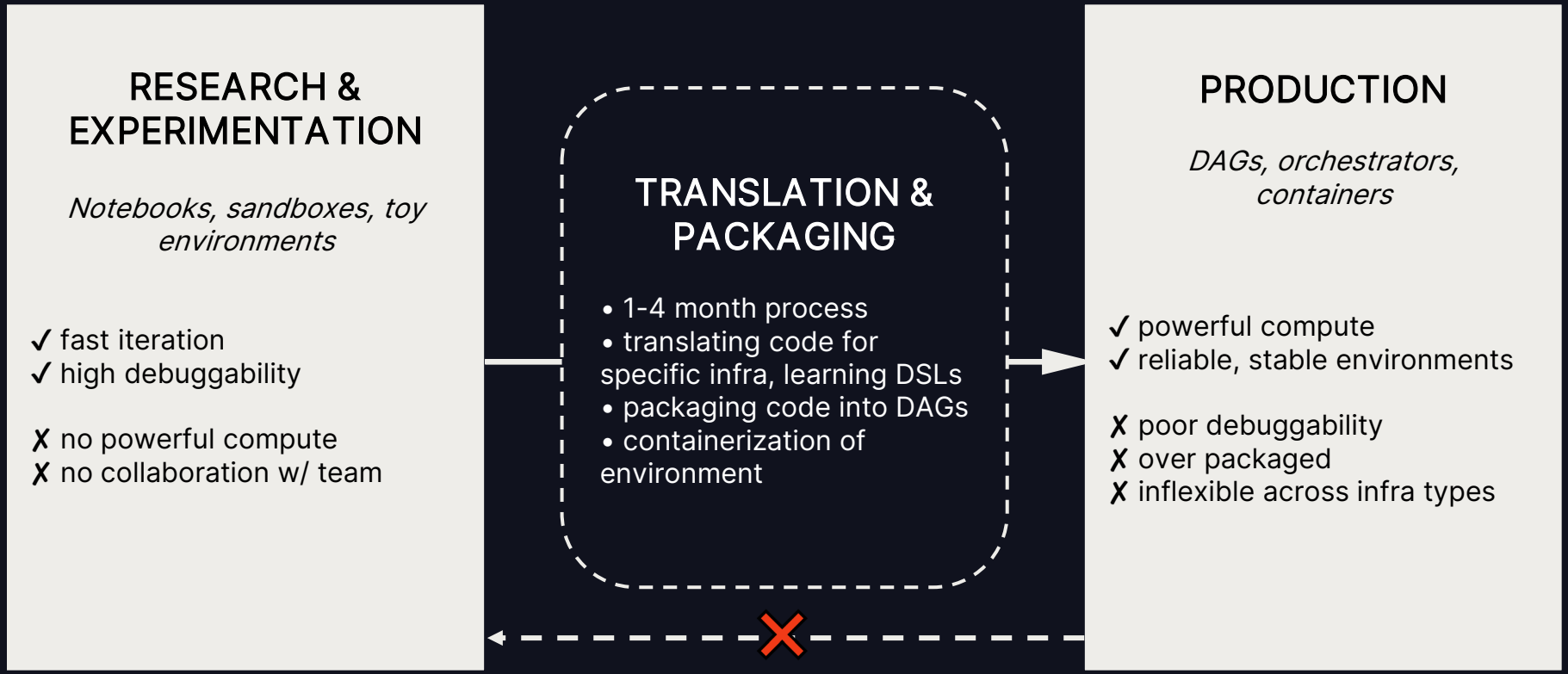
DATA+AI SUMMIT BY databricks

RUNHOUSE: A PYTORCH APPROACH TO ML INFRA

Rohin Bhasin & Caroline Chen
June 11, 2024

MODERN ML INFRA IS FRAGMENTED

Workflows are inflexible, duplicative, and unreproducible.



ISSUES STEM FROM FRAGMENTATION

No ML Flywheel

Slow iteration speed
Duplication everywhere

No Infrastructure Flexibility

Blocks scaling and cost optimization
Migrations lead to infra lock-in

No E2E Management & Visibility

Monitoring, control, and allocation infra specific and fragmented

WHAT ML DEVELOPMENT SHOULD LOOK LIKE

Infra Agnostic

- No migrations and DSLs
- Flexible to scaling & cost optimization

High Iteration Speed

- No excessive builds during dev work
- As smooth as developing locally

Central Control

- Single control plane for resource visibility and management
- Lineage tracking and governance

Multiplayer

- Reusable compute and services
- Reproducible behavior
- Shareable

A PYTORCH-LIKE SOLUTION

Runhouse is a Python native, infra-agnostic interface into your ML infrastructure.

