MLOps Playbook

Paul Brabban, Jon Carney, Simon Case,
Scott Cutts, Claudio Diniz, Bas Geerdink, Thorben Louw,
Jennifer Stark, Rajesh Thiagarajan

EQUALEXPERTS.COM/PLAYBOOKS/MLOPS

EQUAL EXPERTS

01 Introduction

3 Overview

02 Key terms

5 Overview

03 What is MLOps

6 Overview

04 Principles

8 Solid data foundations

10 Provide an environment that allows data scientists to create and test models

12 A machine learning service is a product

13 Apply continuous delivery

13 Evaluate and monitor algorithms throughout their lifecycle

14 Operationalising ML is a team effort

05 Practices

18 Collect performance data of the algorithm in production

19 Ways of deploying your model

23 How often do you deploy a model?

25 Keep a versioned model repository

25 Measure and proactively evaluate quality of training data

27 Testing through the ML pipeline

29 Business impact is more than just accuracy

31 Regularly monitor your model in production

33 Monitor data quality

34 Automate the model lifecycle

36 Create a walking skeleton/steel thread

38 Appropriately optimise models for inference

06 Explore

42 Feature stores

07 Pitfalls

44 User Trust and Engagement

47 Explainability

49 Avoid notebooks in production

52 Poor security practices

53 Don’t treat accuracy as the only or even the best way to evaluate your algorithm

53 Use machine learning judiciously

54 Don’t forget to understand the at-inference usage profile

54 Don’t make it difficult for data scientists to access data or use the tools they need

55 Not taking into consideration the downstream application of the model

Introduction

Machine learning, and by extension a lot of artificial intelligence, has grown from a niche idea to a tool that is being applied in multiple areas and industries.

At EE we have been involved in developing and deploying machine learning for a number of applications, including to:

- Assess cyber-risk
- Evaluate financial risk
- Improve search and recommendations in retail web sites
- Price used vehicles
- Improve logistics and supply chains

An ML solution depends on both the algorithm - which is code - and the data used to develop and train that algorithm. For this reason, developing and operating solutions that use ML components is different to standard software development.

INTRODUCTION
KEY TERMS
WHAT IS MLOPS?
PRINCIPLES
PRACTICES
EXPLORE
PITFALLS

This playbook brings together our experiences working with algorithm developers to make machine learning a normal part of operations. It won't cover the algorithm development itself - that is the work of the data scientists. Instead it covers what you need to consider when providing the architecture, tools and infrastructure to support their work and integrate their outputs into the business.

It is a common mistake to focus on algorithms - after all they are very clever, require deep expertise and insight and in some cases seem to perform miracles. But in our experience, obtaining business value from algorithms requires engineering to support the algorithm development part of the process alongside integrating the machine learning solution into your daily operations. To unlock this value you need to:

- Collect the data that drives machine learning and make it available to the data scientists who develop machine learning algorithms
- Integrate these algorithms into your everyday business
- Configuration control, deploy and monitor the deployed algorithms
- Create fast feedback loops to algorithm developers

As with all of our playbooks we have written this guide in the spirit of providing helpful advice to fellow developers creating ML solutions. If you are starting on the ML journey we hope you are not daunted by all the things covered in this playbook. Starting small and being lean in your implementation choices at the start is perfectly fine and will probably help you to iterate quicker.

Key terms

Machine learning (ML) - a subset of AI that involves training algorithms with data rather than developing hand-crafted algorithms. A machine learning solution uses a data set to train an algorithm, typically training a classifier that says what type of thing this data is (e.g. this picture is of a dog); a regressor, which estimates a value (e.g. the price of this house is £400,000.) or an unsupervised model,such as generative models like GPT-3, which can be used to generate novel text). Confusingly, when data scientists talk about regression it means something completely different than is meant when software engineers use the same terminology.

Model - in machine learning a model is the result of training an algorithm with data, which maps a defined set of inputs to outputs. This is different from the standard software use of the term 'data model' - which is a definition of the data entities, fields, relationships etc for a given domain, which is used to define database structures among other things.

Algorithm - we use this term more or less interchangeably with model. (There are some subtle differences, but they're not important and using the term 'algorithm' prevents confusion with the other kind of data models).

Ground-truth data - a machine-learning solution usually needs a data set that contains the input data (e.g. pictures) along with the associated answers (e.g. this picture is of a dog, this one is of a cat) - this is the 'ground-truth.'

Labelled data - means the same as ground-truth data.

