

OSS Stack for ML

Soma S. Dhavala

01-01-2025



[@TheSaddlePoint](https://www.youtube.com/@TheSaddlePoint)



github.com/mlsquare



[linkedin.com/in/somasdhavala](https://www.linkedin.com/in/somasdhavala)



[@TheSaddlePoint](https://twitter.com/TheSaddlePoint)

Agenda

What should an OSS Stack to build ML look like?

Part-0: Framing MLOps

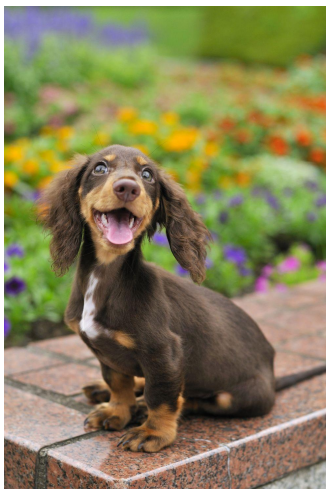
Part-1: OSS for ML Engineering

Part-2: OSS for ML Science

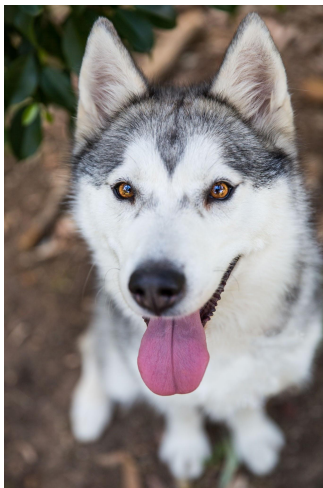
Part-3: OSS model for ML Development

Stakeholder Expectations

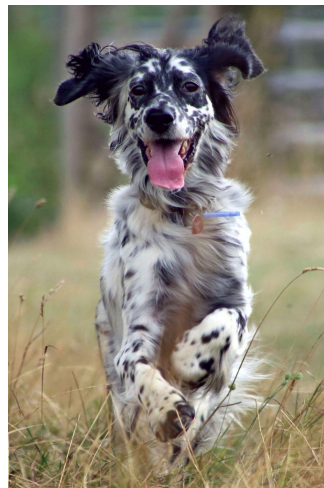
ML team
highest accuracy



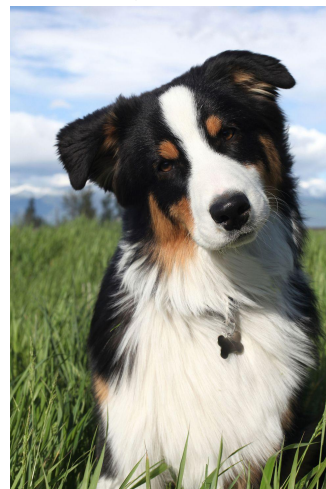
Sales
sells more ads



Product
fastest inference



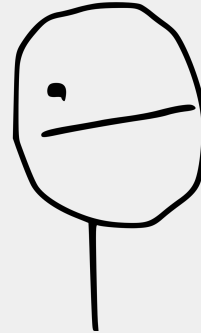
Manager
maximizes profit
= laying off ML teams



Expectation

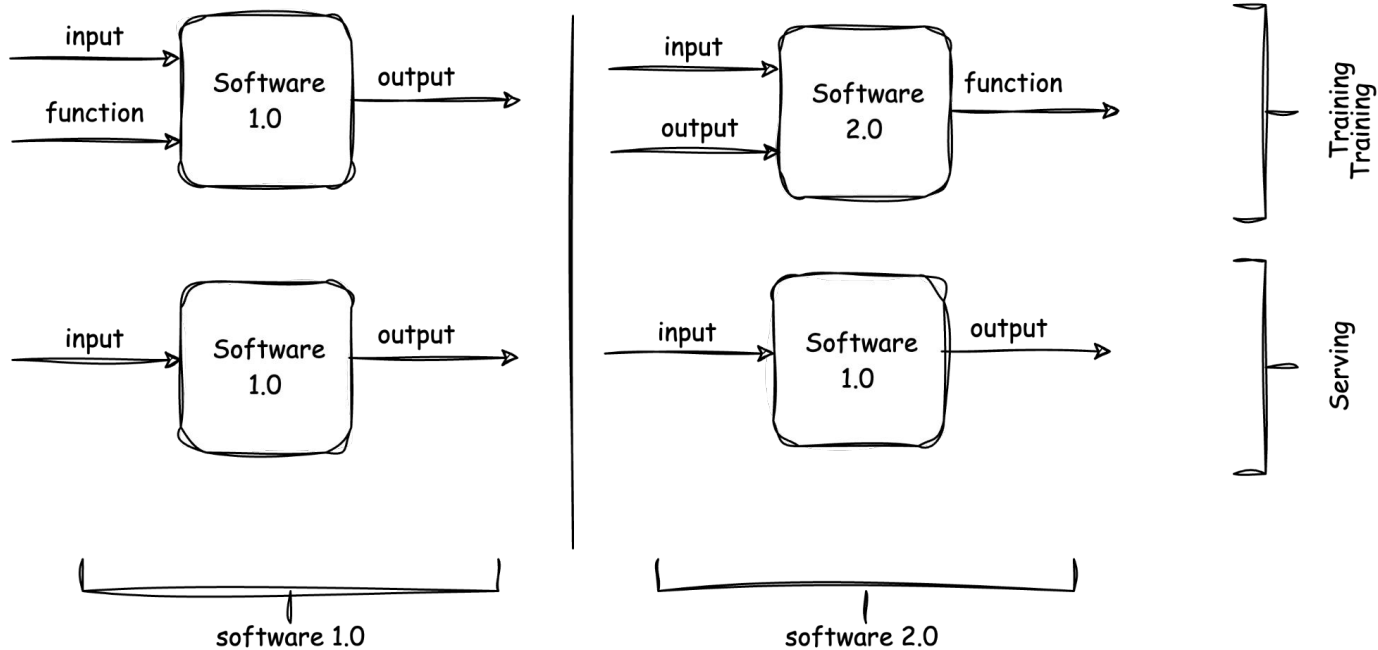


Reality



But why?

