AI Systems Engineering

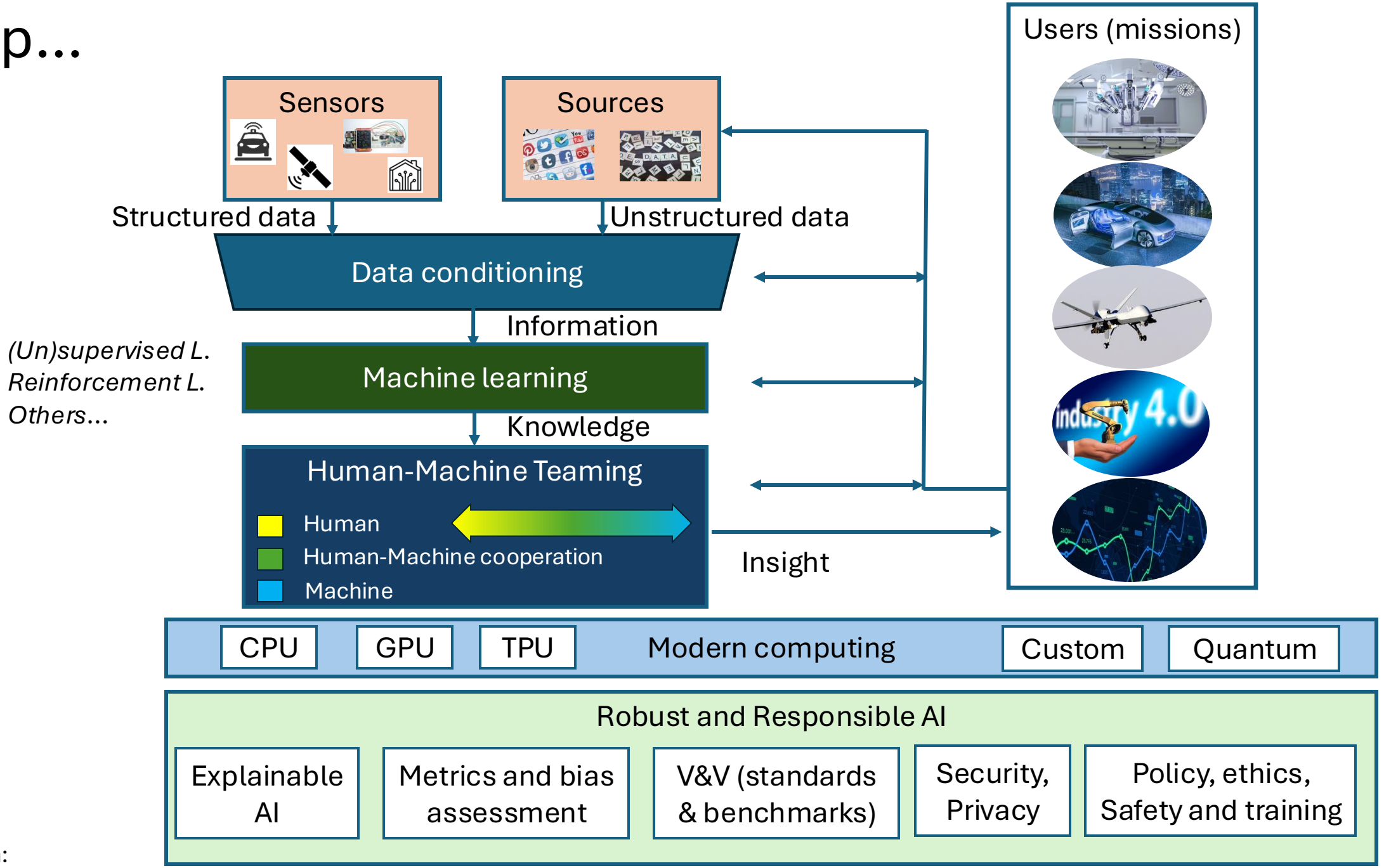
Prof. Roberto Pietrantuono

Lecture 4

Outline - Building a Model Factory

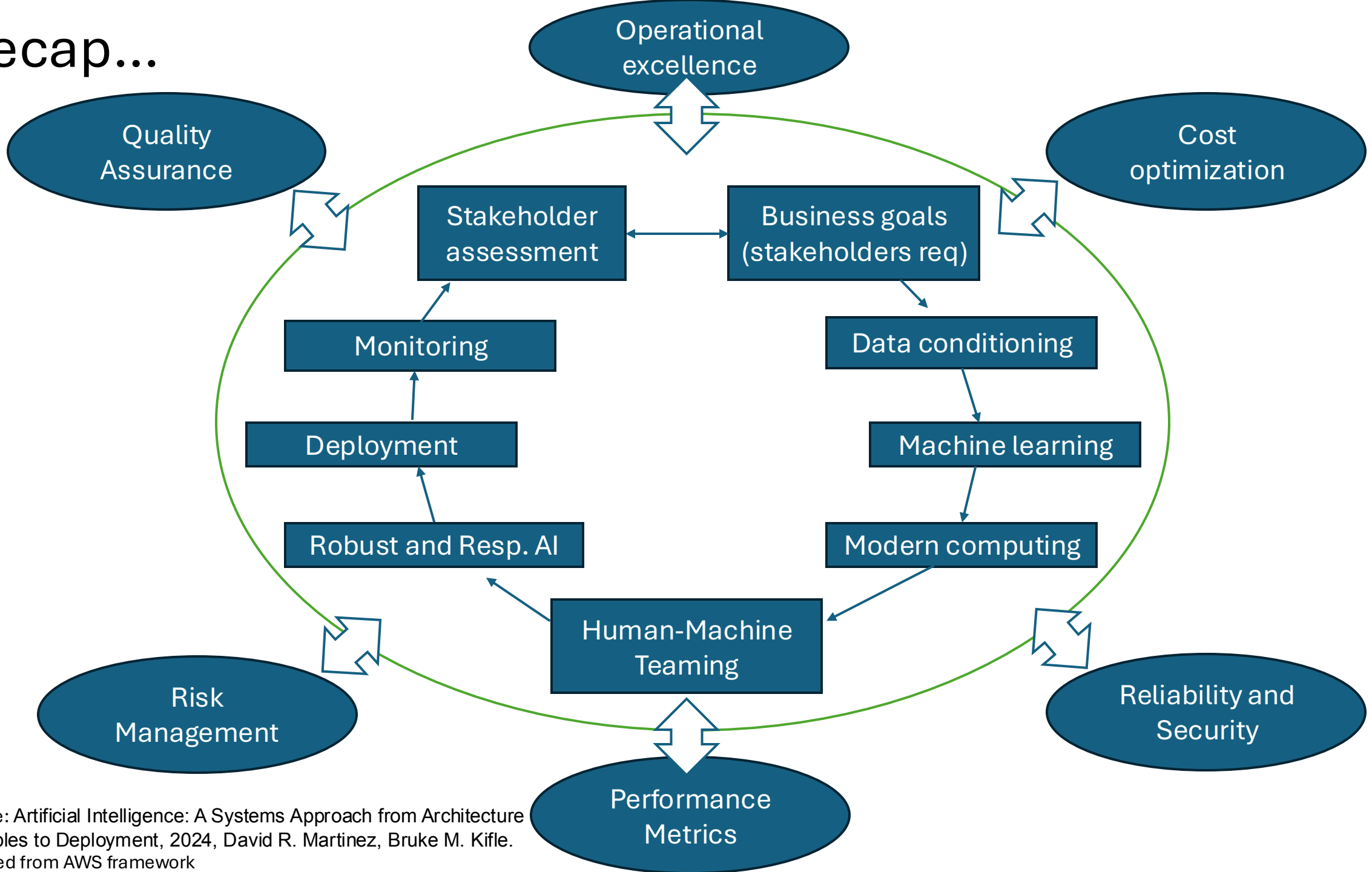
- Model Training
- Features Engineering
- Designing Training System
  - Training and Prediction Patterns
- Drift Detection
- Monitoring
- Automated Training
  - Optimization
  - AutoML
- Model Persistence
- Pipelining

Recap...