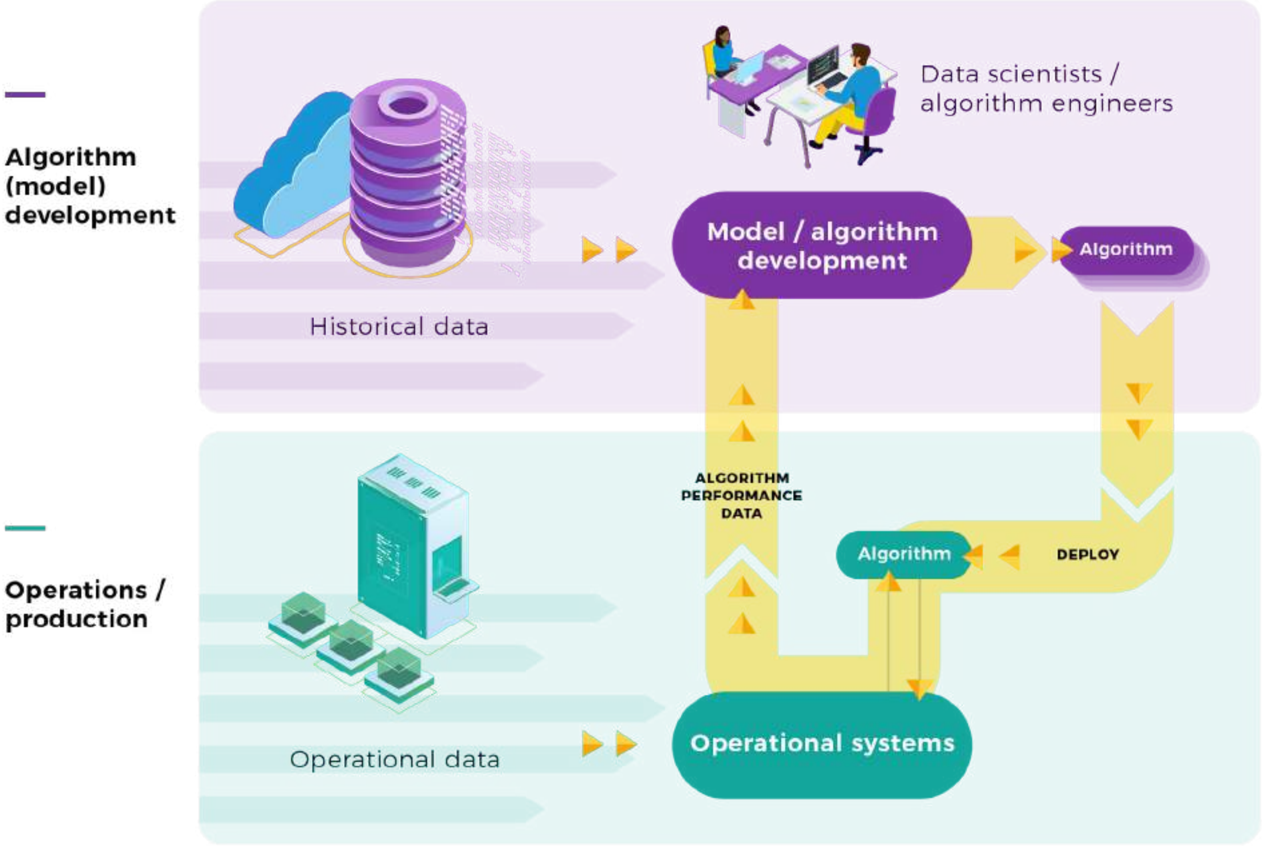
INTRODUCTION
KEY TERMS
WHAT IS MLOPS?
PRINCIPLES
PRACTICES
EXPLORE
PITFALLS

What is MLOps?

Operationalising Machine Learning is a dual track process:

1 | Data Scientists work on historical data to create algorithms

2 | ML engineering teams integrate the model into operational systems and data flows


Algorithm (model) development
Operations / production
Data scientists / algorithm engineers
Historical data
Model / algorithm development
Algorithm
ALGORITHM PERFORMANCE DATA
Algorithm
DEPLOY
Operational data
Operational systems

INTRODUCTION
KEY TERMS
WHAT IS MLOPS?
PRINCIPLES
PRACTICES
EXPLORE
PITFALLS

To create and use a machine learning algorithm you will typically go through the following steps:

1 Initial development of algorithm

The first step in applying a machine learning solution to your business is to develop the model. Typically, data scientists understand what the need is, then identify or create ground-truthed data sets and explore the data. They prototype different machine learning approaches and evaluate against hold-out data sets (the test sets) to gain an understanding of the performance of the algorithm against unseen data.

2 Integrate / deploy model

Once a model has been selected and shown to meet the required performance criteria, it needs to be integrated into the business. There are a variety of methods of integration and the right option will depend on the consuming services. In modern architecture, for example, the model is likely to be implemented as a standalone microservice.

The environment in which data scientists create the model is not the one in which you want to deploy it, so you need to have a mechanism for deploying it - copying an approved version of the algorithm into the operational environment.

3 Monitor performance

The model needs to be monitored when in operation. As well as monitoring the same things you would do for any piece of software - that it is running, that it scales to meet demand etc., you will also want to monitor what accuracy it is providing when deployed in practice.

4 Update model

In most cases the model will be retrained on a regular basis - in which case the different iterations of the model need to be version controlled and downstream services directed towards the latest version. Better performing versions of the model will become available as a result of new data or more development effort on the model itself, and you will want to deploy these into production. Model updates are usually the result of two different sorts of changes:

- Retrain on new data - in most cases the business collects more ground-truthed data that can be used to retrain the model and improve its accuracy. In this case no new data sources are needed and there is no change to the interfaces between the model and its data sources and no change to the interface with operational systems.
- Algorithm change - sometimes new data sources become available which improve the performance of the algorithm. In this case the interfaces to the model will change and new data pipelines need to be implemented. Sometimes, additional outputs are required from the model, and the interface to downstream operational systems is altered.

Principles

Solid data foundations

> Have a store of good quality, ground-truth (labelled) historical data that is accessible by your data scientists

A machine learning solution is fundamentally dependent on the data used to train it. To maintain and operate an ML solution, the data used to develop the model/algorithm must be available to the maintainers. They will need the data to monitor performance, validate continued performance and find improvements. Furthermore, in many cases the algorithm is modelling an external world that is undergoing change, and they will want to update or retrain the model to reflect these changes, so will need data updates.

The data needs to be accessible by data science teams and it will also need to be made available to automated processes that have been set-up for retraining the model.

