Level 0 Maturity : Manual Process

→ Manual build, train, line, and deploy model { Jupyter Notebook }

→

Characteristics.

① Manual | Script Driven | Interactive process.

② Disconnection between ML and operations teams.

③ Infrequent release of models | no CD of models.

④ No CI. or CI is ignored.

⑤ Deployment ⇒ prediction service

✗ No ML pipeline deployed.
↓↓
No Cont. Training.

⑥ Lack of monitoring of performance.

Challenges : -

① Time consuming

② Maintenance cost → high

③ No new ML ideas will pushed easily

④ Prone to bugs.

Possible solution:

(1) Actively monitor your model in production.

(2) frequently (depends on the situation) retrain your model.

(3) frequent experiments with new optimized implementation to produce better models.

for eg: Choose or experiment with latest SOTA

Make it more adaptable / Robust

Level 1 maturity : ML pipeline automation.

Aim

→ perform CT of the model in production by automating ML pipeline ⇒ CD of model in prediction service

→ To automate :-

→ new data ⇒ fresh model based on latest trend

→ data & model validation

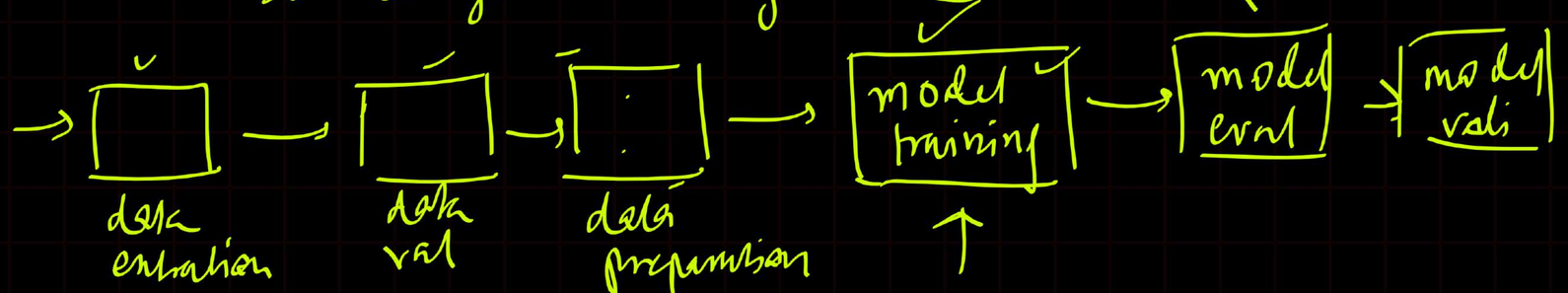
→ pipeline trigger & metadata management

→ Pipeline Orchestration

Characteristics :-

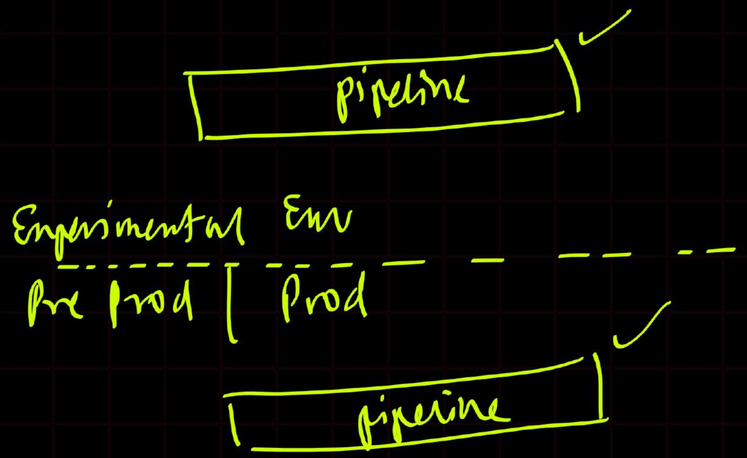
1 Rapid Experiment

because you are using orchestration



2. CT of model in production

3. Experimental - Operational Symmetry.



4. Modularized code components and pipeline :-

- Reusable
- Shareable

5. CD of model.

6. Pipeline deployment

deploy pipeline → deploys model

Frequently used terms :-

① Pipeline Orchestration
ordered execution of components.

earlier failure of M2 project :-
↳ Glue code | code to connect two components of pipeline.