Software 1.0 vs Software 2.0



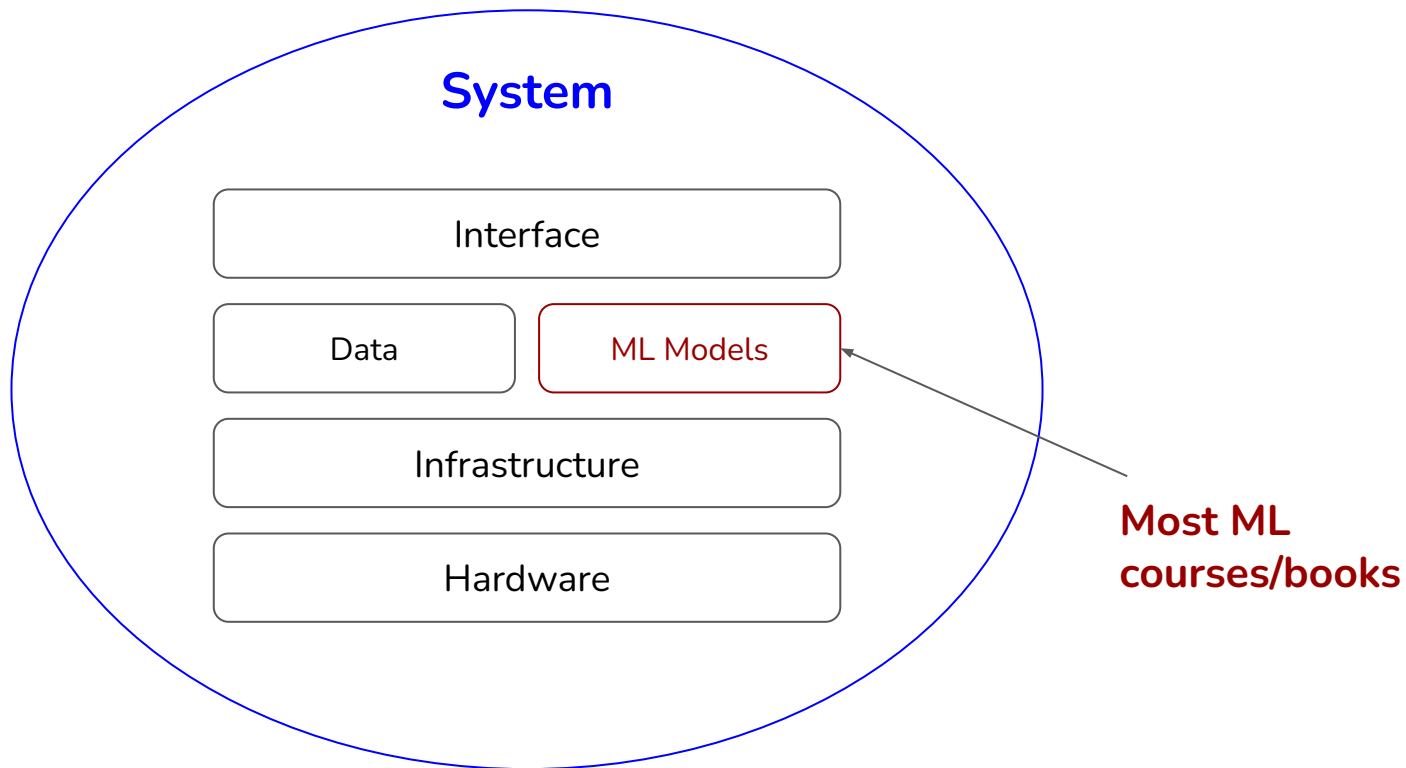
ML as a software is a *fundamentally* different beast.

But why?

	Software 1.0	Software 2.0
Codified in	Formal Language	Weights & Biases (parameters)
Developed by	Programming	Training
Specification	PRD/SRD	Data
Behaviour	Deterministic	Stochastic
	Provably correct	Provably wrong
	Debuggable	Hard to Debug
	Verifiable	Hard to verify
	Explainable	Hard to explain
	Fixable	Hard to fix
	Idempotent	Hard to reproduce