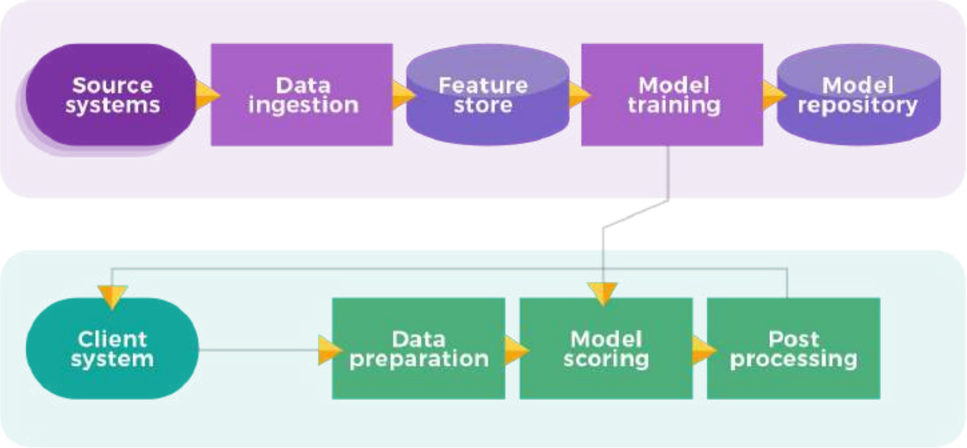
In most applications of ML, ground-truth data will need to be captured alongside the input data and it is essential to capture these data points as well.

It is common to create data warehouses, data lakes or data lakehouses and associated data pipelines to store this data.

Our data pipelines playbook covers our approach to providing this data.

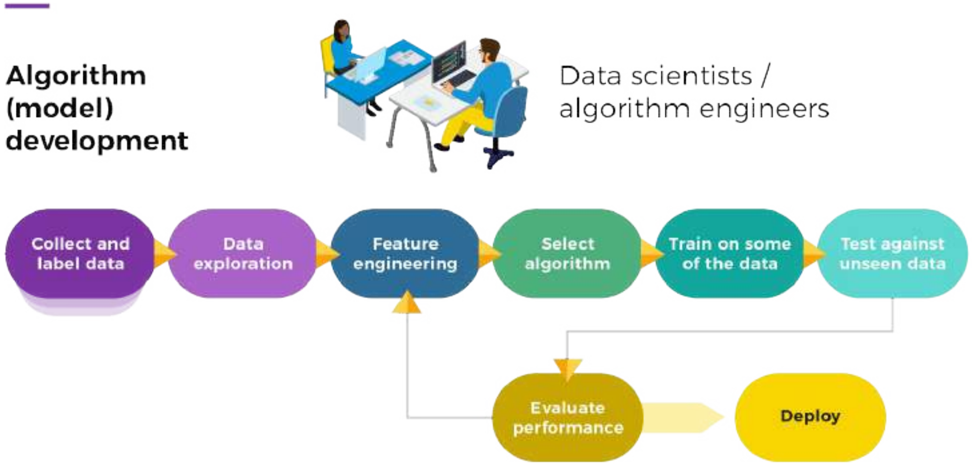
The below diagram shows the two processes involved in building machine learning systems and the data they need to access:

- An evaluation process that makes predictions (model scoring). This may be real-time.
- A batch process that retrains the model, based on fresh historical data.



Provide an environment that allows data scientists to create and test models

Developing a machine learning model is a creative, experimental process. The data scientists need to explore the data and understand the features/fields in the data. They may choose to do some feature engineering - processing on those fields - perhaps creating aggregations such as averages over time, or combining different fields in ways that they believe will create a more powerful algorithm. At the same time they will be considering what the right algorithmic approach should be - selecting from their toolkit of classifiers, regressors, unsupervised approaches etc. and trying out different combinations of features and algorithms against the datasets they have been provided with. They need a set of tools to explore the data, create the models and then evaluate their performance.



Algorithm (model) development

Data scientists / algorithm engineers

Collect and label data

Data exploration

Feature engineering

Select algorithm

Train on some of the data

Test against unseen data

Evaluate performance

Deploy

Ideally, this environment should:

- Provide access to required historical data sources (e.g. through a data warehouse or similar).
- Provide tools such as notebooks to view and process the data.
- Allow them to add additional data sources of their own choosing (e.g. in the form of csv files).
- Allow them to utilise their own tooling where possible e.g. non-standard python libraries.
- Make collaboration with other data scientists easy e.g. provide shared storage or feature stores.
- Have scalable resources depending on the size of the job (e.g. in AWS Sagemaker you can quickly specify a small instance or large GPU instance for deep learning).
- Be able to surface the model for early feedback from users before full productionisation.

Some of the approaches we have successfully used are:

- Development on a local machine with an IDE or notebook.
- Development on a local machine , deployment and test on a local container and run in a cloud environment.
- Using cloud first solutions such as AWS Sagemaker or GCP Collab.
- Using dashboarding tools such as Streamlit and Dash to prototype and share models with end users.

Local development using an IDE may lead to better structured code than with a notebook, but make sure that the data is adequately protected (data with PII should not be handled in this way), and that the dependencies needed to run the model are understood and captured.

Taking a container approach eases the promotion of a development model to production and often reduces turn-around times across the full model lifecycle.

A machine learning service is a product

We believe that an ML service should be developed and treated as a product, meaning that you should apply the same behaviours and standards as you would when developing any other software product.

These behaviours include:

- Identify, profile and maintain an active relationship with the end-users of your ML service. Work with your users to identify requirements that feed into your development backlog, involve your users in validation of features and improvements, notify them of updates and outages, and in general, work to keep your users happy.
- Maintain a roadmap of features and improvements. Continue to improve your service throughout its lifetime.
- Provide good user documentation.
- Actively test your service.
- Capture the iterations of your service as versions and help users migrate to newer versions. Clearly define how long you will support versions of your service, and whether you will run old and new versions concurrently.
- Understand how you will retire your service, or support users if you choose not to actively maintain it any longer.
- Have an operability strategy for your service. Build in telemetry that is exposed through monitoring and alerting tools, so you know when things go wrong. Use this data to gain an understanding of how your users actually use your service.
- Define who is supporting your service and provide runbooks that help support recovery from outages.
- Provide a mechanism for users to submit bugs and unexpected results, and work toward providing fixes for these in future releases.