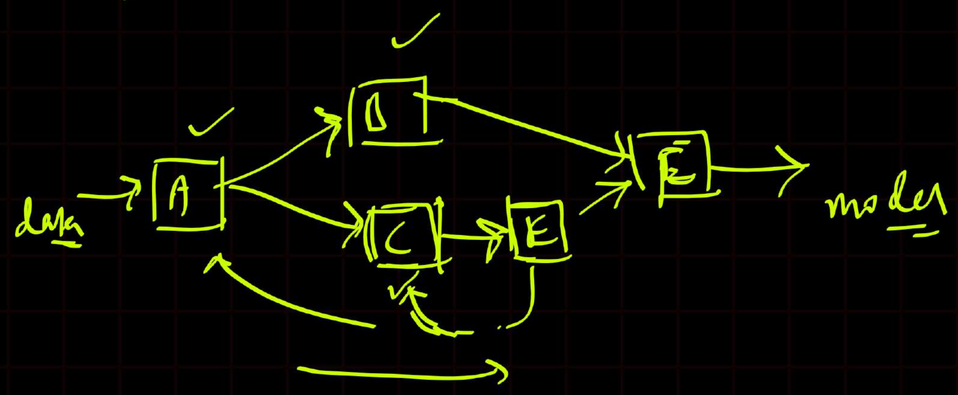
Tools to solve above issue :-
① Apache Beam
② " Airflow → UI
③ Kubeflow (Kubernetes based)
} supported by major cloud platforms

Advantage :-
① Standard Orchestration + abstraction
② Supported by many cloud platforms
③ Easy to monitor / debug.

UI

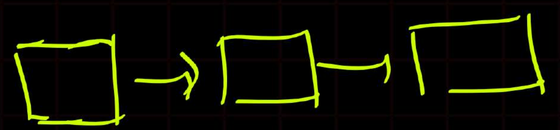
DAGs → Directed Acyclic Graphs.

Graphs →



cyclic → ∞ loop

ML pipelines



DAGs → core component of all your pipeline orchestration tools.

Level 2. → Important components.

pipeline execution

new data / live data → new model version on new data

Requirements to fulfill above execution :-

① Automatic data validation

↳ Decides whether to start retraining

or

stop execution of pipeline. & so for manual investigation {DS}

on following basis :-

Ⓐ Data schema skews :-

↳ data is not in compliance with expected schema or as per DSA → Data sharing agreement

Solution :- → stop pipeline & let DS team investigate.

serve last stable model

or

maintenance time out { worst case }

(5) Data Validation skew :-

When data pattern or statistical properties changes

↓

Trigger retraining.

Requirement

(2) Model Validation

↳ After successful model training on new data

↓

Evaluate & validate your model before pushing it into production.

Offline steps	Online steps
before serving your model	~~model is serving~~ after deployment

offline steps :-

(A) Get evaluation metrics using your trained model on Test data set

to

know its prediction quality.

(B) Compare above metrics with current- model in production

(C) Performance } your model must be consistent on regional / cluster samples

(D) Test the model deployment for infrastructure compatibility

pickle, h5, pb, pt

REST API
gRPC

n features
n-1 features

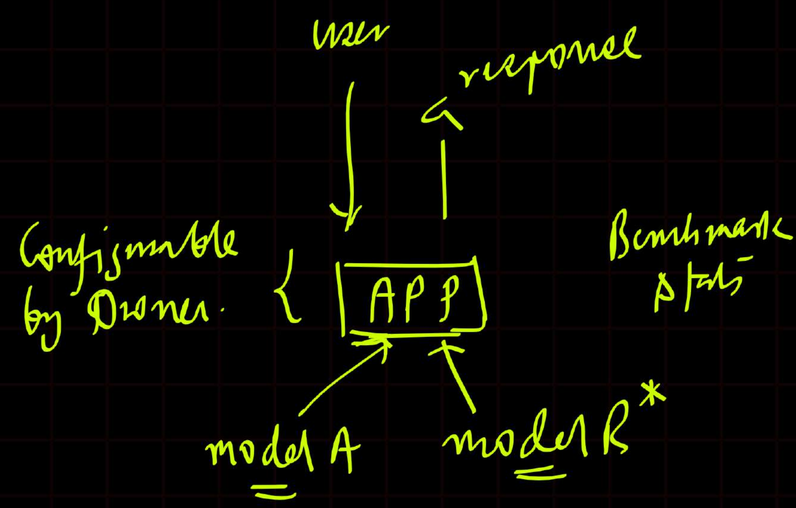
Online model validation (when your model is serving in production)

(A) Canary deployment
↳ roll out for subset of users or servers
eg. Colab
Tesla autopilot
HAL → Tejas, LCA, Dhruv

(B) A/B testing setup

Advantage: you get info & data on which model A & B performed well.