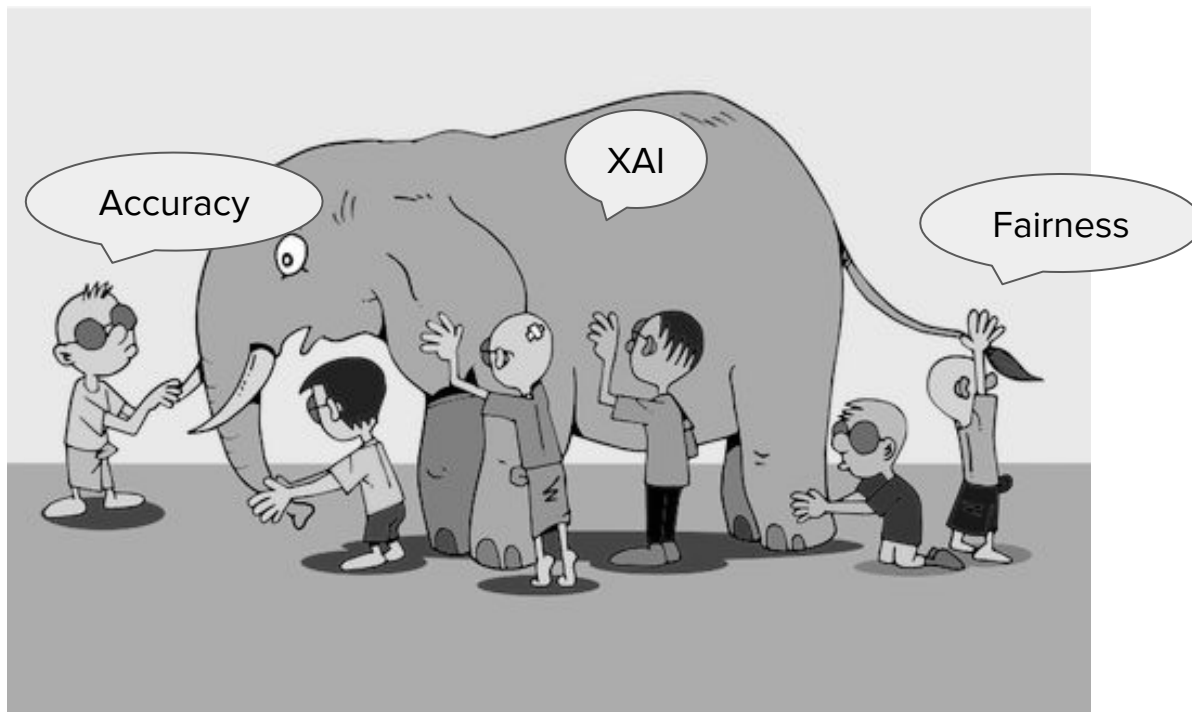
Determinism and Control (of the build process and products) - we take them for granted. But no longer.

But why?



Think systems, not models. Think modeling, not models.
Combating this intellectual inertia is hard.

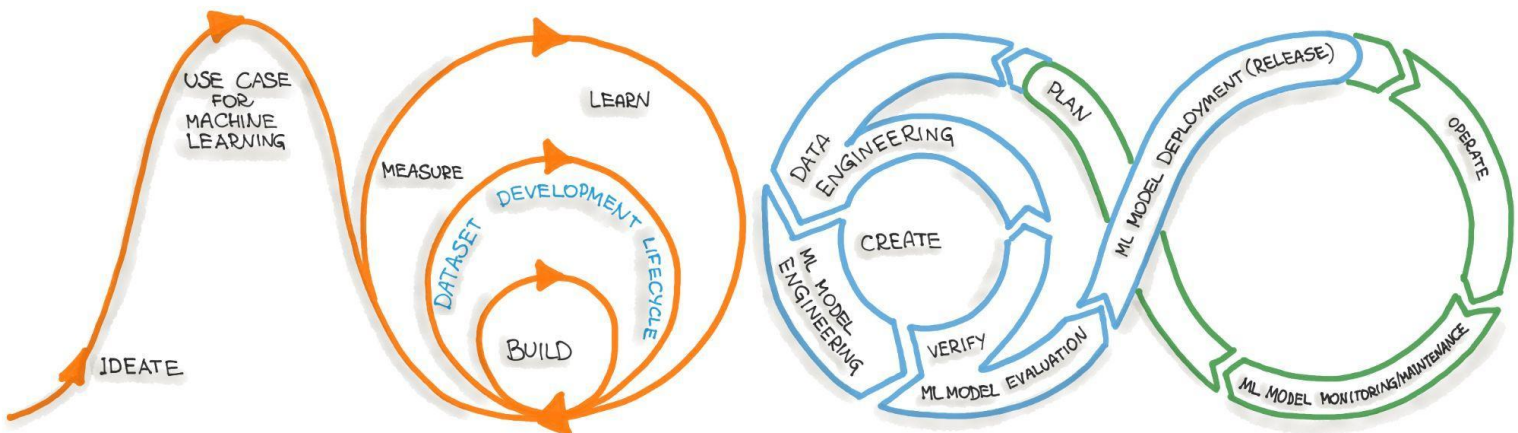
But why?



Semi-orthogonal sub-fields.

But why?

CRISP-ML(Q)



PHASES

BUSINESS & DATA UNDERSTANDING

MODEL DEVELOPMENT

MODEL OPERATIONS

@visonger

anything changes > everything changes
modeling is highly iterative and nonlinear

source: ml-ops.org

Non-deterministic systems

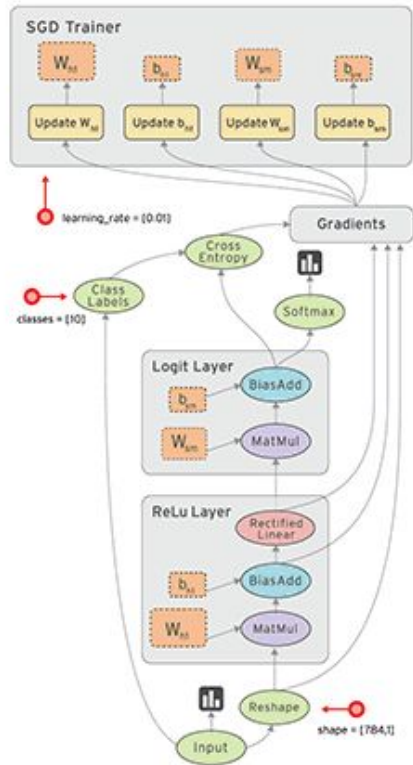
Build must be agile and address ambiguity