Apply continuous delivery

Machine learning solutions are complex software and should use best practice

We want to be able to amend how our machine learning models consume data and integrate with other business systems in an agile fashion as the data environment, downstream IT services and needs of the business change. Just like any piece of working software, continuous delivery practices should be adopted in machine learning to enable regular updates of those integrations in production. Teams should adopt typical continuous delivery techniques, use Continuous Integration and Deployment (CI/CD) approaches; utilise Infrastructure as Code (Terraform, ansible, packer, etc.) and work in small batches to have fast and reasonable feedback, which is key to keeping a continuous improvement mindset.

Evaluate and monitor algorithms throughout their lifecycle

ML solutions are different from standard software delivery because we want to know that the algorithm is performing as expected, as well as all the things we monitor to ensure the software is working correctly. In machine learning, performance is inherently tied to the accuracy of the model. Which measure of accuracy is the right one is a non-trivial question - which we won't go into here except to say that usually the Data Scientists define an appropriate performance measure.

This performance of the algorithm should be evaluated throughout its lifecycle:

- During the development of the model - it is an inherent part of initial algorithm development to measure how well different approaches work, as well as settling on the right way to measure the performance.
- At initial release - when the model has reached an acceptable level of performance, this should be recorded as a baseline and it can be released into production.
- In production - the algorithm performance should be monitored throughout the lifetime to detect if it has started performing badly as a result of data drift or concept drift.

MLOps is a team effort

Turning a model from a prototype to an integrated part of the business requires a cross-functional team working closely together.

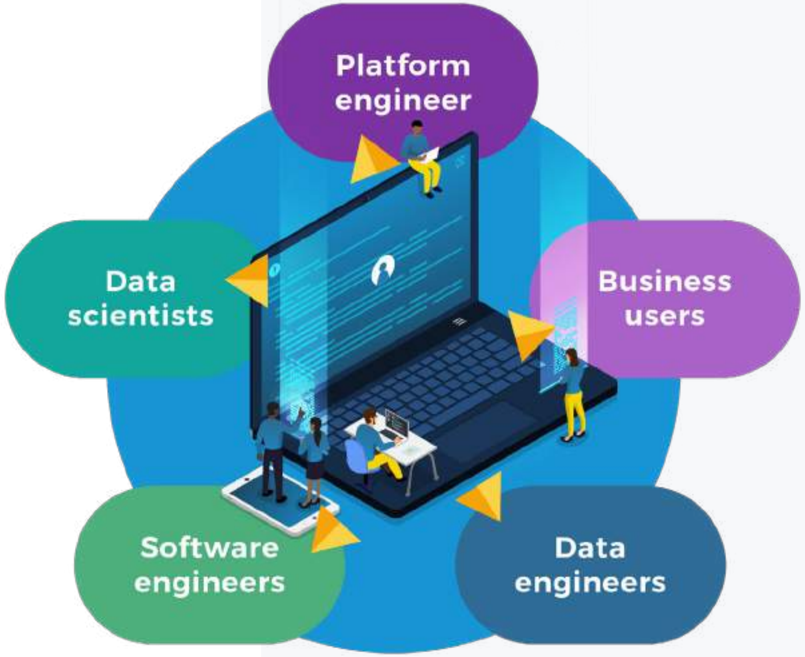
You will need:

- Platform/Machine Learning engineer(s) to provide the environment to host the model.
- Data engineers to create the production data pipelines to retrain the model.
- Data scientists to create and amend the model.
- Software engineers to integrate the model into business systems (e.g. a webpage calling a model hosted as a microservice)

MLOps is easier if everyone has an idea of the concerns of the others. Data Scientists are typically strong at mathematics and statistics, and may not have strong software development skills. They are focused on algorithm performance and accuracy metrics. The various engineering disciplines are more concerned about testing, configuration control, logging, modularisation and paths to production (to name a few).

It is helpful if the engineers can provide clear ways of working to the data scientist early in the project. It will make it easier for the data scientists to deliver their models to them. How do they want the model/algorithm code delivered (probably not as a notebook)? What coding standards should they adhere to? How do you want them to log? What tests do you expect? Create a simple document and spend a session taking them through the development process that you have chosen. Engineers should recognise that the most pressing concern for data scientists is prototyping, experimentation and algorithm performance evaluation.

When the team forms, **recognise that it is one team and organise yourself accordingly**. Backlogs and stand-ups should be owned by and include the whole team.



Experience report

I started as a data scientist but quickly realised that if I wanted my work to be used I would need to take more interest in how models are deployed and used in production, which has led me to move into data engineering and ML Operations, and now this has become my passion! There are many things that I have learned during this transition.

In general, models are developed by data scientists. They have the maths and stats skills to understand the data and figure out which algorithms to use, whilst the data engineers deploy the models. New features can get added by either of these groups.

In my experience, data scientists usually need to improve their software development practices. They need to become familiar with the separation of environments (e.g. development, staging, production) and how code is promoted between these environments. I'm not saying they should become devops experts, but algorithms are software and if the code is bad or if it can't be understood then it can't be deployed or improved. Try to get your code out of the notebook early, and don't wait for perfection before thinking about deployment. The more you delay moving into production, the more you end up with a bunch of notebooks that you don't understand. Right now I'm working with a great data scientist and she follows the best practice of developing the code in Jupyter Notebooks, and then extracts the key functionality into libraries which can be easily deployed.