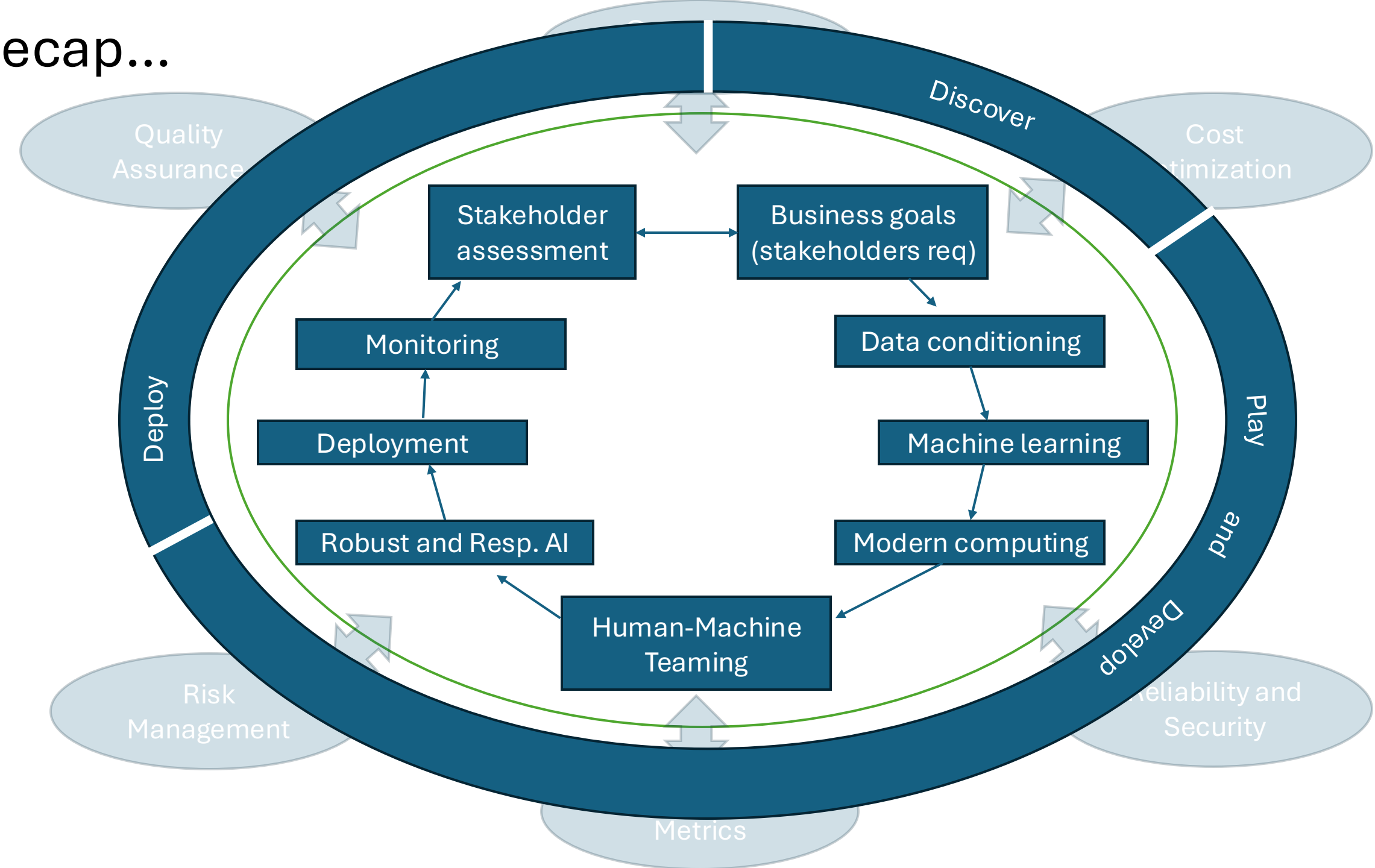
Adapted from:
Artificial Intelligence: A Systems Approach from Architecture Principles to Deployment, 2024, David R. Martinez, Bruke M. Kifle.

Recap...



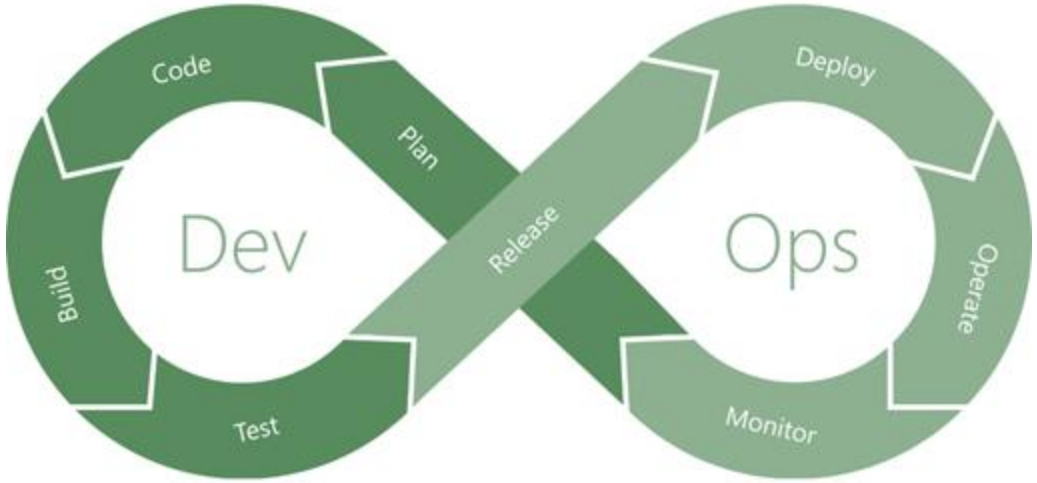
Source: Artificial Intelligence: A Systems Approach from Architecture Principles to Deployment, 2024, David R. Martinez, Bruke M. Kifle.
Adapted from AWS framework

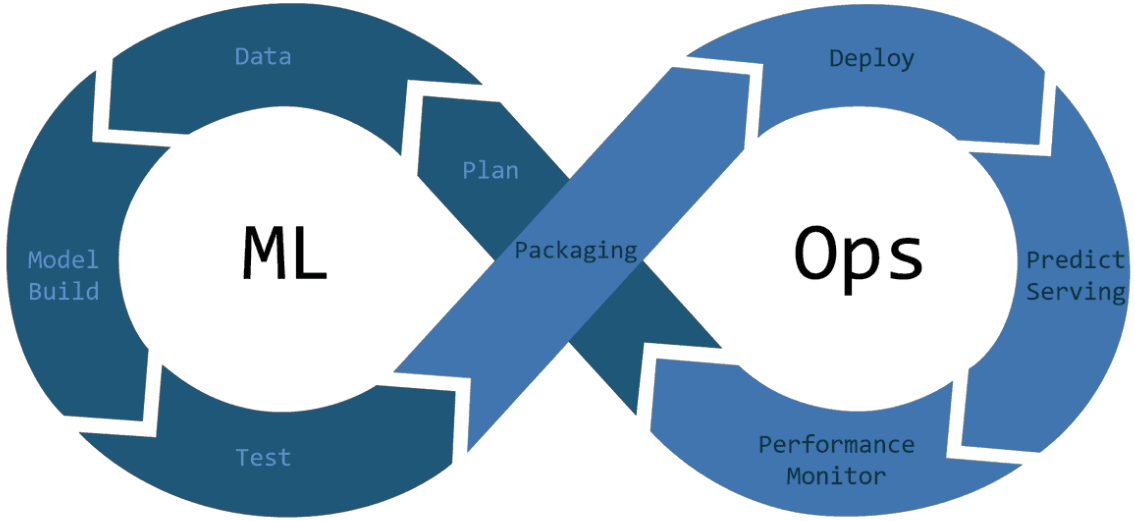
Recap…


Discover
Play
and
Develop
Deploy
Stakeholder assessment
Business goals (stakeholders req)
Data conditioning
Machine learning
Modern computing
Human-Machine Teaming
Robust and Resp. AI
Deployment
Monitoring
Quality Assurance
Cost
timization
Risk Management
eliability and Security
Metrics