Design must address “reliability” in the presence of “uncertainty”

IF
 $CS = DS + A$
THEN
 $ML = ?$

need an abstraction for Models and Modeling

“ML modeling” is a DAG



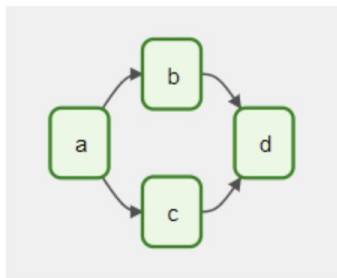
TensorFlow: Data Flow Model to writing programs

Home / Concepts / DAGs

DAGs

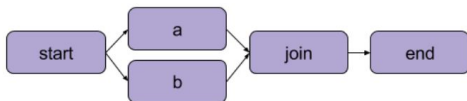
A *DAG* (Directed Acyclic Graph) is the core concept of Airflow, collecting **Tasks** together, organized with dependencies and relationships to say how they should run.

Here's a basic example DAG:



Branch

You can express parallel steps with a **branch**. In the figure below, `start` transitions to two parallel steps, `a` and `b`. Any number of parallel steps are allowed. A benefit of a branch like this is performance: Metaflow can execute `a` and `b` over multiple CPU cores or over multiple instances in the cloud.



```
from metaflow import FlowSpec, step

class BranchFlow(FlowSpec):

    @step
    def start(self):
        self.next(self.a, self.b)

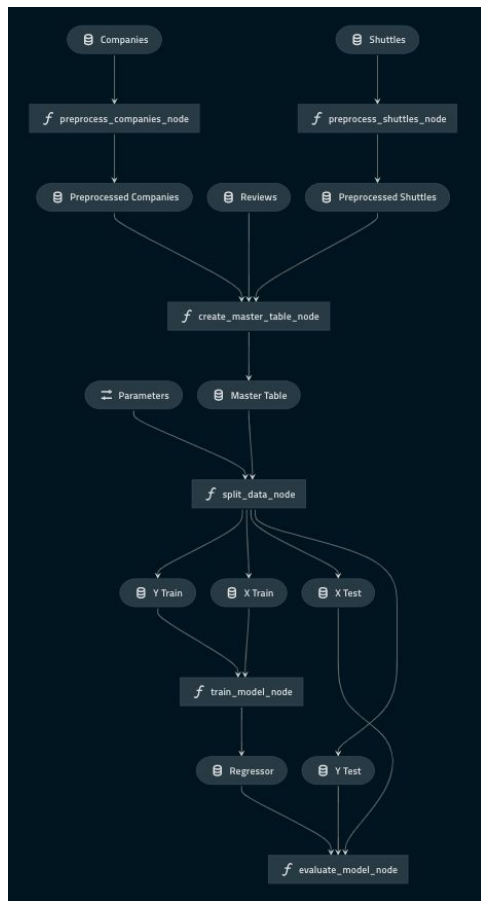
    @step
    def a(self):
        self.x = 1
        self.next(self.join)

    @step
    def b(self):
        self.x = 2
        self.next(self.join)

    @step
    def join(self, inputs):
        print('a is %s' % inputs.a.x)
        print('b is %s' % inputs.b.x)
        print('total is %d' % sum(input.x for input in inputs))
        self.next(self.end)

    @step
    def end(self):
        pass

if __name__ == '__main__':
    BranchFlow()
```



Kedro is a toolbox for production-ready data science. It uses software engineering best practices to help you create data engineering and data science pipelines that are reproducible, maintainable, and modular. You can find out more at kedro.org.

Kedro is an open-source Python framework hosted by the [LF AI & Data Foundation](https://lfaidata.foundation/).

Kedro: build ML with best software engineering practices

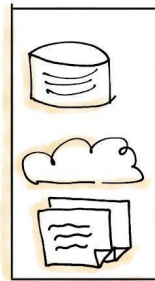
source: [kedro](https://kedro.org)

OSS for ML Engineering

But why?

MACHINE LEARNING ENGINEERING

DATA PIPELINE



EXPLORATION & VALIDATION

WRANGLING (CLEANING)

DATA

TRAIN

TEST

MONITORING & LOGGING

MACHINE LEARNING PIPELINE

MODEL ENGINEERING

MODEL EVALUATION

MODEL PACKAGING

MODEL

INOQ

SOFTWARE CODE PIPELINE

CODE

BUILD & INTEGRATION TESTING

DEPLOYMENT
DEV
↓
PRODUCTION

- Profiling
- "JUnit4Data"

• Data versioning

FEEDBACK
new data from model performance

- Model decay trigger

- Feature engineering
- Hyperparameters tuning

- Best model selection
- Model performance metrics
 - accuracy
 - precision
 - recall
- F₁