For data engineers - find time to pair with data scientists and share best dev practices with them. Recognise that data science code is weird in many respects - lots of stuff is done with Data Frames or similar structures, and will look strange compared to traditional application programming. It will probably be an easier experience working with the data scientists if you understand that they will be familiar with the latest libraries and papers in Machine Learning, but not with the latest software dev practices. They should look to you to provide guidance on this - try to provide it in a way that recognises their expertise!

Matteo Guzzo
Data specialist

Equal Experts, EU

Experience report

As the lead data scientist in a recent project, my role was to create an algorithm to estimate prices for used vehicles. There was an intense initial period where I had to understand the raw data, prototype the data pipelines and then create and evaluate different algorithms for pricing. It was a really intense time and my focus was very much on data cleaning, exploration and maths for the models.

We worked as a cross-functional team with a data engineer, UX designer and two user interface developers. Wehad shared stand-ups; and the data engineering, machine learning and user experience were worked on in parallel.

I worked closely with our data engineer to develop the best way to deploy the ETL and model training scripts as data pipelines and APIs. He also created a great CI/CD environment and set up the formal ways of working in this environment, including how code changes in git should be handled and other coding practices to adopt. He paired with me on some of the initial deployments so I got up to speed quickly in creating production-ready code. As a data scientist I know there are 100 different ways someone can set up the build process - and I honestly don't have any opinion on which is the right way. I care about the performance of my model! I really appreciated working together on it - that initial pairing meant that we were able to bring all our work together very quickly and has supported the iteration of the tool since then.

Adam Fletcher

Data scientist

Equal Experts, UK

Practices

INTRODUCTION

KEY TERMS

WHAT IS MLOPS?

PRINCIPLES

PRACTICES

EXPLORE

PITFALLS

Collect performance data of the algorithm in production and make it accessible to your data scientists

Deciding on the right way to evaluate the performance of an algorithm can be difficult. It will, of course, depend on the purpose of the algorithm. Accuracy is an important measure but will not be the only or even the main assessment of performance. And even deciding how you measure accuracy can be difficult.

Furthermore, because accurate measures of performance require ground-truth data it is often difficult to get useful performance measures from models in production - but you should still try.

Some successful means of collecting the data that we have seen are:

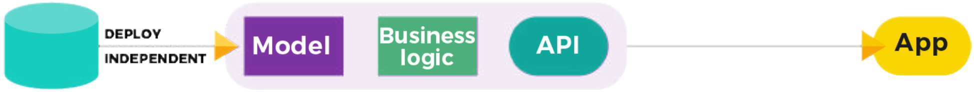
> **A/B testing** - In A/B testing you test different variations of a model and compare how the variations perform, or you compare how a model performs against the absence of a model, like the statistical Null Hypothesis testing. To make effective comparisons between two groups, you'll need to orchestrate how it will happen with the production models, because the usage of models is split. For example, if the models are deployed in APIs, the traffic for the models can be routed 50%. If your performance metric is tied to existing statistics (e.g. conversion rates in e-commerce) then you can use A/B or multivariant testing.

> **Human in the loop** - this is the simplest technique of model performance evaluation,but requires the most manual effort. We save the predictions that are made in production. Part of these predictions are classified by hand and then model predictions are compared with the human predictions.
>
> In some use-cases (e.g. fraud) machine-learning acts as a recommender to a final decision made by a human. The data from their final decisions can be collected and analysed for acceptance of algorithm recommendations.

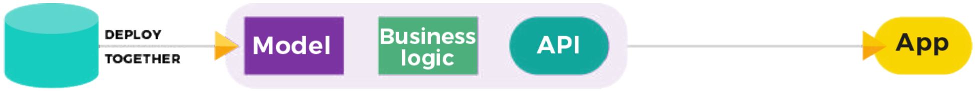
> **Periodic Sampling** - if there is no collection of ground-truth in the system then you may have to resort to collection of samples and hand-labelling to evaluate the performance in a batch process.

Ways of deploying your model

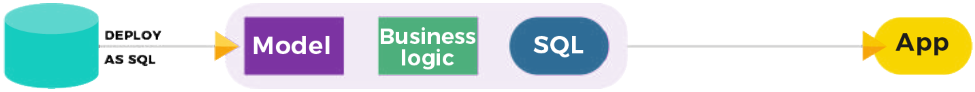
Microservices: API-ify your model (Pickle, Joblib)



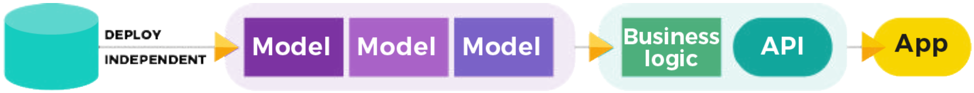
Deploy model together with your application (Python, MLlib)



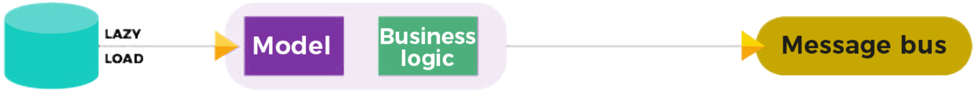
Deploy model as SQL stored procedure



Shared service: host your model in a dedicated tool, possibly automated



Streaming: load your model in memory (PMML, ONNX)



Many cloud providers and ML tools provide solutions for model deployment that integrate closely with their machine learning and data environments. These can greatly speed up deployment and ease infrastructure overhead such as:

- GCP Vertex AI
- AWS Sagemaker
- MLFlow

Once a machine learning model has been generated, that code needs to be deployed for usage. How this is done depends on your use case and your IT environment.