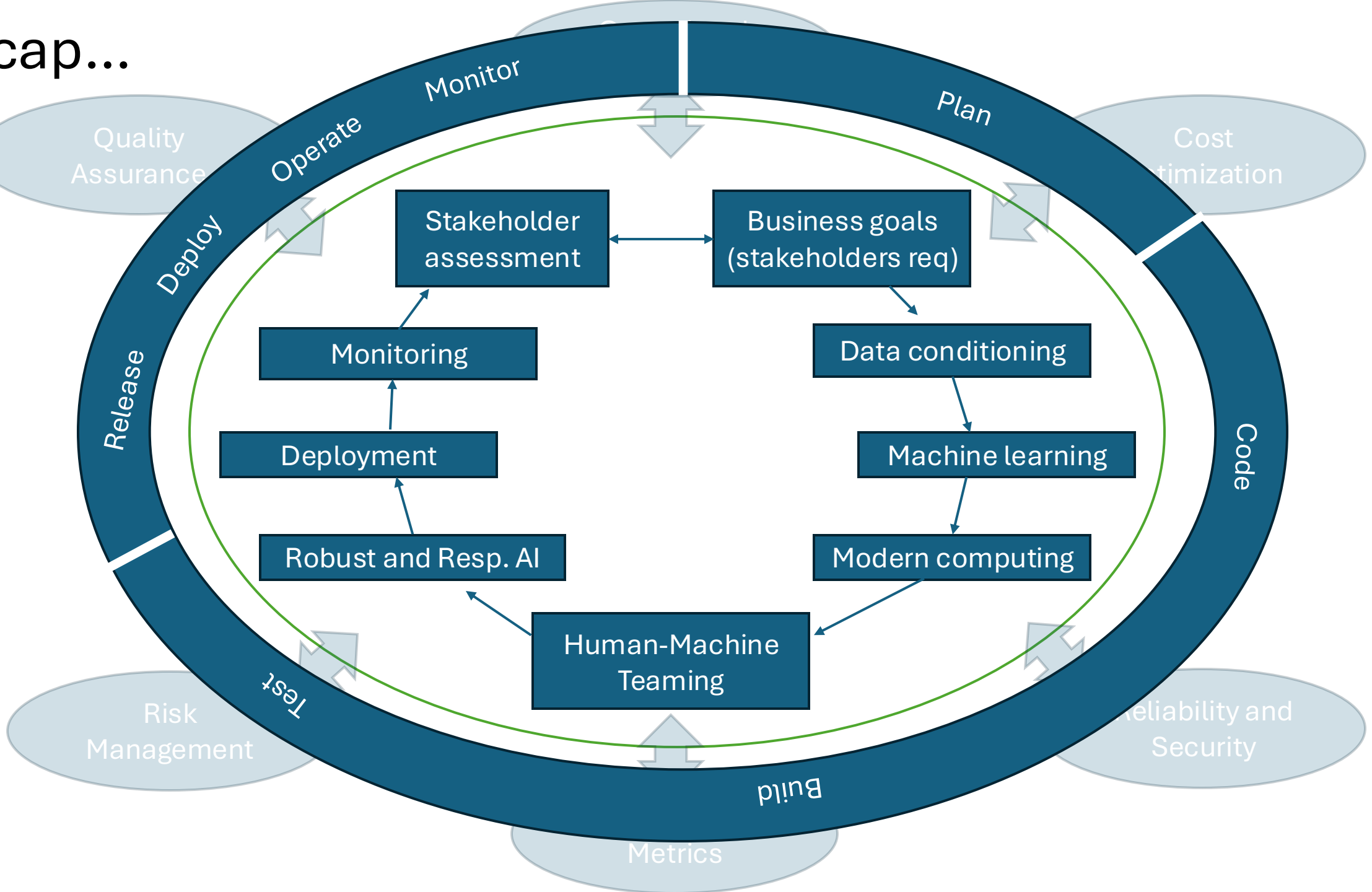
Recap…

Still on “how”: An ML perspective





Recap...


Monitor
Plan
Operate
Code
Deploy
Build
Release
Test
Quality Assurance
Cost Optimization
Risk Management
Reliability and Security
Metrics
Stakeholder assessment
Business goals (stakeholders req)
Monitoring
Data conditioning
Deployment
Machine learning
Robust and Resp. AI
Modern computing
Human-Machine Teaming

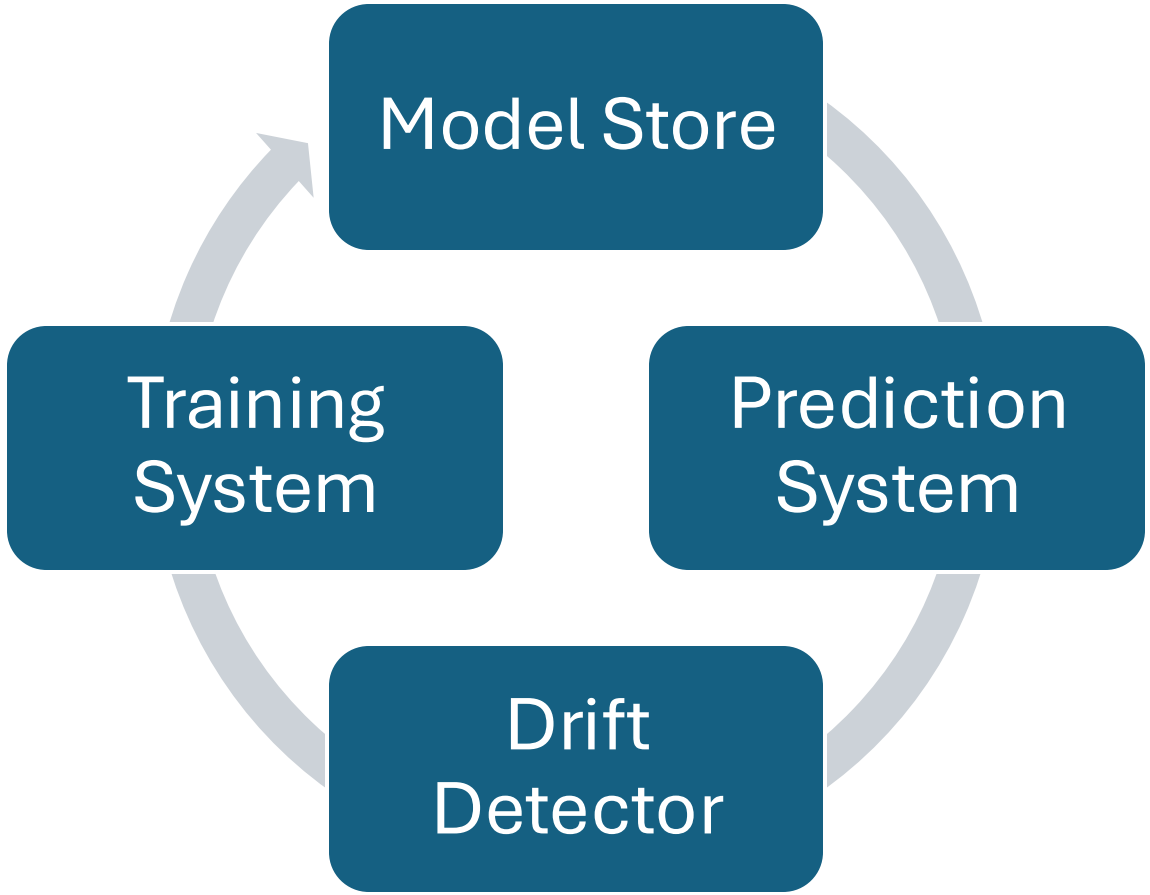
Building a Model Factory: Technical requirements

- Anaconda, IDE/Python Notebooks

```
git clone https://github.com/rpietrantuono/AISE_Ch3.git
conda env create –f mlewp-chapter03.yml
```

If doesn't work , rename mlewp-chapter03.yml as environment.yml and call «conda env create»
Also, you might need to install tensorflow manually on MacOS via
`pip install tensorflow-macos` and
`pip install tensorflow-metal`

Model Factory



Model Training - optimizations

- **Loss Functions**
- E.g. for regression:
  - Mean Squared Error / L2 loss
  - Mean Absolute Error / L1 loss
- E.g. for binary classification:
  - Log loss/Logistic loss/Cross-entropy loss
  - Hinge loss
- E.g. for multi-class classification:
  - Multi-class across entropy loss
  - Kullback Leibler divergence loss

- **Cutting your loss**
- Gradient descent
- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent
- With adaptive learning rate:
  - AdaGrad
  - AdaDelta
  - RMSprop
  - Adam

Model Training – optimizations – NN Example

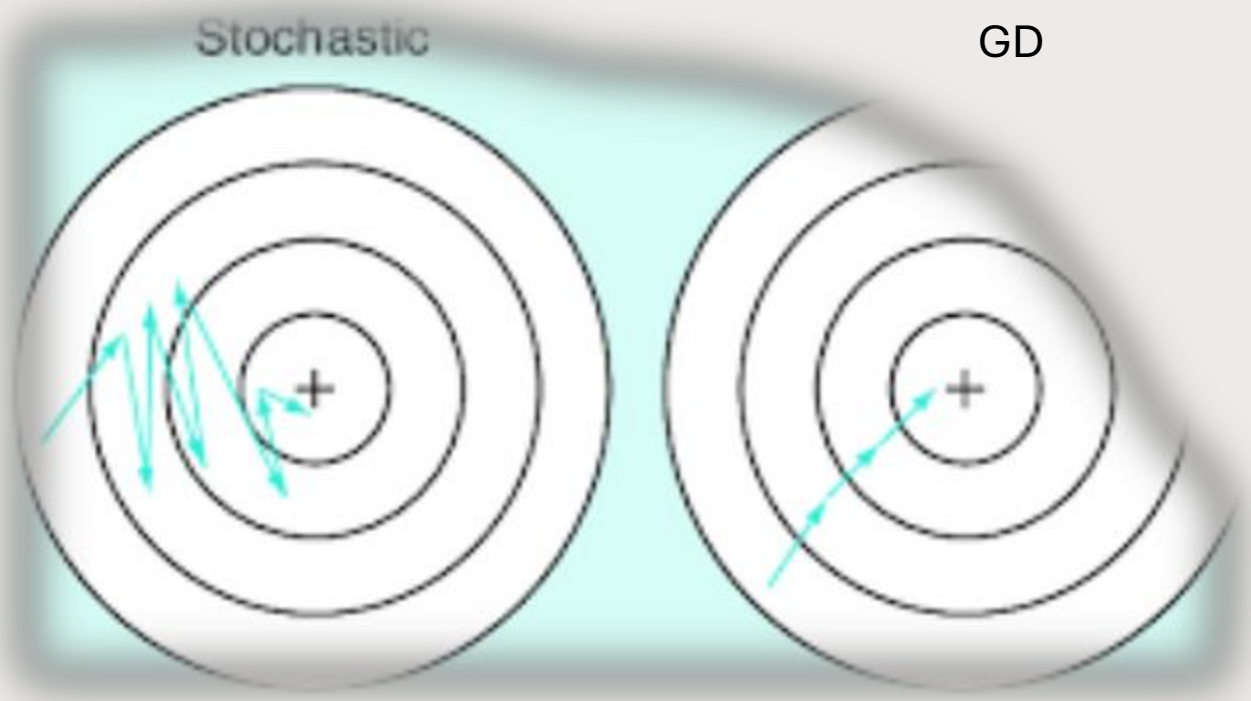
a(L) activation of last Layer L
z(L) = w(L) a(L-1) + b
a(L) = Fa(z(L)) (Fa is the activation function)
C = Fc(a(L), y) (Loss/Cost, to be minimized)
Question: how sensitive is C to the change of w (and similarly to b and a(L-1))

$$\frac{dC}{dw^{(L)}} = \frac{dC}{da^{(L)}} \frac{da^{(L)}}{dz^{(L)}} \frac{dz^{(L)}}{dw^{(L)}}$$