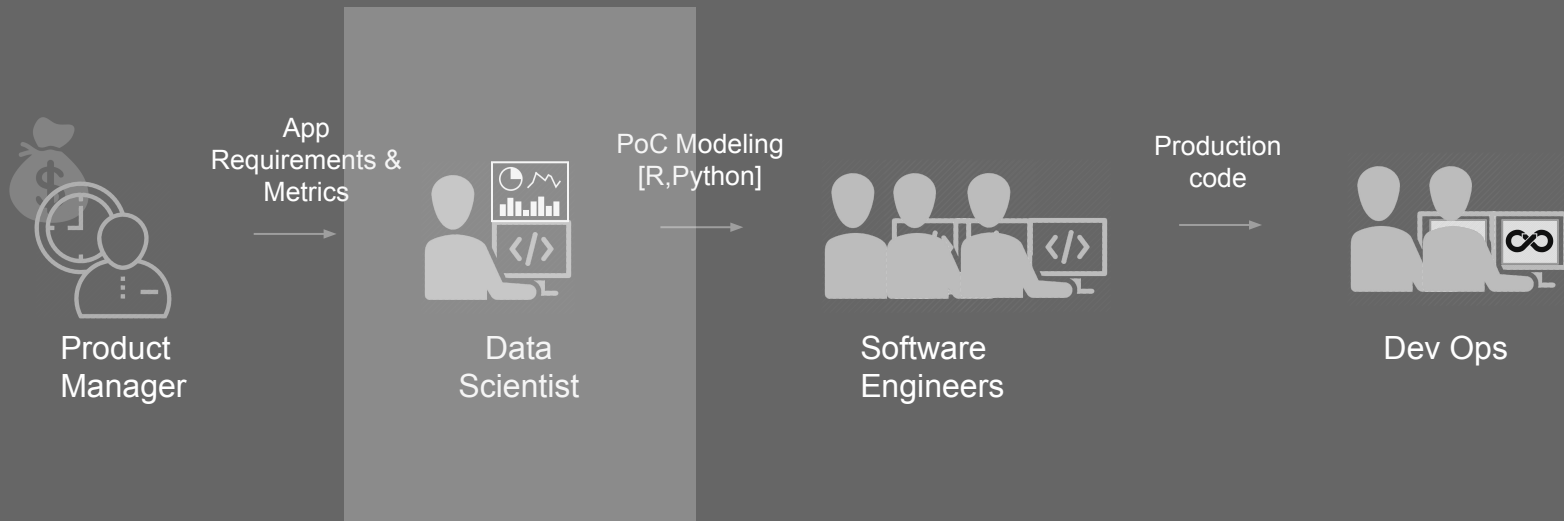
- Model format
 - ONNX
 - JAR
 - PKI

- Model serving
 - service
 - Docker
 - K8S

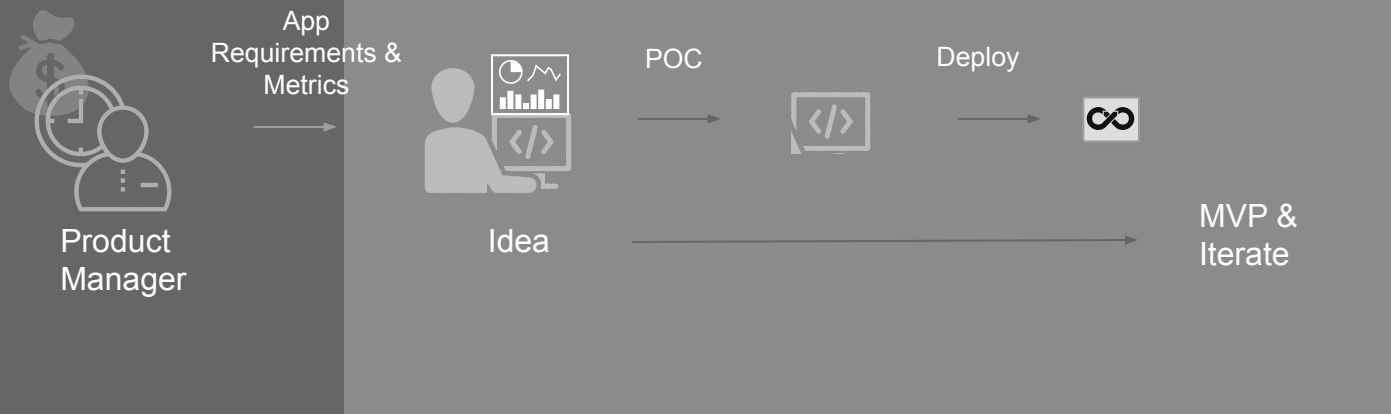
- Trunk based dev.
- Code versioning

1. Data Engineering
2. Model Engineering
3. Model Deployment

(Lot of) Hidden Tech Debt



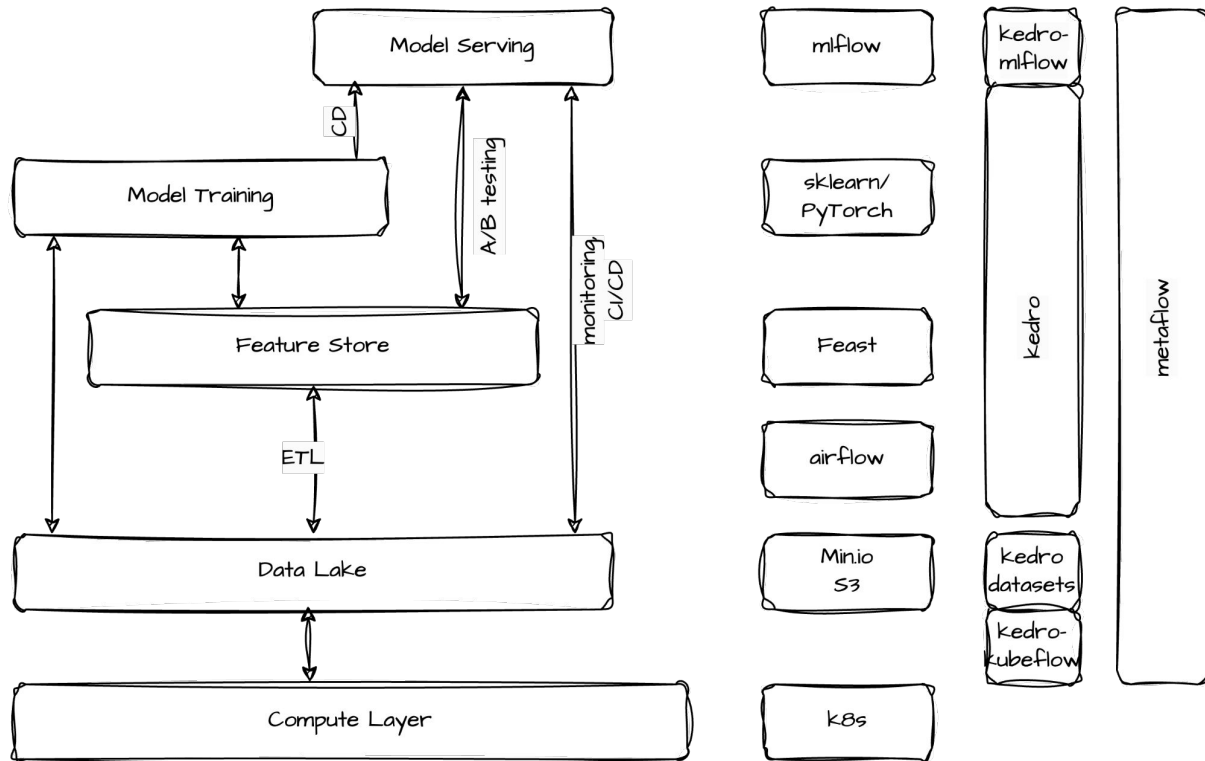
- Tech stacks could be different.
- Dev and Prod env could be different.
- Scalability could be an issue!
- Most importantly, skill sets could be different



Automate and Delegate the responsibility to Tools

Layer	Promise	Task	Tool
Solution	Reproducibility	Versioning - Data, Code, Model	DVC , Kedro , mlflow , MetaFlow
	Observability	Logging Monitoring Tracking	Kedro/mlflow , MetaFlow evidently WandB
	Agility	EDA Experiment Tracking	Notebook environment (Kedro , MetaFlow) mlflow , Weights and Biases
	CI/CD	Integrate Deploy Serve	git-actions docker/ Kedro/ mlflow FastAPI
Docs	Truthful/ Up to date	Code Data Solution	quarto/ sphinx
Code	Testability Readability	Linting Testing Documentation	ruff/ black pytest quarto/ sphinx
Data	Veracity	Drift Detection and Schema Validation	pymfe/ pydantic/ evidently/ greatexpectations
		ETL Feature Stores	airflow feast
	Agility Auditability	Data Lake	minio/ s3 + athena Kedro Data Catalogues
Compute	Scalability	Elastic Compute	k8s

