Scaling Machine Learning with Containers: Lessons Learned

Rustem Feyzkhanov

ML Engineering Manager

Instrumental

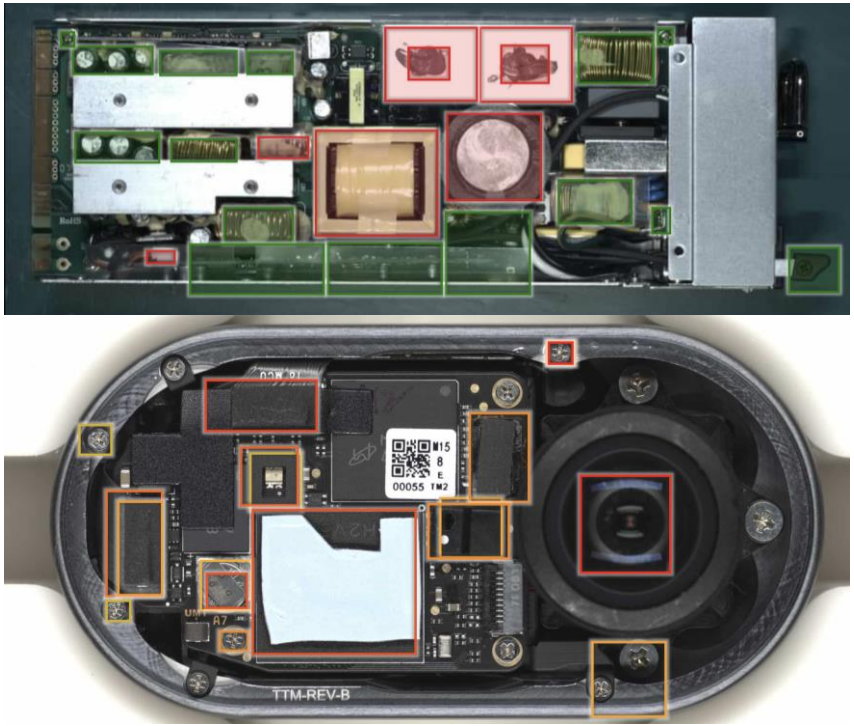
INSTRUMENTAL

Agenda

- Instrumental
- Manufacturing ML project lifecycle
- ML team challenges
- Migration
- Lessons learned

Let AI do the heavy lifting

- Discover unknown defects
- Monitor for known defects
- Drive corrective action



Convert “AI discoveries” into AI tests that intercept issues on the line

Challenges with infrastructure for MLaaS

Scale
Large-scale model training and inference

Flexibility
Need to make sure models can work for different customers

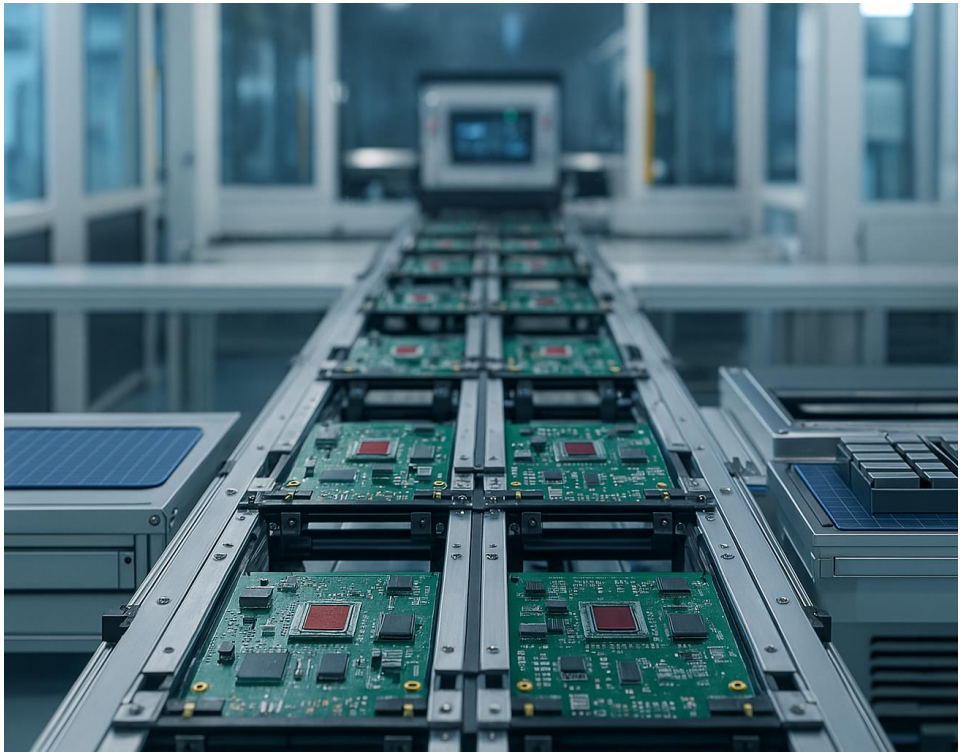
Robustness
Models are running on the line and need to be reliable