And iteratively backpropagating…

$$\frac{dC}{dw^{(L-1)}} = \frac{dC}{da^{(L)}} \frac{da^{(L)}}{dz^{(L)}} \frac{dz^{(L)}}{dw^{(L)}} \frac{dw^{(L)}}{da^{(L-1)}} \frac{da^{(L-1)}}{dz^{(L-1)}} \frac{dz^{(L-1)}}{dw^{(L-1)}}$$

Model Training – optimizations – NN Example



…but we just did for one example

The full cost is the average of all individuals

A GD step would require the derivative for all *n* training examples

$$\frac{dC}{dw} = \frac{1}{n}\sum_{k=1}^{n}\frac{dC_k}{dw_k}. \qquad \frac{dC}{db} = \frac{1}{n}\sum_{k=1}^{n}\frac{dC_k}{db_k}$$

Given the cost, we use random mini-batches (e.g., 16, 32) – Stochastic GD (SGD)

Repeatedly going through mini-batches ensures to coverge to the global minimum

Model Training – optimizations

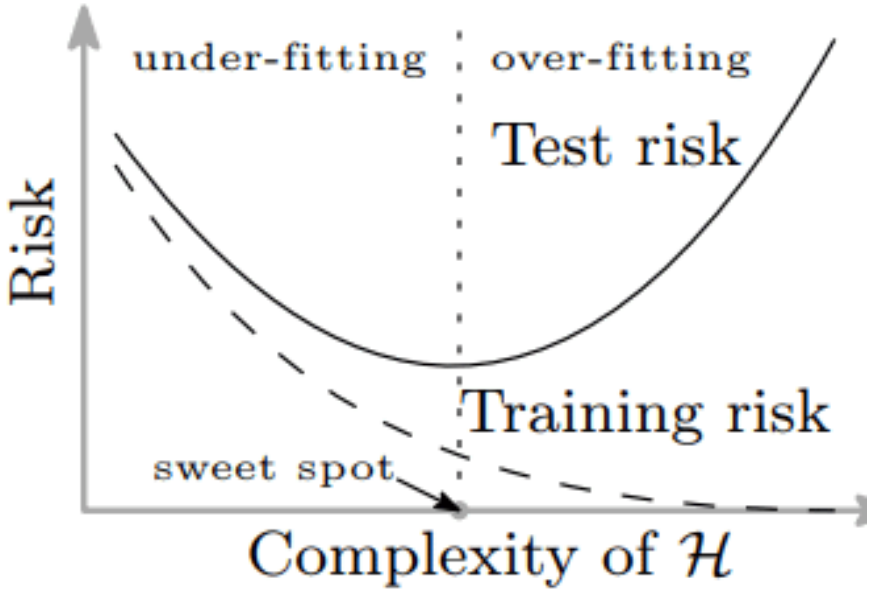
Many other choices

- Number of layers
- Number of Neurons
- Activation Functions
- Architecture (e.g., Convolutional, Recurrent, …)
- Hyperparameters (batch size, learning rate, …)

- More on this at the end of the lecture

Features Engineering

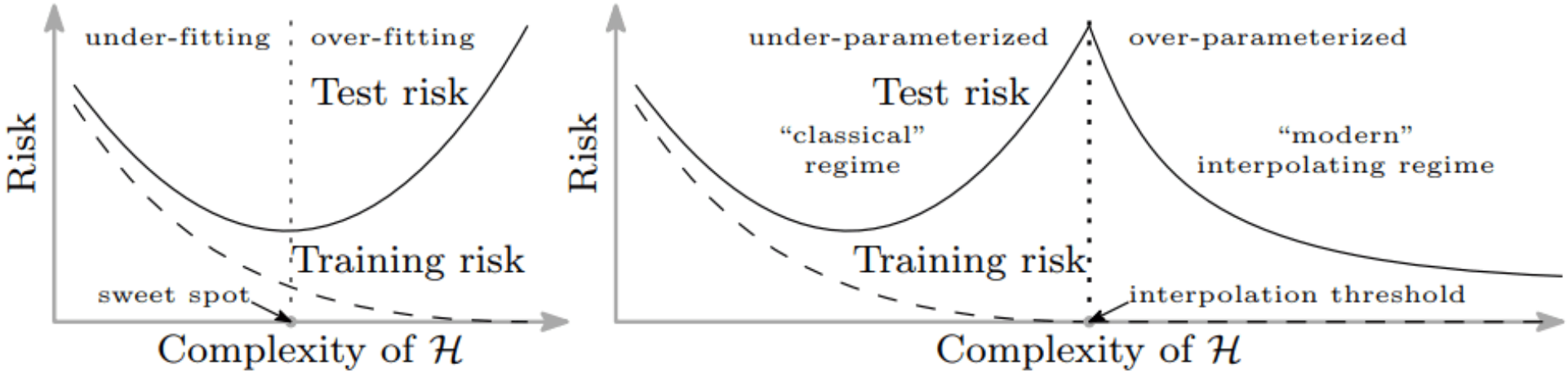
- We also need to make sure we only work on the data we deem useful for improving the performance of the model, as it is far too easy to explode the number of features and fall victim to the ***curse of dimensionality*.**
  - In high-dimensional problems, data becomes increasingly sparse in the feature space, so achieving statistical significance can require exponentially more data



(a) U-shaped "bias-variance" risk curve

Features Engineering

- ...While considering the double descent phenomenon



(a) U-shaped "bias-variance" risk curve

(b) "double descent" risk curve

Features Engineering : Categorical Features

```
from sklearn import preprocessing

data = [['Bleach'], ['Cereal'], ['Toiled Roll']]


ordinal_enc = preprocessing.OrdinalEncoder()
ordinal_enc.fit(data)


for c in data:
    print("{}: {}".format(c, ordinal_enc.transform([c])[0]))

# Output
# ['Bleach']: [0.]
# ['Cereal']: [1.]
# ['Toiled Roll']: [2.]
```

Categorical Feature Representing Household Essentials

The OrdinalEncoder converts categorical features into numerical representations by assigning unique integers to each category, preserving the categories order

ISSUE: These numbers seem to suggest that cereal is to bleach as toilet roll is to cereal, and that the average of toilet roll and bleach is cereal.

Features Engineering : Categorical Features

```
from sklearn import preprocessing

data = [['Bleach'], ['Cereal'], ['Toiled Roll']]


onehot_enc = preprocessing.OneHotEncoder()
onehot_enc.fit(data)


for c in data:
    print("{}: {}".format(c, onehot_enc.transform([c]).toarray()))

# Output
# ['Bleach']: [[1. 0. 0.]]
# ['Cereal']: [[0. 1. 0.]]
# ['Toiled Roll']: [[0. 0. 1.]]
```

The `OneHotEncoder` transforms categorical features into numerical representations by creating binary features for each unique category.

Curse of Dimensionality

Features Engineering : Numerical Features

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import RidgeClassifier
from sklearn import metrics
from sklearn.datasets import load_wine
from sklearn.pipeline import make_pipeline

X,y = load_wine(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)


no_scale_clf = make_pipeline(RidgeClassifier(tol=1e-2, solver="sag"))
no_scale_clf.fit(X_train, y_train)
y_pred_no_scale = no_scale_clf.predict(X_test)

std_scale_clf = make_pipeline(StandardScaler(), RidgeClassifier(tol=1e-2, solver="sag"))
std_scale_clf.fit(X_train, y_train)
y_pred_std_scale = std_scale_clf.predict(X_test)
```

Pipeline with only the classifier

Pipeline combining data standardization and classifier

Features Engineering : Numerical Features

```
print('\nAccuracy [no scaling]')
print('{:.2%}\n'.format(metrics.accuracy_score(y_test, y_pred_no_scale)))
print('\nClassification Report [no scaling]')
print(metrics.classification_report(y_test, y_pred_no_scale))

print('\nAccuracy [scaling]')
print('{:.2%}\n'.format(metrics.accuracy_score(y_test, y_pred_std_scale)))
print('\nClassification Report [scaling]')
print(metrics.classification_report(y_test, y_pred_std_scale))
```

Displaying model evaluation reports