	WHY	HOW	WATCH OUT
AS A MICRO-SERVICE	Your model is intended to provide output for a real time synchronous request from a user or system.	The model artefact and accompanying code to generate features and return results is packaged up as a containerised microservice.	The model and microservice code should always be packaged together - this avoids potential schema or feature generation errors and simplifies versioning and deployment. Your model and feature generation code will need to be performant in order to respond in real time and not cause downstream timeouts. It will need to handle a wide range of possible inputs from users.
EMBEDDED MODELS	Your model is intended to directly surface its result for further usage in the context it is embedded e.g. in an application for viewing.	The model artefact is packaged up as part of the overall artefact for the application it is contained within, and deployed when the application is deployed.	The latest version of the model should be pulled in at application build time, and covered with automated unit, integration and end-to-end tests. Realtime performance of the model will directly affect application response times or other latencies.
AS A SQL STORED PROCEDURE	The model output is best consumed as an additional column in a database table. The model has large amounts of data as an input (e.g. multi-dimensional time-series).	The model code is written as a stored procedure (in SQL, Java, Python, Scala etc. dependent on the database) and scheduled or triggered on some event (e.g. after a data ingest). Modern data warehouses such as Google BigQueryML or AWS RedShift ML can train and run ML models as a table-style abstraction.	Stored procedures not properly configuration controlled. Lack of test coverage of the stored procedures.
AS PART OF BATCH DATA PIPELINE	Your model is intended to provide a set of batch predictions or outputs against a batched data ingest or a fixed historical dataset.	The model artefact is called as part of a data pipeline and writes the results out to a static dataset. The artefact will be packaged up with other data pipeline code and called as a processing step via an orchestration tool. See our data pipeline playbook for more details.	Feature generation can be rich and powerful across historical data points Given the lack of direct user input, the model can rely on clean, normalised data for feature generation. Parallelisation code for model execution may have to be written to handle large datasets.
AS PART OF A STREAMING DATA PIPELINE	Your model is used in near-real-time data processing applications, for example in a system that makes product recommendations on a website while the users are browsing through it.	The model artefact is served in memory in the streaming processing framework, but using an intermediate format such as ONNX or PMML. The artefact is deployed while the stream keeps on running, by doing rolling updates.	Performance and low latency are key. Models should be developed with this in mind; it would be good practice to keep the number of features low and reduce the size of the model.

Experience report

I worked on a model that was used to forecast aspects of a complex logistics system. The input data was a large number (many thousands) of time-series and we needed to create regular forecasts going into the future for some time, so that downstream users could plan their operations and staffing appropriately. There was a lot of data involved at very high granularity, so it was a complex and time-consuming calculation. However, there was no real-time need and forecast generation once a week was more than enough to meet the business needs.

In this context the right approach was to use a batch-process in which forecasts were generated for all parts of the logistics chain that needed them. These were produced as a set of tables in Google BiqQuery. I really liked this method of sharing the outputs because it gave a clean interface for downstream use.

One of the challenges in this work was the lack of downstream performance measures. It was very hard to get KPIs in a timely manner. Initially we measured standard precision errors on historical data to evaluate the algorithm and later we were able to augment this with A/B testing by splitting the logistics network into two parts.

Katharina Rasch
Data engineer

Equal Experts, EU

How often do you deploy a model?

Coming up with a good model for your data once is hard enough, but in practice, you will need to retrain and deploy updates to your model – perhaps regularly! These are necessary because:

- the data used to train your model changes in nature over time,
- you discover better models as part of your development process, or
- because you need to adapt your ML models to changing regulatory requirements

Two useful phrases help to describe the way data changes are

Data drift describes the way data changes over time (e.g. the structure of incoming data involves new fields, or changes in the previous range of values you originally trained against) perhaps because new products have been added or upstream systems stop populating a specific field.

Concept drift means that the statistical nature of the target variables being predicted might change over time. You can think of examples such as an ML-enhanced search service needing to return very different results for “chocolates” at Valentine’s day versusEaster, or a system that recognises that users’ fashions and tastes change over time, so the best items to return won’t be the same from season to season. Processes that involve human nature are likely to result in concept drift.

Measure drift over time to understand when a model’s accuracy is no longer good enough and needs to be retrained.

> It’s also good practice to regularly redeploy your model, even if you haven’t improved it or noticed changes in data characteristics! This allows you to make use of new framework versions and hardware, to address security vulnerabilities through updates to components, and to be sure that when you **need** to deploy a fresh model, you know that your deployment processes work.

Experience report

In one engagement with a client who was a leader in the travel industry, we had used data from the past five years to build a prediction model of repurchase behaviour. The model had good accuracy and was running well in production.

Travel behaviours exhibited sudden and drastic change from March 2020, when the whole world reacted to the rapid spread of "SARS-Cov-2" by closing borders. The data that the model had been trained on had absolutely no pattern of what was happening in real life. We realised that continuing to use the model output would not be useful.

The team changed the model to factor in the border closures to effect on the predictions. We also incorporated a signal analyser into the model, which constantly monitored incoming data for a return to normalcy. It was changed to identify data patterns which matched the pre-covid historical data so that the model can switch off the dependency on specific Covid-related external data, when conditions return to normal.

Uttam Kini

Principal consultant

Equal Experts, India

Keep a versioned model repository

In some cases you will want the ability to know why a decision was made, for example, if there is an unexpected output or someone challenges a recommendation. Indeed, in most regulated environments it is essential to be able to show how a given decision or recommendation was reached, so you know which version of your machine learning model was live when a specific decision was made. To meet this need you will need a store or repository of the models that you can query to find the version of the model in use at a given date and time.