Infrastructure context

Training pipeline in production
Models can be trained on demand and retrained based on data shift

Backend as an ML orchestrator
Model registry, retraining, and prediction logic is stored by Instrumental

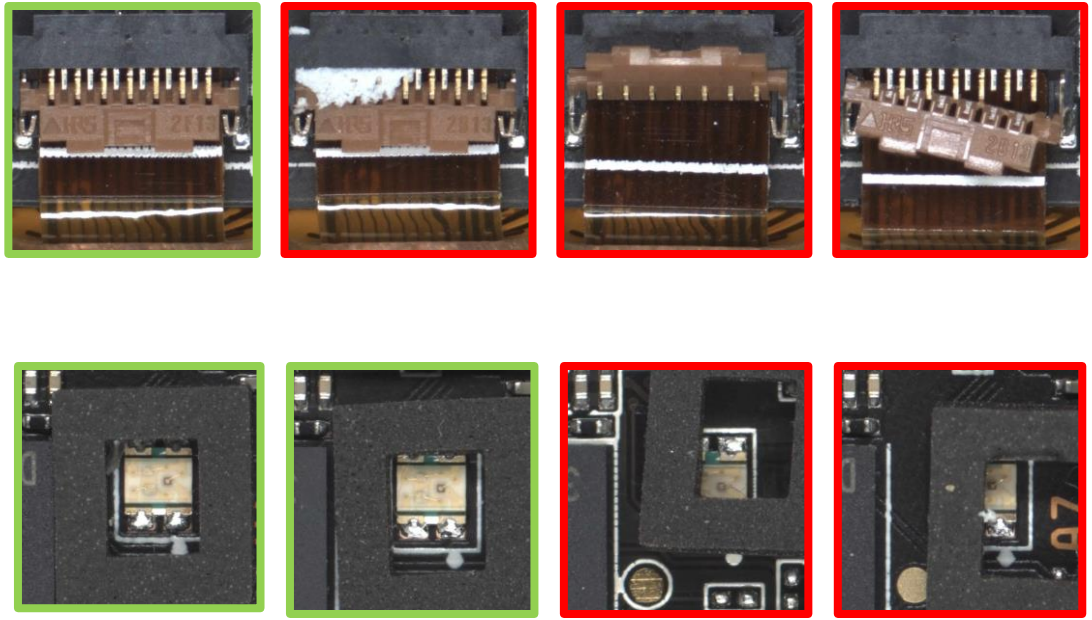
Images are the primary data
The primary models are working with images



ML project lifecycle and challenges

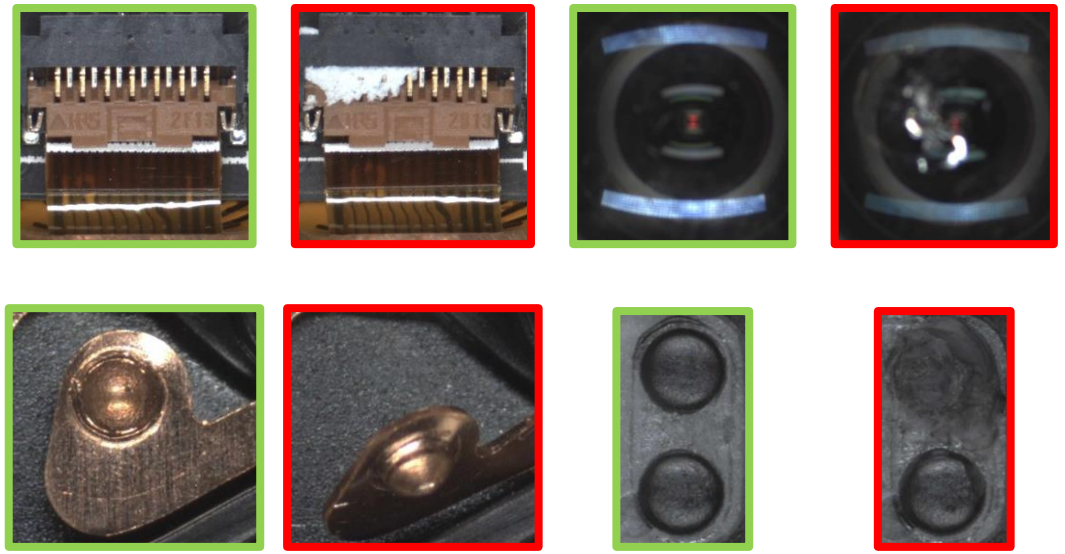
Business problem – QC in industrial manufacturing