```
Accuracy [no scaling]
75.93%
Classification Report [no scaling]
              precision    recall  f1-score   support
           0       0.90      1.00      0.95        19
           1       0.66      1.00      0.79        21
           2       1.00      0.07      0.13        14
    accuracy                           0.76        54
   macro avg       0.85      0.69      0.63        54
weighted avg       0.83      0.76      0.68        54
```

```
Accuracy [scaling]
98.15%
Classification Report [scaling]
              precision    recall  f1-score   support
           0       0.95      1.00      0.97        19
           1       1.00      0.95      0.98        21
           2       1.00      1.00      1.00        14
    accuracy                           0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54
```

Features Engineering – Numerical Features

- For most ML algorithms, the features must be all on similar scales;
- Common techniques
  - Standardization: zi = (xi – mu)/sigma
  - Min-max normalization: xi’ = (xi – min(x)) / (max(x) – min(x))
  - Feature vector normalization = xj’ = xj / ||x||

Designing the training system

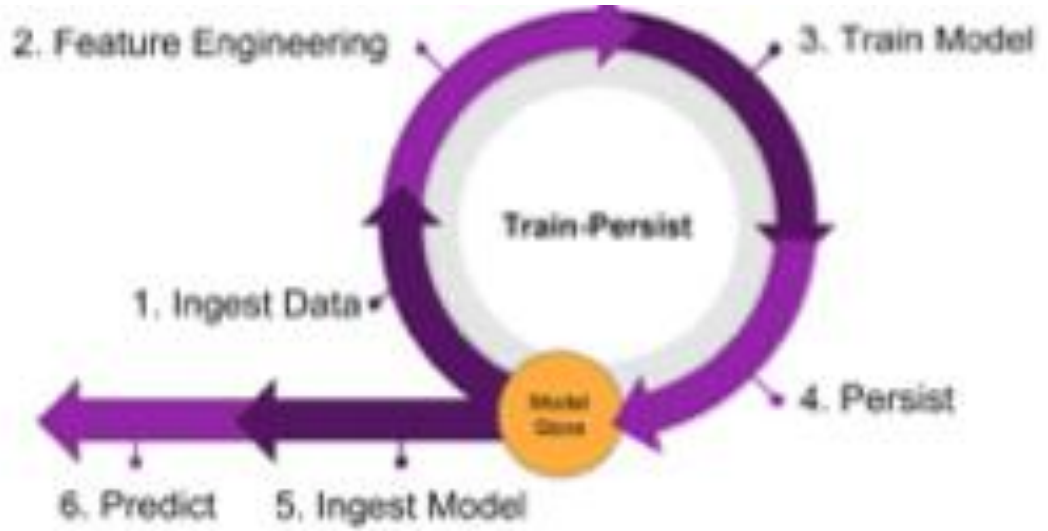
- Training and inference have quite different computational requirements and latency
- Training system design options depend on:
  - Is there an infrastructure available that is appropriate to the problem?
    - E.g., local vs cluster (e.g., Spark), CPU vs GPU vs TPU
  - Where is the data and how will we feed it to the algorithm?
    - Are we going to run a SQL query against a remotely hosted database?
    - If so, how are we connecting to it?
    - Does the machine we're running the query on have enough RAM to store the data
    - If not, do we need to consider using an algorithm that can learn incrementally?
  - How am I testing the performance of the model?
    - train/test/validation splits
    - What Cross-validation strategy?
    - What ML performance metric?
    - As ML engineers, *other* perf. measures: **training time, efficient use of memory, latency, and cost**

Designing the training system – **Train-Run**

- Perform training and prediction in the same process
- Training can be in batch or incremental
- We run our entire training process before making our predictions, with no real *break* in between
- We can automatically rule out this approach if we have to serve prediction in a very low-latency fashion; for example, through an event-driven or streaming solution
- *Could* be valid when the algorithms are very lightweight to train and you need to keep using very recent data, or when running a large batch process relatively infrequently.
- **Training as often as you predict**, hence good against drift

Designing the training system – **Train-Persist**

- Training and prediction are two separate processes
- What are our model storage options?
  - E.g. MLflow
- Is there a clear mechanism for accessing our model store (writing to and reading from)?
- How often should we train versus how often will we predict?
  - On a schedule
  - On a drift-dependent trigger



Designing the training system – **Drift Detection**

- **Concept drift**
  - Change in the fundamental relationship between the features of your data and the outcome you are trying to predict
  - E.g., at the time of training, your data show a linear relationship; after gathering a lot more data post-deployment, the observed relationship turns out to be non-linear
  - The mitigation against this is retraining with data that is more representative of the correct relationship.
- **Data drift**
  - This happens when there is a change in the statistical properties of the variables you are using as your features
  - For example, you could be using *age* as a feature in one of your models but at training time, you only have data for 16–24-year-olds.
  - If the model gets deployed and your system starts ingesting data for a wider age demographic, then you have data drift

Designing the training system – **Drift Detection**

One example that we use is alibi-detect from Seldon

```
pip install alibi
pip install alibi-detect
```

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import alibi
from alibi_detect.cd import TabularDrift
```

Designing the training system – **Drift Detection**

One example that we use is alibi-detect from Seldon

```
wine_data = load_wine()
feature_names = wine_data.feature_names
X, y = wine_data.data, wine_data.target
X_ref, X_test, y_ref, y_test = train_test_split(X, y, test_size=0.50,
random_state=42)

cd = TabularDrift(x_ref=X_ref, p_val=.05 )

preds = cd.predict(X_test)
labels = ['No', 'Yes’]
print('Drift: {}'.format(labels[preds['data']['is_drift']]))
```

Initialize drift detector

Detecting drift:
answer is NO drift

Designing the training system – **Drift Detection**

One example that we use is alibi-detect from Seldon

```
X_test_cal_error = 1.1*X_test
preds = cd.predict(X_test_cal_error)
labels = ['No', 'Yes’]
print('Drift: {}'.format(labels[preds['data']['is_drift']]))
```

Let’s simulate a scenario with drift

- This can be used independently (e.g., on schedule) or after having detected a performance degradation

- This is a type of data drift, **feature** drift. There can be **label drift** too…

Designing the training system – **Drift Detection**

Data label drift

```
cd = TabularDrift(X_ref=y_ref, p_val=.05 )
preds = cd.predict(y_test) labels = ['No', 'Yes’]
print('Drift: {}'.format(labels[preds['data']['is_drift']]))
```

This gives NO

```
y_test_cal_error = 1.1*y_test
preds = cd.predict(y_test_cal_error)
labels = ['No', 'Yes’]
print('Drift: {}'.format(labels[preds['data']['is_drift']]))
```

Let’s simulate a label drift

Designing the training system – **Drift Detection**

Concept drift

- Performance monitoring is important for this
- The following assumes PyTorch or Tensorflow, because of usage of untrained AutoEncoders

```
from alibi_detect.cd import MMDDriftOnline

ert = 50
window_size = 10
cd = MMDDriftOnline(X_ref, ert, window_size, backend='pytorch',
n_bootstraps=2500)
```

Online Maximum Mean Discrepancy method

ert is Expected runtime to regulate the false positive/negatives trade-off
window_size regulates the short vs long-term drifting patterns detection

Designing the training system – **Drift Detection**

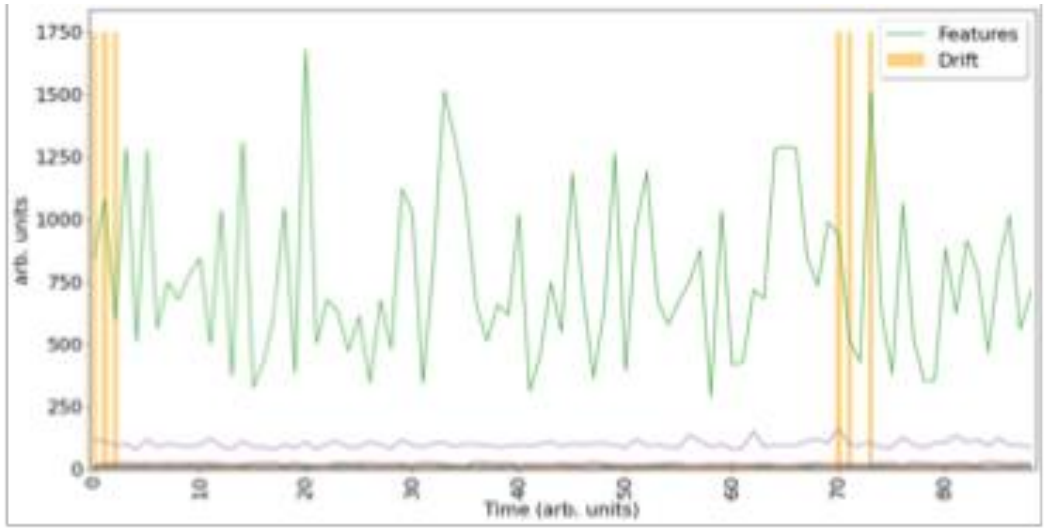
Concept drift

- Performance monitoring is important for this
- The following assumes PyTorch or Tensorflow, because of usage of untrained AutoEncoders

```
cd.predict(x)['data'] ['is_drift']
```

Online drift detection

x is the feature vector of an instance. Performing on all data and plotting gives the plot on the right