Disadvantage: Not a dynamic routing

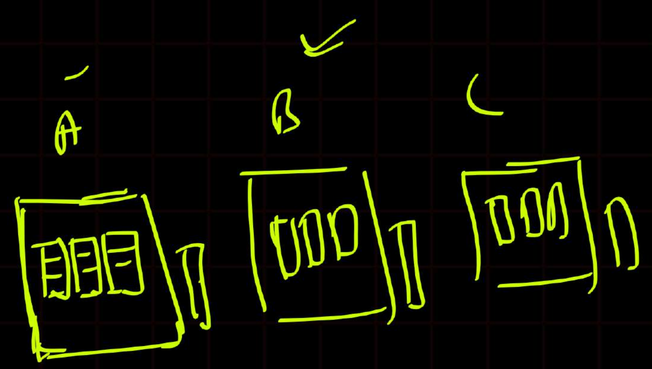


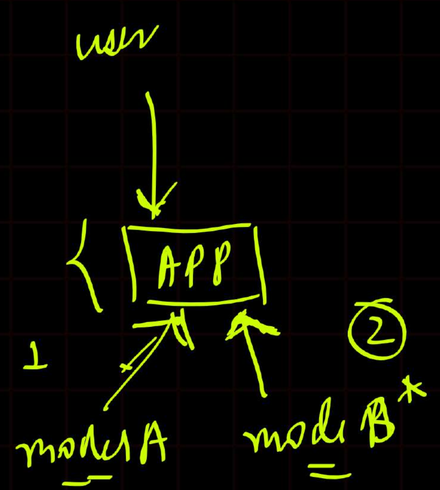
⑥ Multi Arm Bandit deployment

Algorithm

Dynamically allocates in favour of better performing variation

{ Dynamic routing }





Advantage : Dynamic routing

Disadvantage : You'll never get to know which input data point which was responsible for better model performance

② Feature store

↳ central repo

↳ for? → standard definition storage & access of feature for training and serving

features → (x y z) → x y z

x y z

** Central Approach provides better control

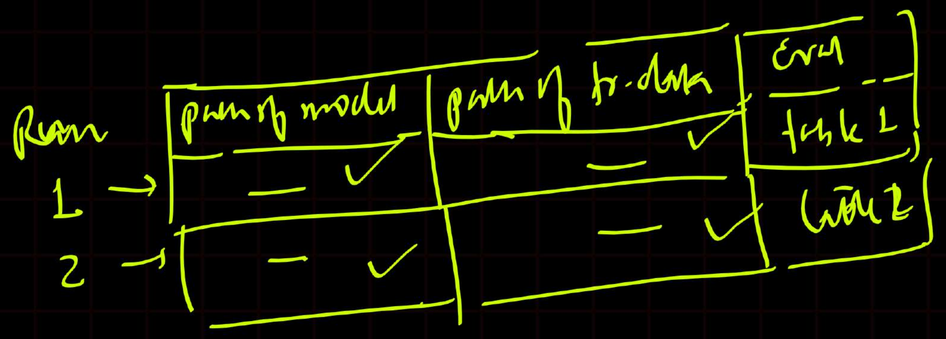
how? → provides API for
→ high throughput batch serving
→ low latency real time serving for feature values.
→ supports training & serving workloads

→ happens ! how?
↳ Avoids similar features, different definitions. how it maintains feature & relevant metadata
↳ serve up to date feature value
* ↳ Avoids training-serving skew. by using feature store as a central source for experimentation | CT | online serving.

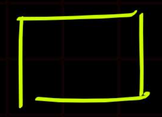
(3) Metadata Storage —

↳ records information about each execution of ML pipeline

why?



↳ To help in: —



① Reproducibility:

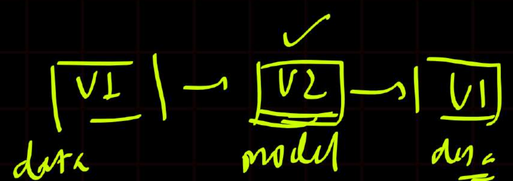
② Comparisons

③ Artifacts Lineage

④ Debugging errors & anomalies

What it records in every run?