- Handle a high variety of products and defects
- Train the model from a couple of images
- Handle scale when a lot of images are uploaded daily



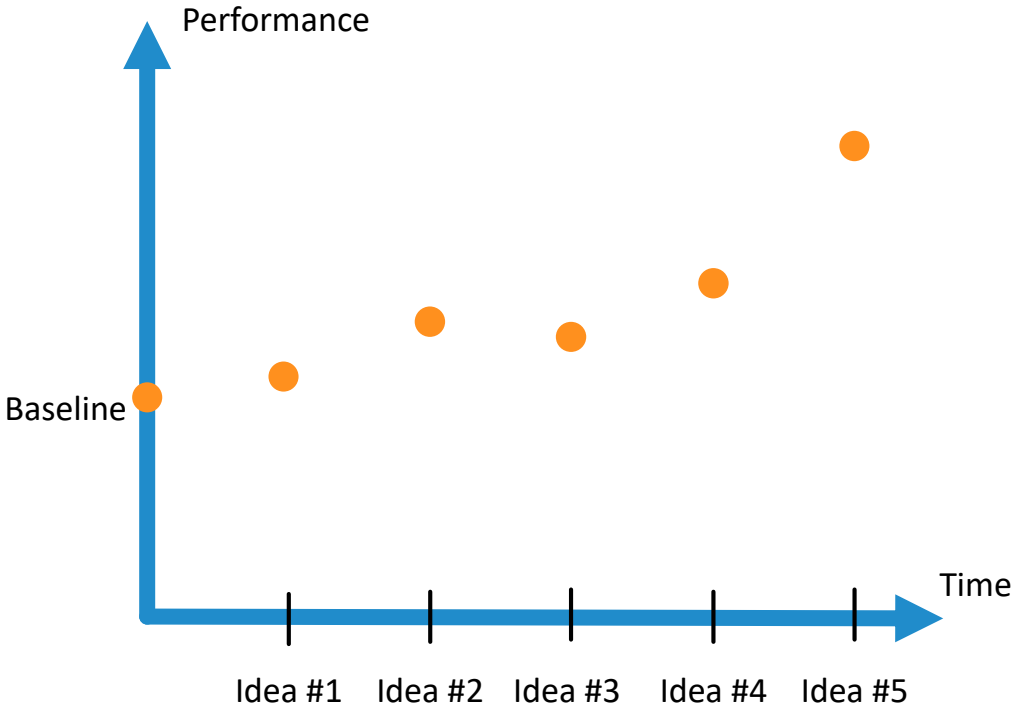
Step 1 – Research

- Gather training data
- Prototype the solution
- Get baseline performance
- Iterate on different approaches/improvements
- Find the best-performing solution