Designing the training system – **Drift Diagnosis**

- We need to look at the features … but **not all are equally important**
- Need to understand what are the most important ones before prioritizing which ones need remedial action.
- **Feature importance**
  - Model-dependent , e.g., decision trees, random forest (e.g., **Gini importance**)

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
rf = RandomForestClassifier(max_depth=2, random_state=0)
mdi_importances = pd.Series(rf[-1].feature_importances_,
                        index=feature_names). sort_values(ascending=True)
```

Impurity measure
- Can exhibit bias toward features with high cardinality (e.g., numerical)
- Computed only on training set data, ignoring generalizability on unseen test data.

Designing the training system – **Drift Diagnosis**

- We need to look at the features … but **not all are equally important**
- Need to understand what are the most important ones before prioritizing which ones need remedial action.
- **Feature importance**
  - Model-dependent , e.g., permutation importance

```
from sklearn.inspection import permutation_importance
result = permutation_importance( rf, X_test, y_test, n_repeats=10,
                random_state=42, n_jobs=2 )
sorted_importances_idx = result.importances_mean.argsort()
importances = pd.DataFrame( result.importances[sorted_importances_idx].T,
                columns=X.columns[sorted_importances_idx])
```

Take a feature, shuffle in several ways, see the impact on performance

Designing the training system – **Drift detection**

- We need to look at the features … but **not all are equally important**
- Need to understand what are the most important ones before prioritizing which ones need remedial action.
- **Feature importance**
  - Model-independent , e.g., SHAP (SHapley Additive exPlanation)

```
pip install shap
```

```
explainer = shap.Explainer(rf.predict, X_test)
shap_values = explainer(X_test)

shap.plots.bar(shap_values)
```

*To determine feature importances, you should take the average of the absolute magnitudes of the `shap_values`

training the model on all permutations of features that include or exclude the considered feature and then calculating the marginal contribution to the predicted value of that feature.

Designing the training system – **Drift Remediation**

- Remove feature(s) and retrain
- Retrain with more data
- Roll-back the model
- Rewrite or debug the solution
- Fix the data source

Designing the training system – **More on monitoring -Evidently**

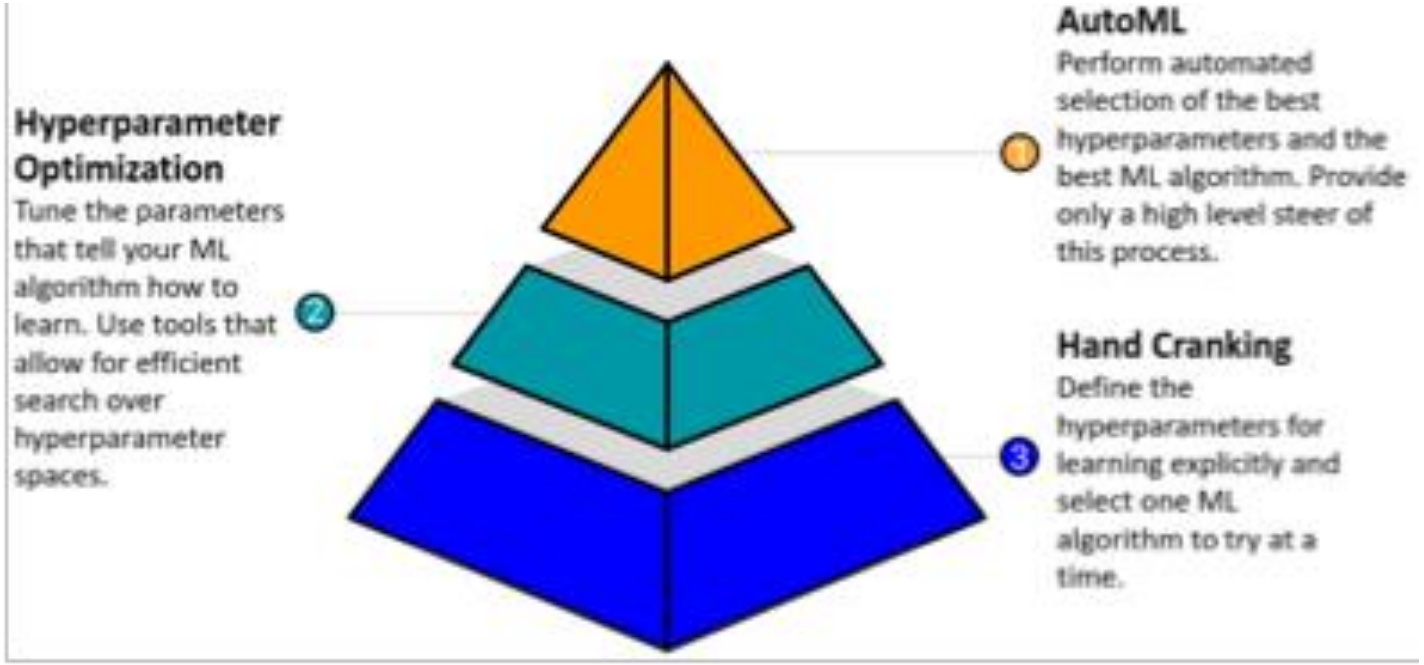
```
pip install evidently
```

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
import evidently
from evidently.report import Report
from evidently.metric_preset import DataDriftPreset
# Grab the data
wine_data = load_wine(as_frame=True)
feature_names = wine_data.feature_names

X, y = wine_data.data, wine_data.target
X_ref, X_test, y_ref, y_test = train_test_split(X, y, test_size=0.50,random_state=42)
data_drift_report = Report(metrics = [DataDriftPreset()])
data_drift_report.run(reference_data=X_ref, current_data=X_test)
data_drift_report.save_json('data_drift_report.json')
data_drift_report.save_html('data_drift_report.html')
```

Designing the training system – **Automating Training**

- How to streamline, optimize, and, in some cases, fully automate elements training



Optimizing hyperparameters - Examples

Algorithm	Hyperparameters	What This Controls
Decision Trees and Random Forests	Tree depth. Min/max leaves.	How many levels are in your trees. How much branching can occur at each level.
Support Vector Machines	C Gamma	Penalty for misclassification. The radius of influence of the training points for Radial Basis Function (RBF) kernels.
Neural Networks	Learning rate. Number of hidden layers. Activation function. *Many more….*	Update step sizes. How deep your network is. The firing conditions of your neurons.
Logistic Regression	Solver Regularization type. Regularization prefactor.	How to minimize the loss. How to prevent overfitting/make the problem well behaved The strength of the regularization type
K-Nearest Neighbors	K Distance metric	The number of clusters. How to define the distance between points.
DBSCAN	Epsilon Minimum number of samples. Distance metric.	The max distance to be considered neighbors. How many neighbors are required to be considered core. How to define the distance between points.

Scikit-learn, Hyperopt

- For a finite number of hyper-parameter value combinations, we want to find the set that gives the best model performance.
  - Just another optimization problem that's similar to that of training in the first place!

- Scikit-learn offers (https://scikit-learn.org/stable/modules/grid_search.html)
  - **Grid Search**
  - **Random search**
  - **Successive halving**
- Hyperopt - `https://github.com/Hyperopt/Hyperopt`
- Methods supported:
  - **Random search**
  - **Tree of Parzen Estimators (TPE)** – Bayesian method
  - **Adaptive TPE**

Hyperopt

- Define the space of hyper-parameters to cover:
  - Whether we want to reuse parameters that were learned from the previous model runs (`warm_start`)
  - Whether we want the model to include a bias in the decision function (`fit_intercept`)
  - The tolerance set for deciding when to stop the optimization (`tol`)
  - The regularization parameter (`C`)
  - Which `solver` we want to try
  - The maximum number of iterations, `max_iter`
  - Define an objective function to optimize. In the case of our classification algorithm, we can simply define the `loss` function we want to minimize as 1 minus the `f1-score`.