- (i) pipeline components & component versions executed.
  e.g. dvc
  md5 :



- (ii) start / end date & time / Time duration for execution of each step.
- (iii) Executor (user) —
- (iv) parameters passed to the pipeline
- (v) pointers to the artifacts produced in each step
  e.g. path to prepared data
  validation anomalies
  computed stats
  } keeps in debugging the failed step in ML pipelines
- (vi) pointer to prev. stable model in case rollback
- (vii) Model eval metrics for every training & testing ses-

④ ML pipeline trigger.

↳ train on new data in production.

When you can trigger the training. ?

↳ On demand: Manual execution

↳ scheduled: if new data is comes at regular intervals then scheduled trigger is fine.

↳ Availability of New training data:

↳ Model performance degradation: on noticeable change in performance drop.

↳ significant change in data distribution.

in experimentation EDA → if unexpected → fix the computed stats

↑ ↓ ↔

new stats

When level 1 is suitable ?

→ No. freq deployment of new implimintation of pipeline
→ Manual testing of few pipeline is sufficient.
→ Manual deployment of new pipeline.
→ Submit tested code base for pipeline to IT team to deploy your ML pipeline into target environment
→ New model based on new data { not on new ideas }

Challenge :

→ Time consuming
→ No freq updates

Solution :

↳ CI CD setup to automate build | test of | deployment of ML pipeline

Level 2 : CI/CD ML pipeline automation.

↳ rapid & reliable updates of ML pipeline

↳ automating build / test / deployment of ML pipeline

beneficial for DS team ⇒ They can now focus on new R&D. (new ideas

↳ New Idea can be brought into production easily

CI/CD steps

(i) Dev / Expt:
- i/p: Raw data
- perform: experimenting with new ML ideas.
- o/p: source code for ML pipeline steps.

(ii) Pipeline CI:
- i/p: prev o/p
- perform: testing,
- o/p: Packages / containers / executables

(iii) Pipeline CD:
- i/p: prev o/p
- perform: Deployment packages into the target environment.
- o/p: Deployed ML pipeline with new ML idea

(iv): Automated triggering / CT:
- i/p: prev o/p
- perform: trigger training in prod. based on schedule => (other examples discussed)
- o/p: Trained model → model registry.

(v) model CD :

↳ i/p : pred o/p

perform : pick the suitable model from model reg.
& integrate it with prediction service (flask).

o/p : pred. service.

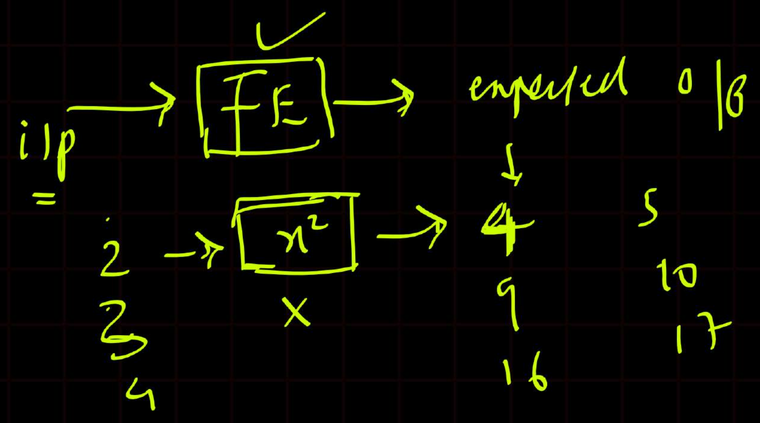
(vi) Monitoring :

perform : collect stats of the model performance on new data

o/p → trigger execution of pipeline in prod. { minor changes }
→ trigger the new experiment cycle { major changes }

more on CI!

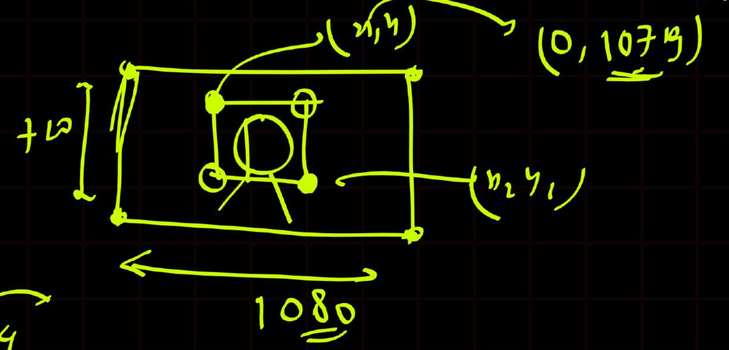
① unit test for feature engineering logic!