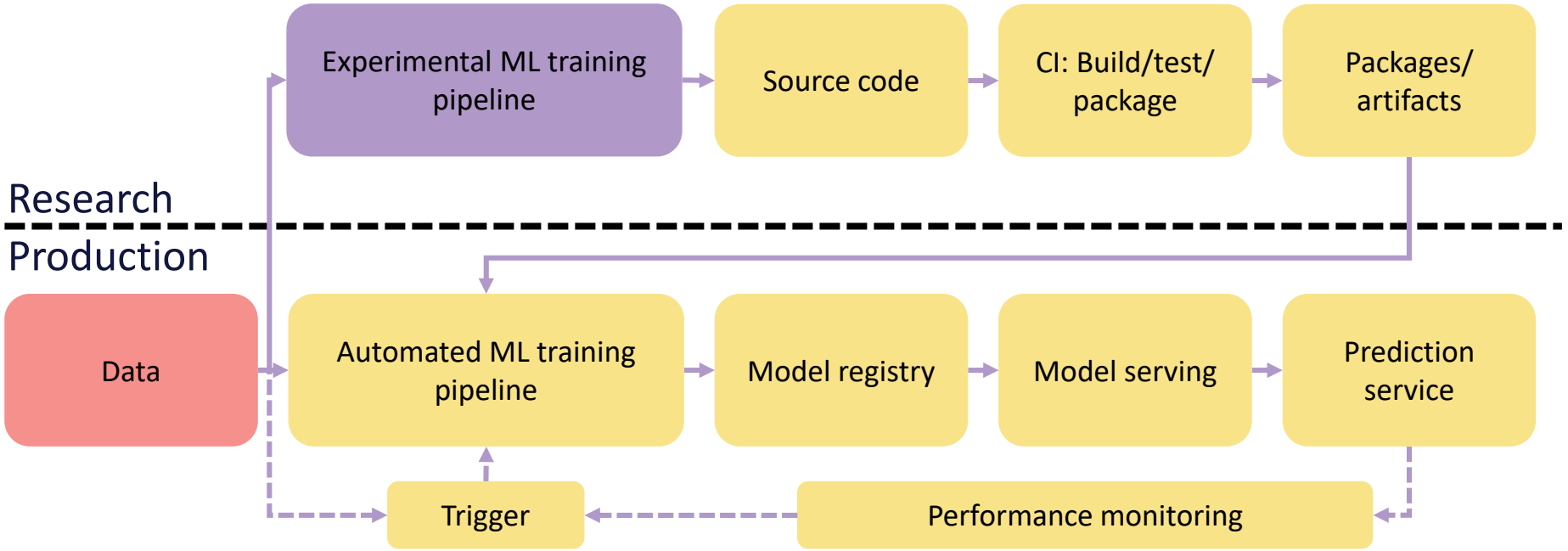


Step 1 – Research

- Gather training data
- Prototype the solution
- Get baseline performance
- Iterate on different approaches/improvements
- Find the best-performing solution