See Notebook and py file

Optuna

- Same process as above
- *Define-by-run :* dont ‘need to define the full set of parameters, which is *define-and-run*.
  - Can provide some initial values and ask Optuna to suggest experiments to run
- Methods:
  - **grid search**
  - **random search**
  - **TPE**
  - **Covariance Matrix Adaptation Evolution Strategy** (**CMA-ES**)
- Applies **Pruning** or **automated early stopping**. During optimization, if detects evidence that a trial will not lead to a better trained algorithm, it terminates that trial

See Notebook and py file

AutoML

Goal: automating the search for the best models and hyperparameters setting

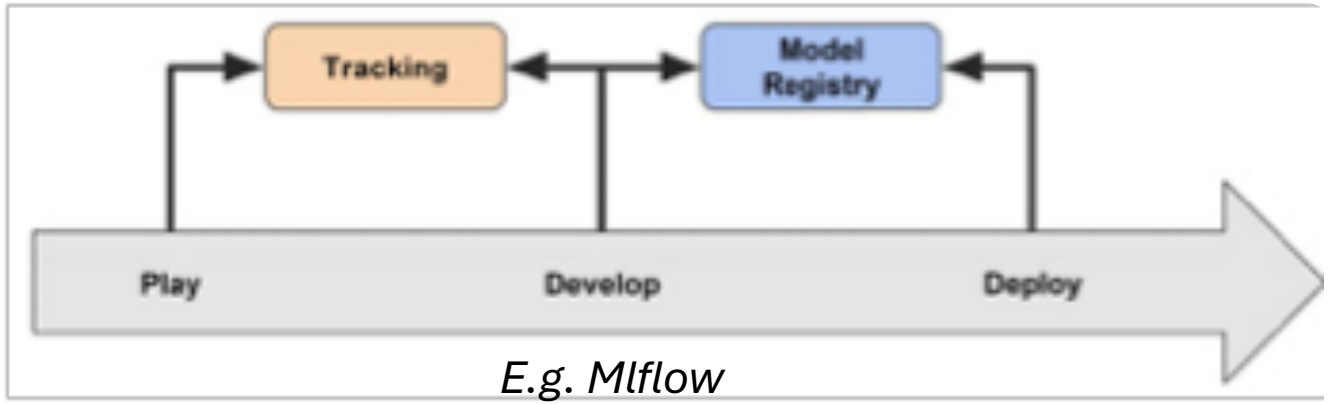
AutoKERAS – see Python Notebook

- Other Top 10 AutoML Python Libraries
  - H2O AutoML
  - PyCaret
  - Auto-sklearn
  - MLBox
  - TPOT
  - Autokeras
  - AutoGluon
  - Auto-ViML
  - EvalML
  - FLAML

https://www.run.ai/guides/automl/automl-python

Persisting your model

- **Model control System – Goals:**
  - Automate (as far as possible) the search for a first working model, so that we can continually update and create new model versions that still solve the problem in the future
  - Have a simple mechanism that allows sharing the results of the training process with the prediction system
  - Have the ability to track experiment results, to keep the results of the **Play** phase and build on these during the **Develop** phase.
  - Track experiments, test runs, and hyperparameter optimization results in the same place during the **Develop** phase.
  - Then, we can start to tag the performant models as ones that are good candidates for deployment, thus bridging the gap between the **Develop** and **Deploy** development phases.



E.g. Mlflow

Persisting your model

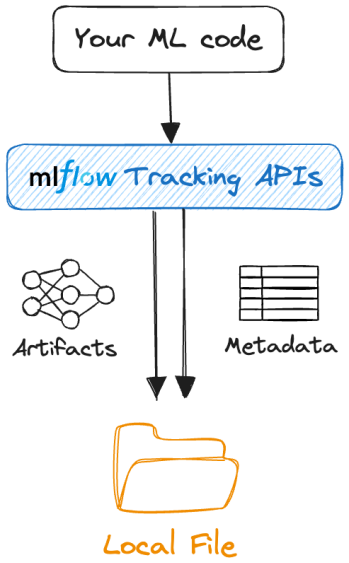
- How to take a logged model and push it to the registry
- How to update information such as the model version number in the registry
- How to progress your model through different life cycle stages.
- How to retrieve a given model from the registry in other programs – a key point if we are to share our models between separate training and prediction services.

- How to chain our main training steps together into single units called **pipelines**.

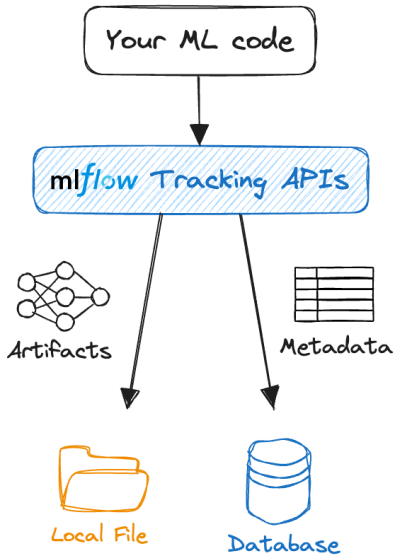
See mlflow-advanced folder in the repo

ML flow model

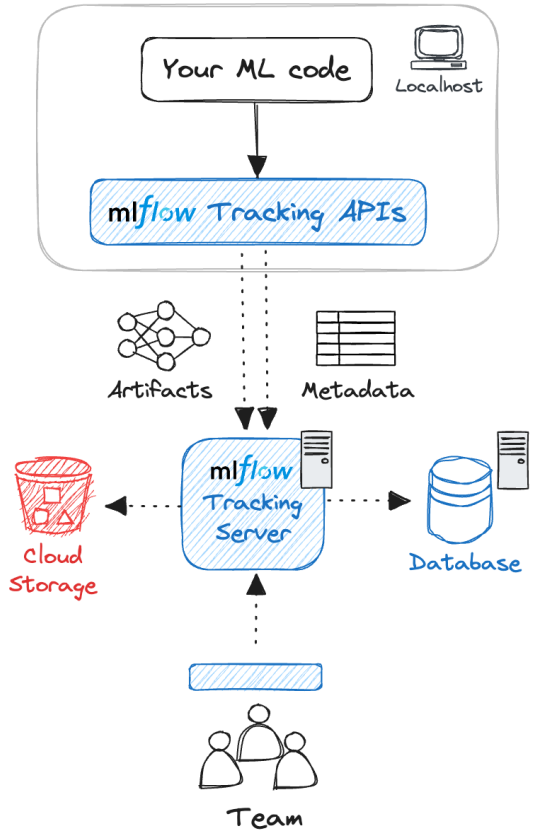
1. Localhost (default)