Quite a number of OSS tools available to address specific gaps, with overlaps

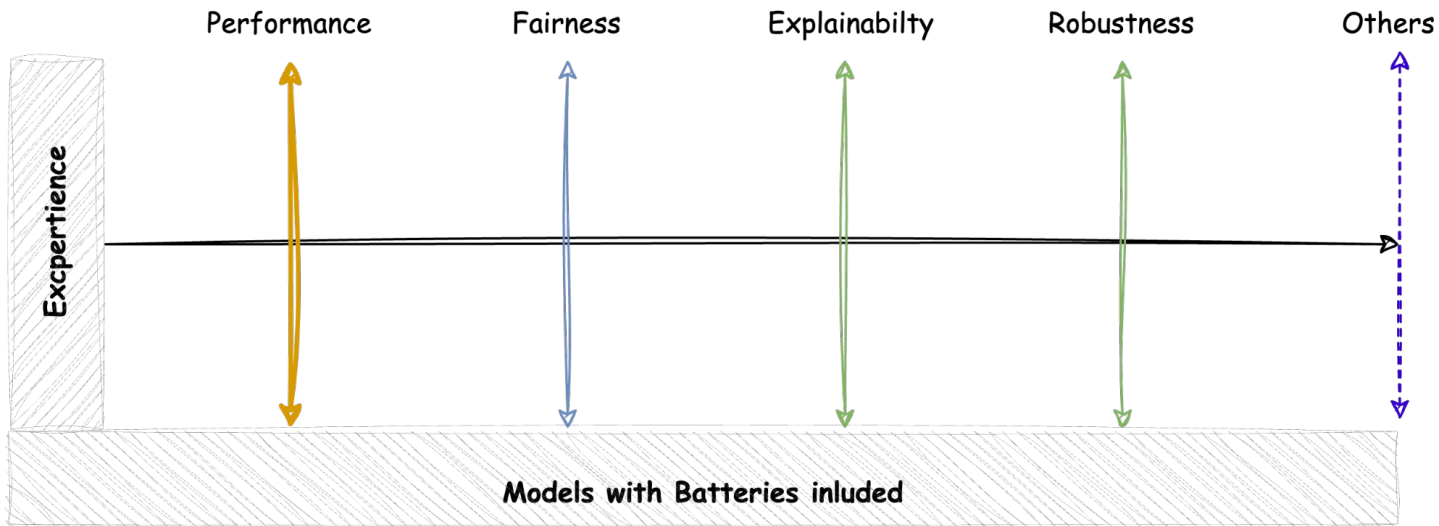


ML POD: ML Engineering







An (opinionated) [git template](#) with integrations enabled

OSS for ML Science

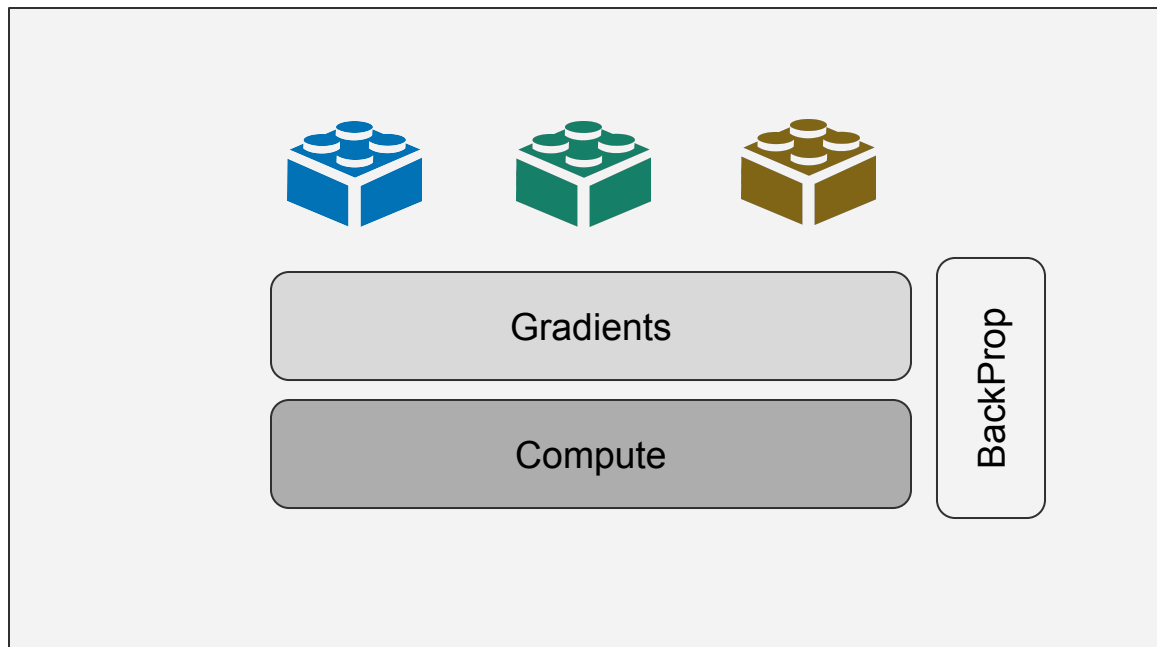
Performance-centric ML development



But what is an enabling technology that addresses these sub-orthogonal fields?

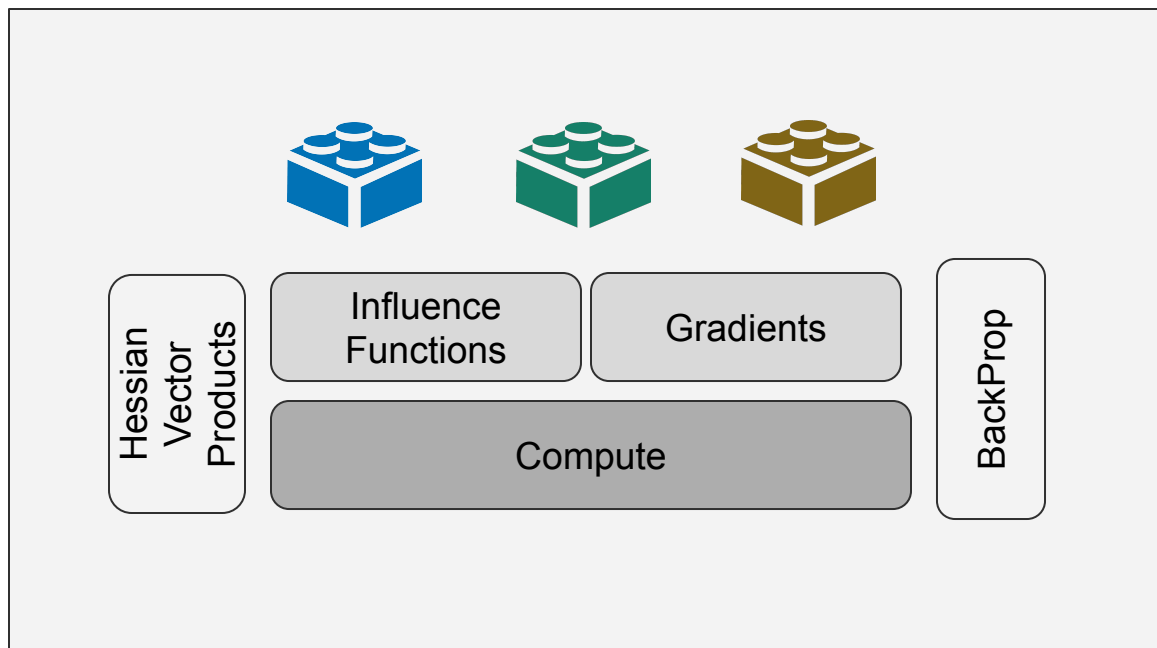
inputs	layers	losses	optimizers	runtime	outputs
					
Text Speech Image Numerical Fixed Variable Single Many	Dense Conv GRU Attention Merge LDA/QDA DT FM Kernels	CE MSE MAE .. CheckLoss Hinge Ordinal	Adam RMSProp NeverGrad ... PSO	cpu gpu tpu	Text Speech Image Numerical al Fixed Variable e Single Many

Deep Learning as a technology breaks monolithic scientific computing paradigm.
It is OOPs for scientific computing at large.