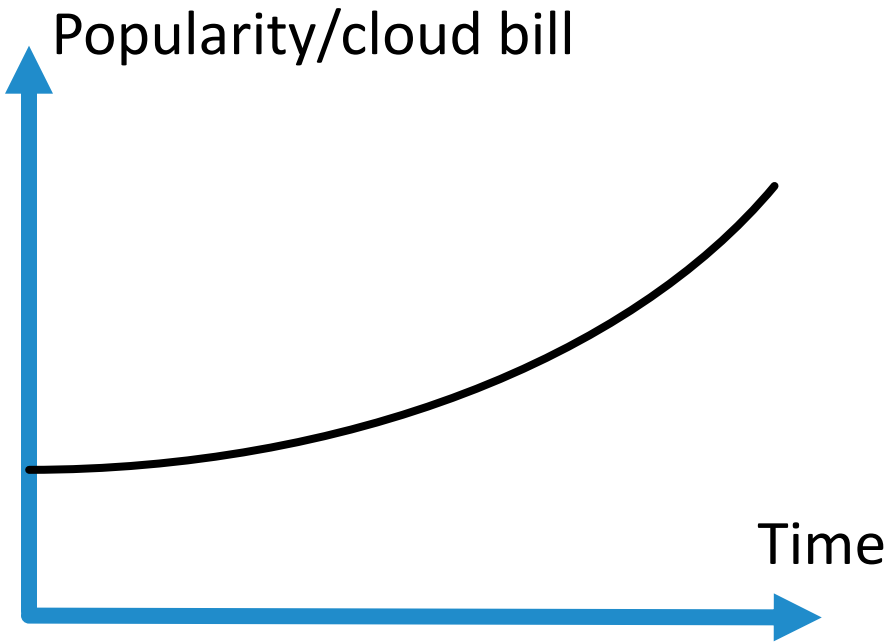


Step 2 – Productization



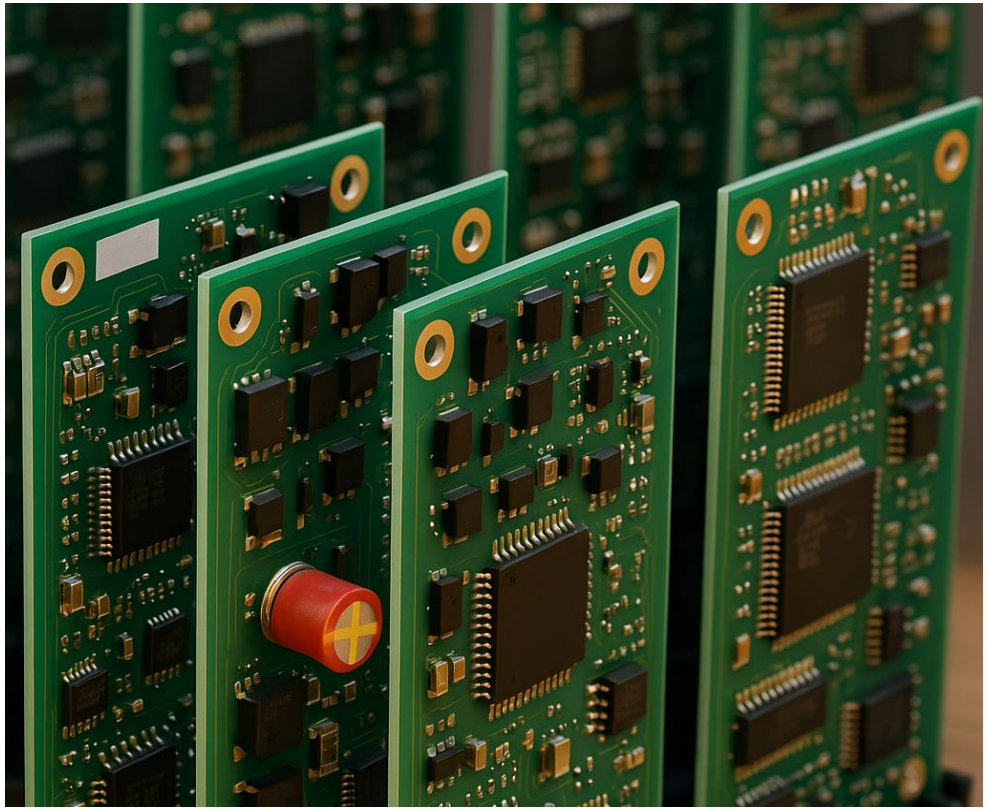
Step 3 – Scaling

- Scaling from 1x to 100x
- Monitoring/troubleshooting in production at scale
- Optimizing cost



ML team challenges

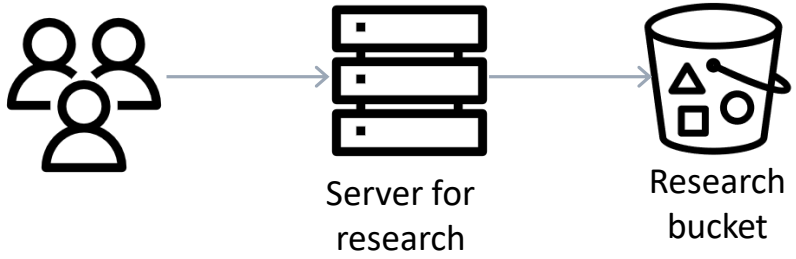
- The speed of research
- Productizing the ML model
- Scaling production

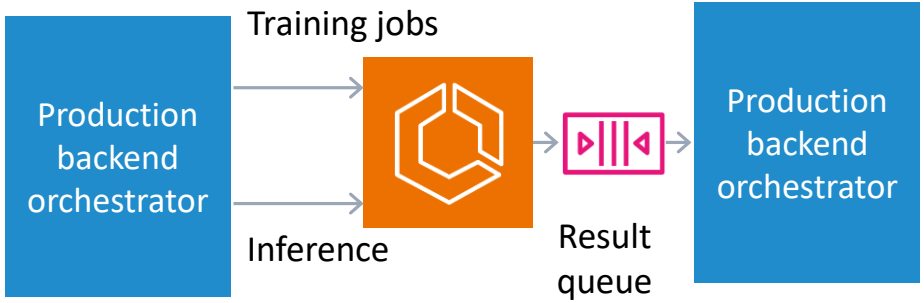


ML containers and migration

Starting point

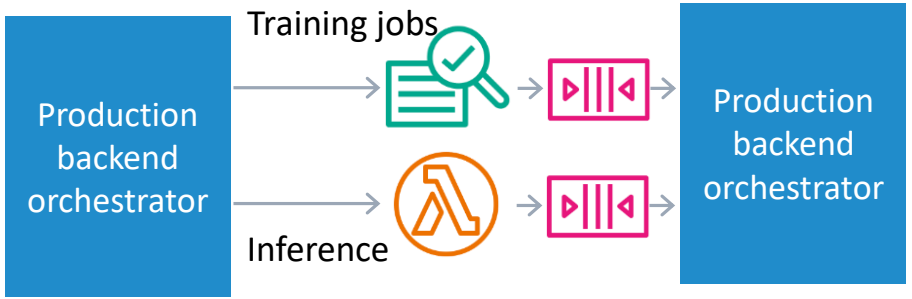
- Separate research environment for running experiments
  - Steep learning curve to start running experiments
  - Big gap between research and production
- Single production cluster for ML workflows
  - Challenging to process peak inference loads
  - Hard to troubleshoot training jobs