PURE PYTHON RUNS ANYWHERE

Easy Setup

```
$ pip install runhouse
```

Runs As-is

No DSLs, configs, decorators, or CLI magic

Develop Anywhere

IDEs, Notebooks

Deploy Anywhere

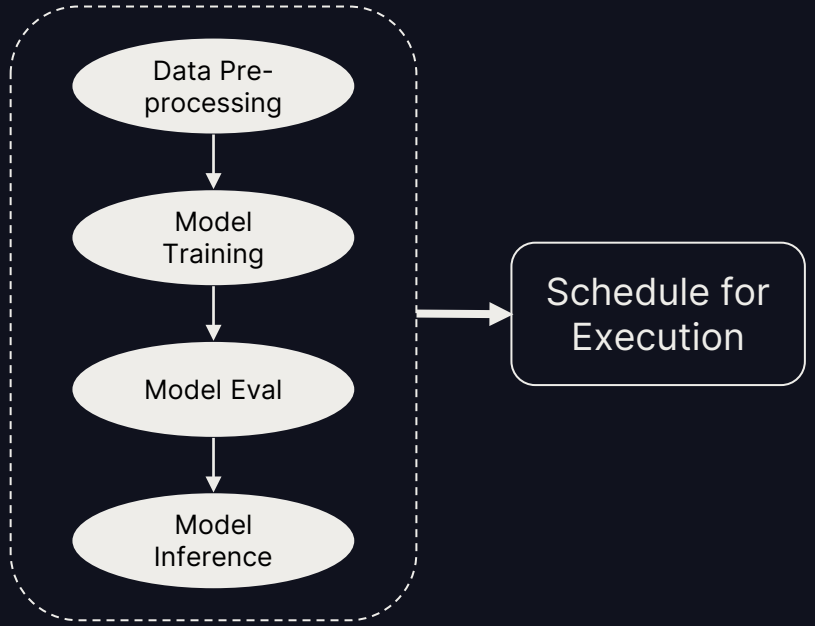
Orchestrators, containers, CI/CD

PYTORCH	RUNHOUSE
`my_model.to(cuda)`	`train_fn.to(my_gpu)`

EAGER EXECUTION FOR INFRA FLEXIBILITY

Traditional DAG

Rigid and siloed



Using Runhouse

Scale and cost-optimize

RESEARCH & EXPERIMENTATION

Notebooks, sandboxes, toy environments

✓ fast iteration
✓ high debuggability

✓ access to compute
✓ easy collaboration

W/ RUNHOUSE

- Code is identical
- Exactly reproducible
- Across teammates, across infrastructure
- Save time & cost

PRODUCTION

DAGs, orchestrators, containers

✓ powerful compute
✓ reliable, stable environments

✓ high debuggability
✓ code run as-is
✓ flexible across infra types

DEMO

“Training” on AWS EC2 via Runhouse.

API FUNDAMENTALS

An overview of Runhouse core components and APIs.

KEY COMPONENTS FOR REMOTE DEPLOYMENT & REPRODUCIBILITY

Compute

Your own infrastructure

```
cluster = rh.ondemand_cluster(
    name="rh-cluster",
    instance_type="A10G:4+",
    provider="aws"
)
```

Environment

Packages, setup, & secrets

```
env = rh.env(
    name="fn_env",
    reqs=["torch"],
    working_dir="./",
    env_vars={"USER": "*****"},
    secrets=["aws", "openai"],
    setup_cmds=[f"mkdir -p
~/results"]
)
```

Python Class / Function

Your existing code

```
def train(params, data):
            ...
class Embedder:
            ...

rh_train = rh.function(train)

RhEmbedder = rh.module(Embedder)
```

DEPLOY, SAVE, AND SHARE SERVICES

.to()

Send resources to any infrastructure and environment.

```
rh_train = rh_train.to(
        cluster, env)

rh_function(args)
```

.save()

Save resources and metadata, to be reused later on.

```
env.save()

cluster.save()

rh_function.save()
```

.share()

Share resources with your teammates, or even publish to the public.

```
cluster.share(
   "team@run.house")

rh_function.share(
   visibility="public")
```

LOAD AND REUSE THE SERVICE HASSLE-FREE

Reuse saved function

```
remote_fn =
     rh.function("fn_name")

result = remote_fn()
```

Use shared function

```
remote_fn =
     rh.function("/rohin/fn_name")

result = remote_fn()
```

DEN: A COLLABORATION LAYER FOR INFRA

Search & Discovery

Search through applications
View details all in one place

Sharing & Collaboration

Share with teammates
Iterate on shared services

Lineage & Versioning

Provenance for executed code
Keep track of previous states

Auth & Governance

Fine-grained access to compute and services
Comprehensive usage tracking

SOON: INFRA MANAGEMENT & VISIBILITY

Cost Optimization

- Compute types, clouds, regions
- Global pool of compute
- Scheduling & autostop
- Servicification of repeated work

Unified Visibility

- Live compute tracking
- Resource utilization
- Usage inspection

TRY IT OUT :)

```
$ pip install runhouse

  run.house/examples
```

rohin@run.house caroline@run.house