In the past we have used a variety of ways to version our models:

- S3 buckets with versioning enabled
- S3 buckets with database to to store model metadata
- MLflow model registry
- DVC to version both the model and the data used to create that model
- Cloud provider model registries (AWS Sagemaker, Google Vertex AI , Azure MLOps)
- Some models can have their coefficients stored as text, which is versioned in Git

Measure and proactively evaluate quality of training data

ML models are only as good as the data they're trained on. In fact, the quality (and quantity) of training data is often a much bigger determiner of your ML project's success than the sophistication of the model chosen. Or to put it another way: sometimes it pays much more to go and get better training data to use with simple models than to spend time finding better models that only use the data you have.

To do this deliberately and intentionally, you should be constantly evaluating the quality of training data.

You should try to:

- Identify and address class imbalance (i.e. find 'categories' that are underrepresented).

- Actively create more training data if you need it (buy it, crowdsource it, use techniques like image augmentation to derive more samples from the data you already have).

- Identify statistical properties of variables in your data, and correlations between variables, to be able to identify outliers and any training samples that seem wrong.

- Have processes (even manual random inspection!) that check for bad or mislabelled training samples. Visual inspection of samples by humans is a good simple technique for visual and audio data.

- Verify that distributions of variables in your training data accurately reflect real life. Depending on the nature of your modelling, it’s also useful to understand when parts of your models rely on assumptions or beliefs (“priors”), for example the assumption that some variable has a certain statistical distribution. Test these beliefs against reality regularly, because reality changes!

- Find classes of input that your model does badly on (and whose poor performance might be hidden by good overall “evaluation scores” that consider the whole of the test set). Try to supplement your training data with more samples from these categories to help improve performance.
- Ideally, you should also be able to benchmark performance against your dataset rather than aim for getting metrics ‘as high as possible’. What is a reasonable expectation for accuracy at human level, or expert level? If human experts can only achieve 70% accuracy against the data you have, developing a model that achieves 75% accuracy is a terrific result! Having quantitative benchmarks against your data can allow you to know when to stop trying to find better models, and when to start shipping your product.

Testing through the ML pipeline

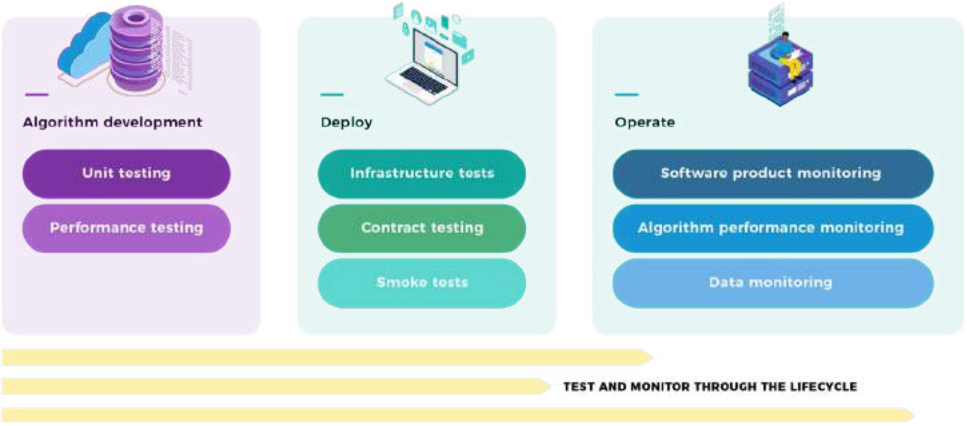
As with any continuous delivery development, an ML pipeline needs to be testable and tested. An ML pipeline is a complex mixture of software (including complex mathematical procedures), infrastructure, and data storage and we want to be able to rapidly test any changes we make before promoting to production.

We have found the following test types to be valuable:

- **Contract testing** - if the model is deployed as a microservice endpoint, then we should apply standard validations of outputs to inputs.
- **Unit testing** - many key functions such as data transformations, or mathematical functions within the ML model are stateless and can be easily covered by unit-tests.
- **Infrastructure tests** - e.g. Flask/FastAPI models start and shutdown.
- **'ML smoke test'** - we have found it useful to test deployed models against a small set of known results. This can flush out a wide range of problems that may occur. We don't recommend a large number - around five is usually right. For some types of model e.g. regression models the result will change every time the model is trained so the test should check the result is within bounds rather than a precise result.

In addition to the tests above, which are typical for any complex piece of software, the performance of the model itself is critical to any machine learning solution. Model performance testing is undertaken by data scientists on an ad-hoc basis throughout the initial prototyping phase. Before a new model is released you should validate that the new model performs at least as well as the existing one. Test the new model against a known data set and performance compared to a specified threshold or against previous versions.

We don't usually do load testing on our models as part of the CI/CD process. In a modern architecture load is typically handled by auto-scaling so we usually monitor and alert rather than test. In some use cases, such as in retail where there are days of peak demand (e.g. Black Friday), load testing takes place as part of the overall system testing.



Experience report

> You can always do more tests! But there are usually time pressures which mean you have to prioritise which ones you do. I think there is a 'hierarchy of needs' for tests which you try to move up depending on your backlog.'

When I was working on recommender systems for retail we had different tests for different parts of the model development and retraining. In the initial development we used the classic data science approach of splitting the data into train and test sets, until we had reached a model with a sufficient baseline performance to deploy. However, once we were in production all our data was precious and we didn't want to waste data unnecessarily so we trained on everything. Like any piece of software, I developed unit tests around key parts of the algorithm and deployment.

I also created functional tests - smoke tests which tested that an endpoint deployed and that the model responded in the right way to queries, without measuring the quality of the recommendations. Our algorithms were deployed within an A/B/multi-variant testing environment so we have an understanding that we are using the best performant algorithm at least.