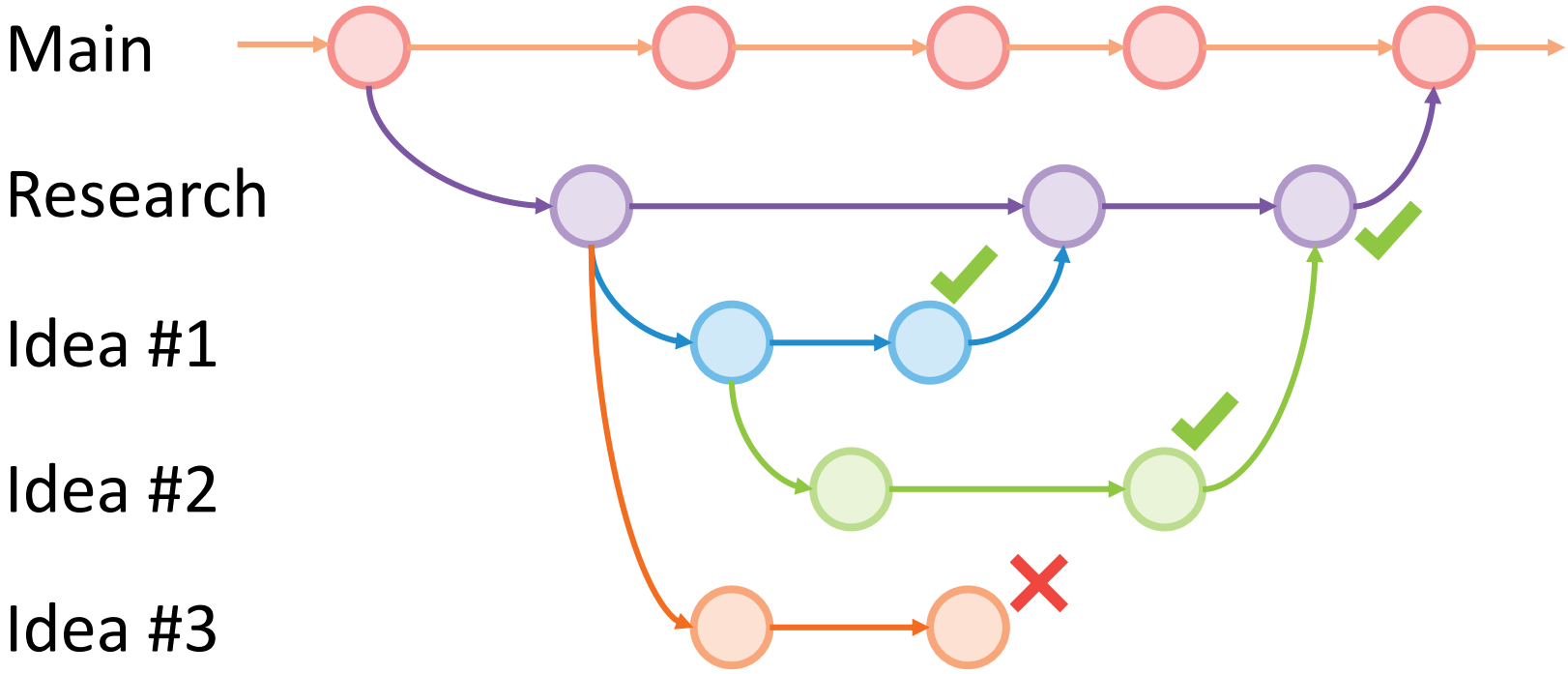


Migration to scalable solution

- Inference
  - Migrate inference to serverless solution
- Training jobs
  - Move training jobs from cluster to async pipeline
- Research environment
  - Reuse async pipeline for the research environment