ML Engineering Backbone

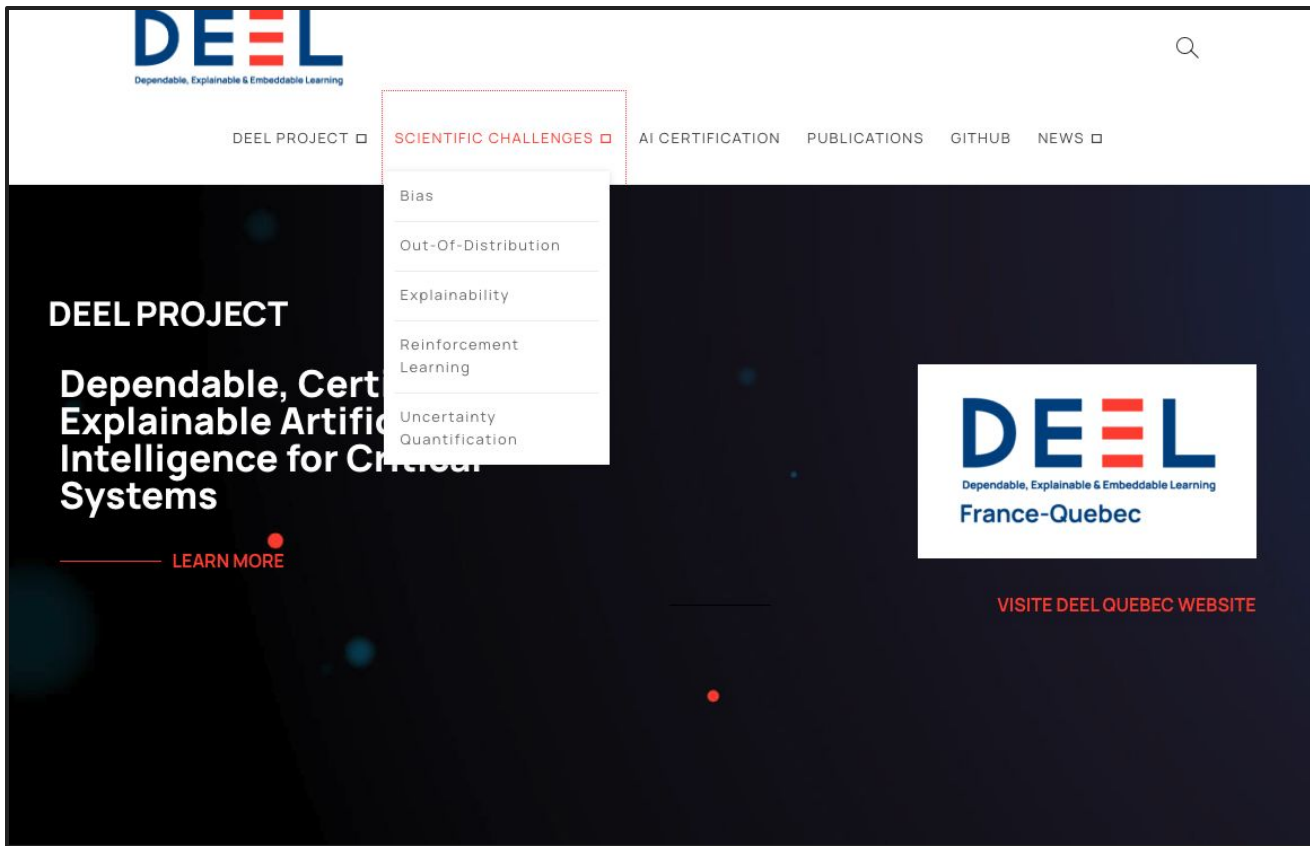
Scalable computation of gradients is at the heart of modern Deep Learning Engineering



ML Science Backbone

Perhaps, the answer is in IHVP

IVHF: [pyDVF](#)
 IFs: [influentiate](#):
 For more see: AI-839@IIIT-B



The image shows a screenshot of the DEEL website homepage. At the top left is the DEEL logo with the tagline "Dependable, Explainable & Embeddable Learning". A search icon is in the top right. The navigation menu includes "DEEL PROJECT", "SCIENTIFIC CHALLENGES", "AI CERTIFICATION", "PUBLICATIONS", "GITHUB", and "NEWS". The "SCIENTIFIC CHALLENGES" dropdown menu is open, listing: Bias, Out-Of-Distribution, Explainability, Reinforcement Learning, and Uncertainty Quantification. The main content area features a dark blue background with the text "DEEL PROJECT Dependable, Cert Explainable Artificial Intelligence for Critical Systems" and a "LEARN MORE" button. On the right, there is a white box with the DEEL logo and "France-Quebec" text, and a "VISITE DEEL QUEBEC WEBSITE" button.

for more see: [Project DEEL](#)

Promise	Methods	Tool
Statistical Reliability	Conformal Prediction Techniques Learn-Then-Test Framework	torchcp , mapie , puncc
OOD	RMD, Entropy, Clustering	pytorch-ood , oodeel
Security	Adversarial Attacks	foolbox
Explainability	Integrated Gradients SHAPLEY values, LIME, Decision Sets, Influence Functions Counterfactuals	shap , xplique , dice
Bias and Fairness	Metrics (for detection) and Methods (to mitigate) Influence Functions	AI Fairness 360 , influential
Robustness	Adversarial Training Robust Losses Regulaization	All of above

OOD, Robustness, Security, XIA, Bias & Fairness – share common principles!

Expectation



Reality



Make ML great again :)