② unit test for methods implemented in source code.

③ Test for model training convergence.

* ④ Test for training NaN, divide by zero, large no. o/p from model.

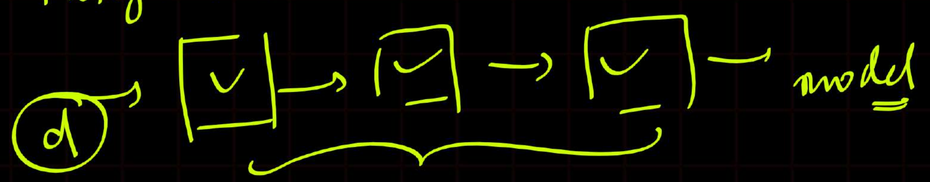


Chemicals

pH → 0 → 14

⑤ Testing each component writes its generating the expected artifacts or not!.

⑥ Integration test -



Colour images

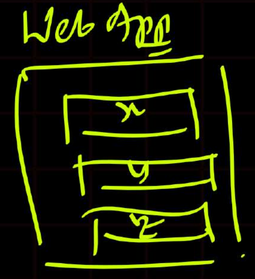
③

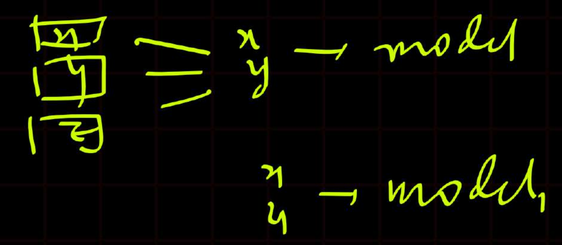
Hyper spectral images

3+ , MRI

How CD can be more smooth process?

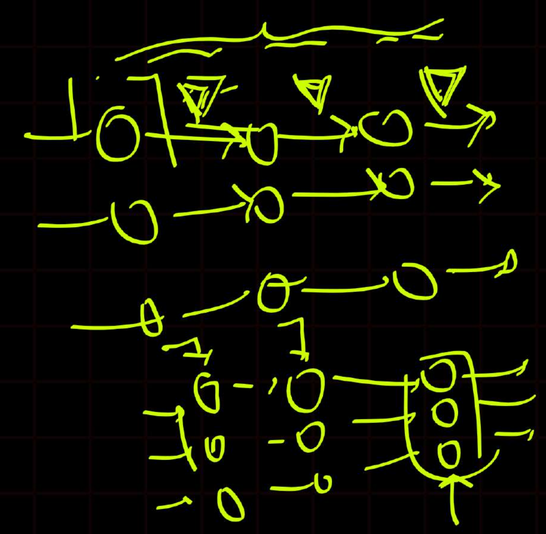
(1) Verify compatibility of model with target infra before deployment





packaging, → memory ! compute
GPU acceleration TPU acceleration

500 MB → 450 MB



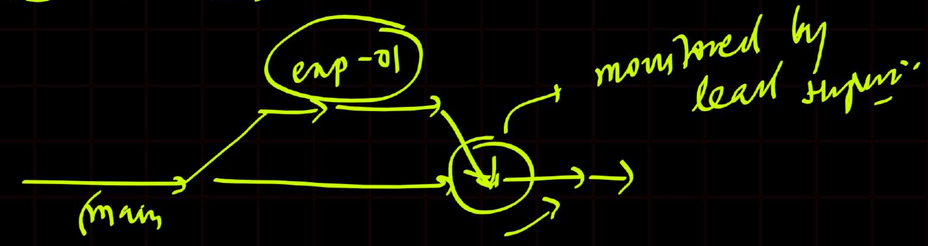
(2) Testing of pred. service API

API → JSON

(3) Test for pred service performance.
└ load testing → query per second, model latency

Deployment type:

Automated: You push the code in dev. branch of your github repo. → CI → CD



Semi Automated:

Manual:

after several successful iterations on pre prod^n

deployment is done manually

Summary level 2 :

(1) Adaptive value.

(2) Less involvement of DS team in maintainance because of customisation

(3) Less time consuming

(4) More freq updates

(5) Scalability.

(6) Maintainance : Easy.

When you should use or go for AI/MLOps. ?

① Experimental phase POC ? ✗

② ~~When~~ You have deployed a webapp ~~only~~ + model ✓
your users are growing ?