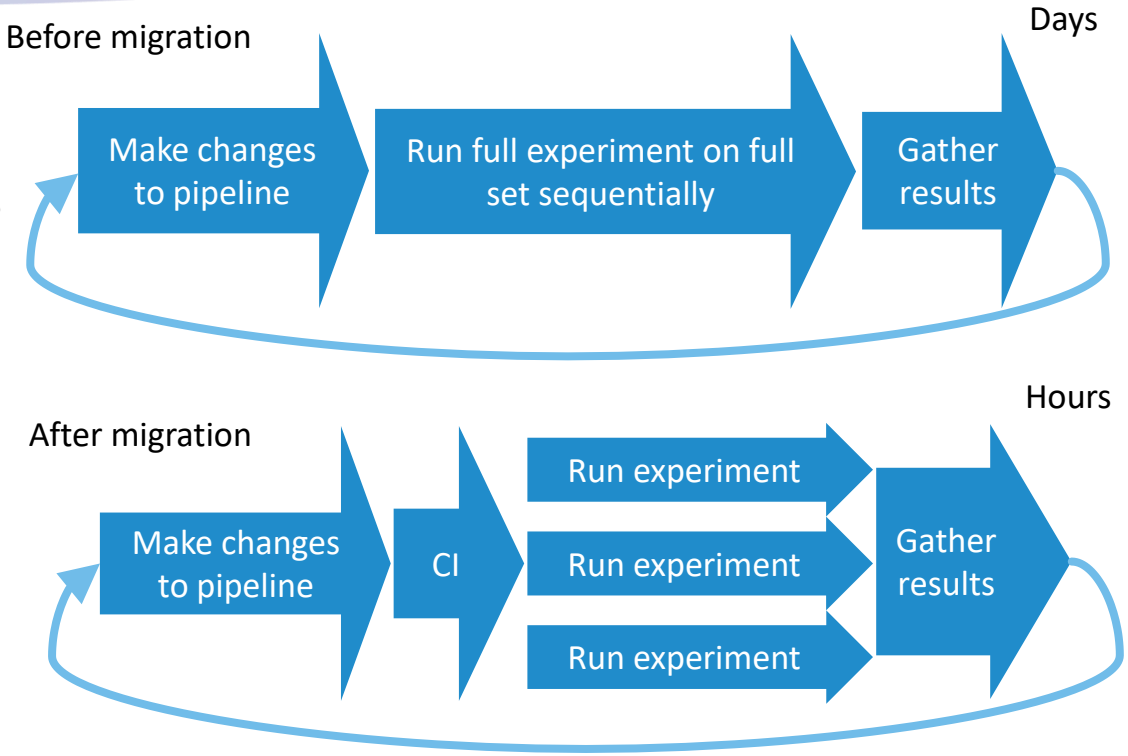
New approach – Research iterations



The speed of research

Speed of research improvements

- Flexibility
  - Easy to use different libraries/base container images
- Time to run experiments
  - Easy to make quick iterations and run experiments in parallel
- Easy to learn
  - New hires can run experiments from Day 1 and apply the changes to the workflow



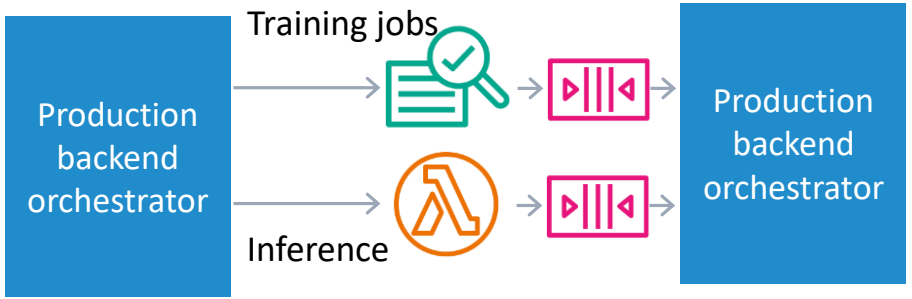
Lessons learned – Tooling

- Local dev has to support end-to-end workflow
- CI pipeline must be able to publish container images fast
- Research framework needs to handle parallel experiments

Productizing ML models

Productizing ML models

- Time to production
  - Easier to productize since infrastructure is similar
- Research production gap
  - Low gap because a big part of the stack is the same
- Managing production
  - The ML team can manage the ML production environment because it's similar to research