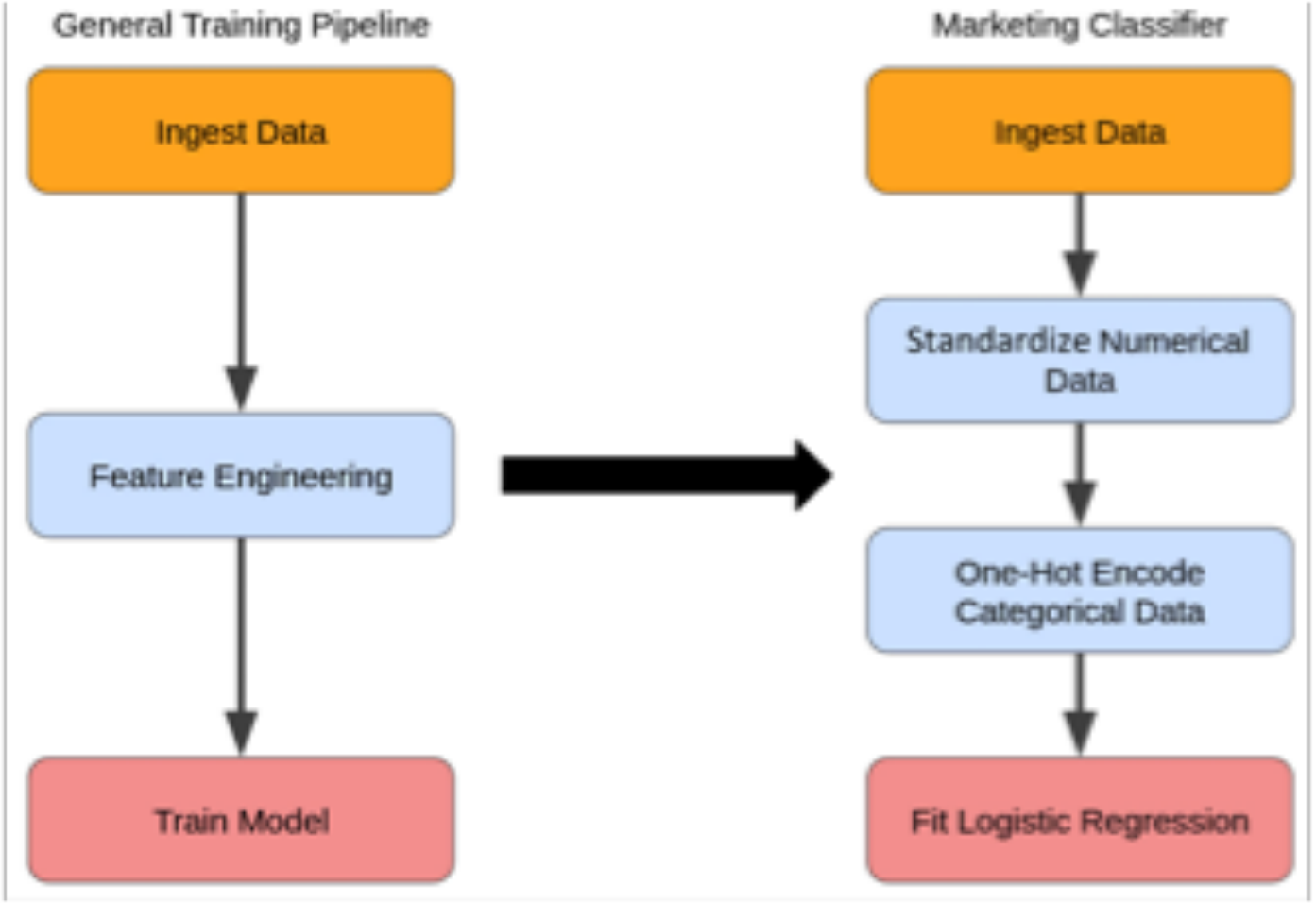
2. Localhost w/ various data stores



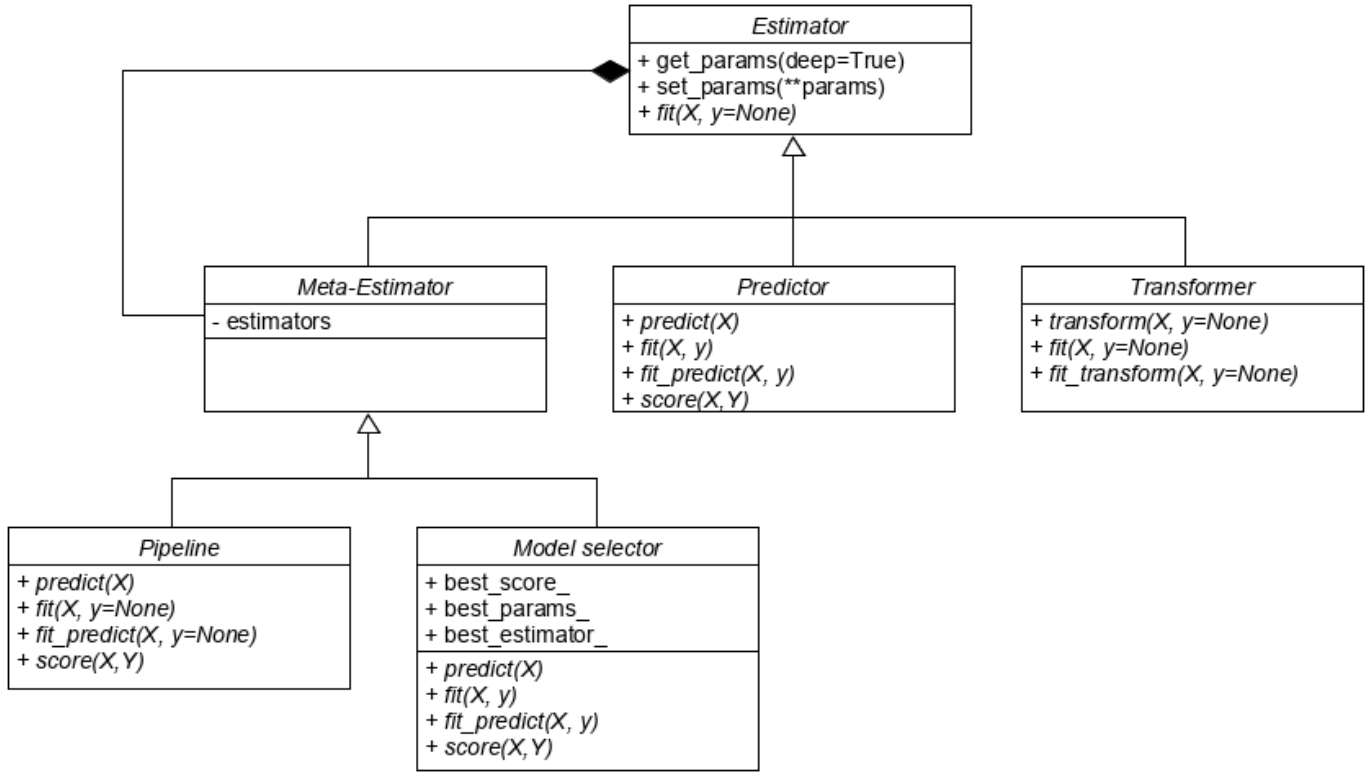
3. Remote Tracking w/ Tracking Server



Pipelining all the steps

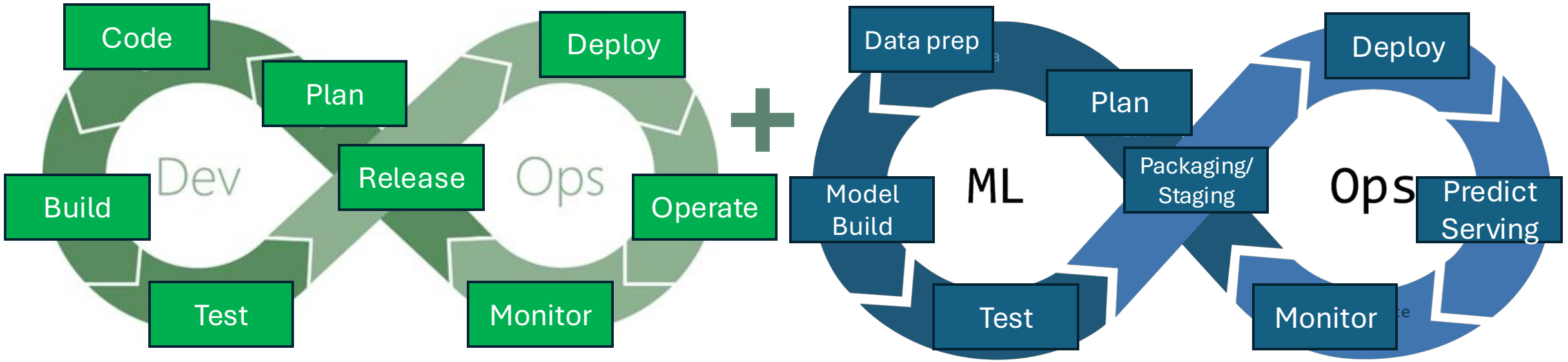


Scikit-learn pipelines

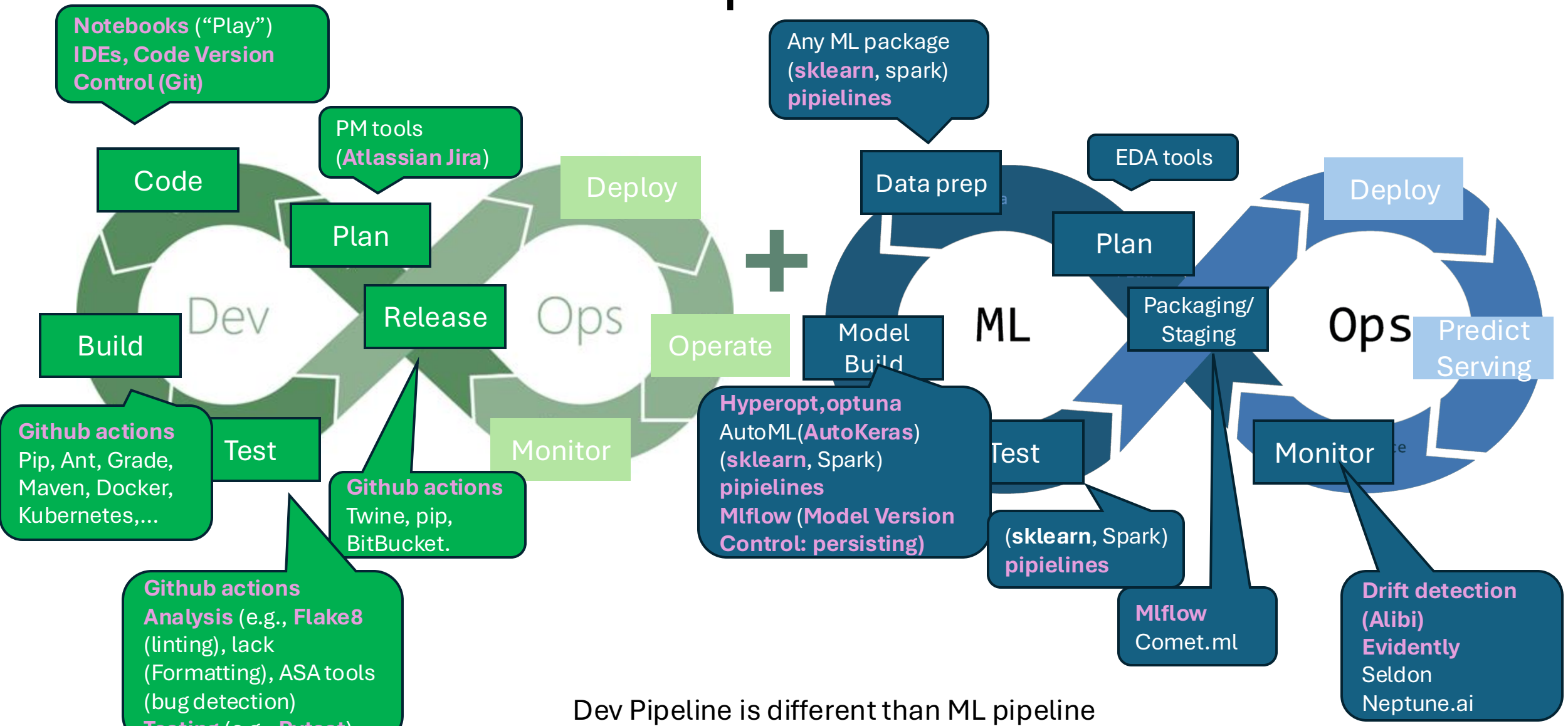


See mlflow-advanced folder in the repo

Where we are in the process…



Where we are in the process...



Dev Pipeline is different than ML pipeline