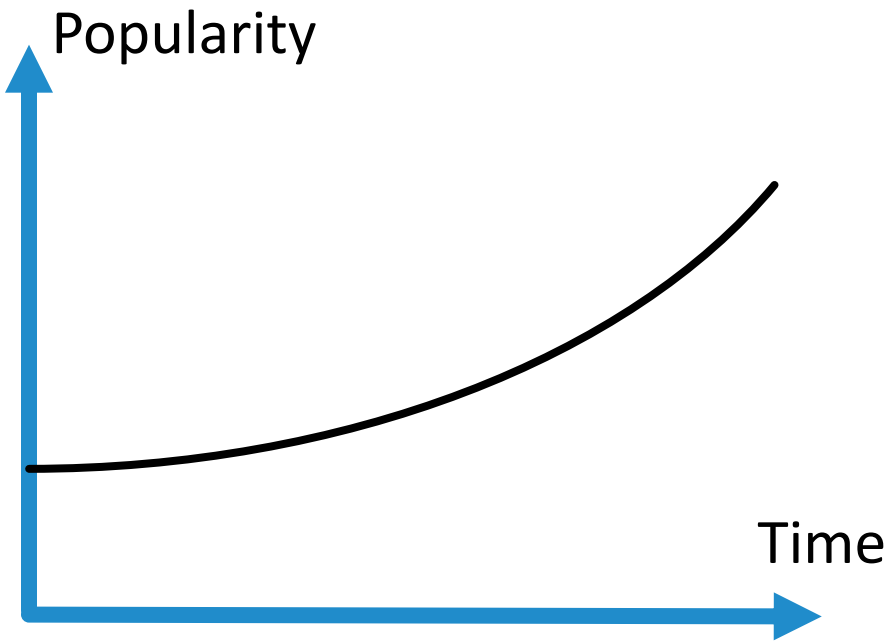
Lessons learned – Infrastructure

- Make it easy to deploy multiple types of container images
- Ensure models are backward compatible by using inference framework (e.g. ONNX)
- Ensure cloud environment is part of your Dev/CI stack to catch bugs earlier

Scaling production

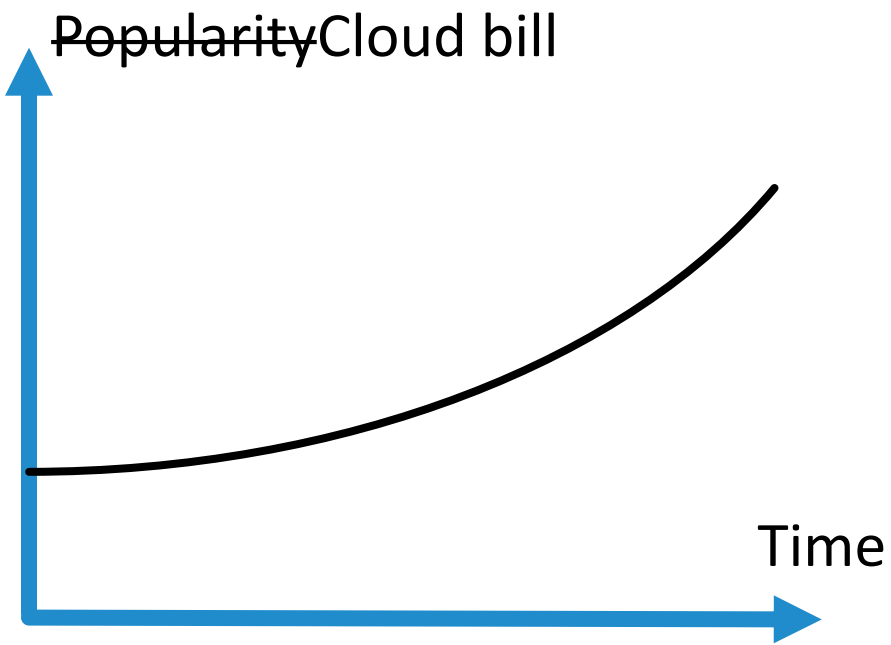
Scaling production

- Scaling 1x to 100x
  - Scaling in cloud case is about increasing account limits
- Cost optimization
  - Using spot instances enables to decrease costs
- Sharing ownership
  - Setting up metrics which could be used by different eng teams for monitoring ML workflows



Cost optimization

- Optimize instance size based on job type
- Switch between different GPU/TPU instances
- Use spot instances or savings plans
- Use external services for cheaper compute



Lessons learned – Design

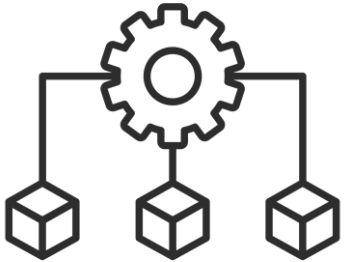
- Make it easy to change type of the hardware for the training job
- In case with cloud it’s important to proactively monitor account limits
- Handling peak loads inference and training jobs requires a different set of observability tools

Takeaways

Takeaways

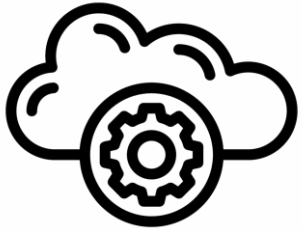
Tooling

- Local dev supports end-to-end workflow
- Optimize CI for quick OCI publish
- Parallel experiments



Infrastructure

- Support multiple OCIs
- Backward compatibility of models
- Serverless infrastructure is part of the CI process



Design

- Keep infrastructure flexible
- Increase limits in advance
- Plan for peak loads

Resources

Instrumental website
https://instrumental.com/

Open source projects with packages for serverless infrastructure
https://github.com/ryfeus/lambda-packs

https://github.com/ryfeus/stepfunctions2processing

My website
https://ryfeus.io