We found that the Vertex AI auto-scaling was not as performant as we had hoped - and noticed some issues which affected our ability to meet demand. Now we do stress testing for each model and for each new version of the model.

Khalil Chourou
Data engineer

Equal Experts, EU

Business impact is more than just accuracy - understand your baseline

When working on an initiative that involves cutting edge technology like AI/ML, it is very easy to get blind sided by the technological aspects of the initiative. Discussion around algorithms to be used, the computational power, speciality hardware and software, bending data to the will and opportunities to reveal deep insights will lead to the business stakeholders having high expectations bordering on magical outputs.

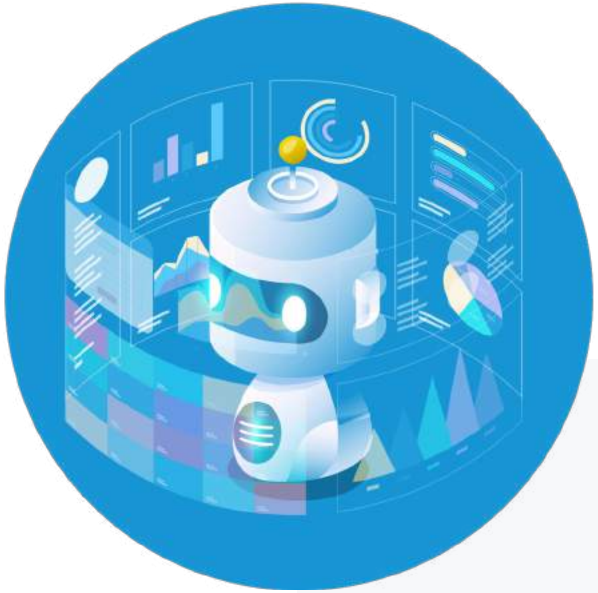
The engineers in the room will want to get cracking as soon as possible. Most of the initiatives will run into data definition challenges, data availability challenges and data quality issues. The cool tech, while showing near miraculous output as a “proof of concept” will start falling short of the production level expectations set by the POC stage, thereby creating disillusionment.

To avoid this disillusionment, it is important at the beginning of the initiative to start detailing the business metrics that would be affected. Then the team and the stakeholders have to translate them into the desired level of accuracy or performance output from the ML based on an established base line. The desired level of accuracy can be staggered in relation to the quantum of business outcome (impact on the business metrics) to define a hurdle rate beyond which it would be acceptable.

Rather than choosing an obsolete or worse, a random accuracy level that may not be possible because of various factors that the team cannot control, this step ensures that they will be able to define an acceptable level of performance, which translates to valuable business outcome.

The minimum acceptable accuracy or performance level (hurdle rate) would vary depending on the use case that is being addressed. An ML model that blocks transactions based on fraud potential would need very high accuracy when compared to a model built to predict repeat buy propensity of a customer that helps marketers in retargeting.

Without this understanding, the team working on the initiative won’t know if they are moving in the right direction. The team may go into extended cycles of performance / accuracy improvement assuming anything less is not acceptable, while in reality they could have generated immense business value just by deploying what they have.



Experience report

At the start of a project to use machine learning for product recommendation, business stakeholders were using vague terminologies to define the outcomes of the initiative. They were planning downstream activities that would use the model output with the assumption that the model would accurately predict the repurchase behaviour and product recommendations, as if it can magically get it right all the time. They did not account for the probabilistic nature of the model predictions and what needs to be done to handle the ambiguities.

During inception, the team took time to explain the challenges in trying to build a model to match their expectations, especially when we could show them that they had limited available data and even where the data was available, the quality was questionable.

We then explored and understood their "as-is" process. We worked with them to establish the metrics from that process as the current baseline and then arrived at a good enough (hurdle rate) improvement for the initiative that can create significant business outcomes. During these discussions we identified the areas where the predictions were going to create ambiguous downstream data (e.g. although the model can predict with high enough accuracy who will buy again, the model can only suggest a basket of products that the customer would buy instead of the one specific product that the business users were initially expecting).

As the business understood the constraints (mostly arising out of the data availability or quality), they were able to design the downstream processes that could still use the best available predictions to drive the business outcome.

The iterative process, where we started with a base-line and agreed on acceptable improvement, ensured that the data team was not stuck with unattainable accuracy in building the models. It also allowed the business to design downstream processes to handle ambiguities without any surprises. This allowed the initiative to actually get the models live into production and improve them based on real world scenarios rather than getting stuck in long hypothetical goals.

Rajesh Thiagarajan
Principal consultant

Equal Experts, India

Regularly monitor your model in production

There are two core aspects of monitoring for any ML Solution:

- Monitoring as a software product
- Monitoring model accuracy and performance

Realtime or embedded ML solutions need to be monitored for errors and performance just like any other software solution. With autogenerated ML solutions this becomes essential - model code may be generated that slows down predictions enough to cause timeouts and stop user transactions from processing.

Monitoring can be accomplished by using existing off the shelf tooling such as Prometheus and Graphite.

You would ideally monitor:

- Availability
- Request/Response timings
- Throughput
- Resource usage

Alerting should be set up across these metrics to catch issues before they become critical.

ML models are trained on data available at a certain point in time. Data drift or concept drift (see How often do you deploy a model?) can affect the performance of the model. So it's important to monitor the live output of your models to ensure they are still accurate against new data as it arrives. This monitoring can drive when to retrain your models, and dashboards can give additional insight into seasonal events or data skew.

- Precision/Recall/F1 Score
- Model score or outputs
- User feedback labels or downstream actions
- Feature monitoring (Data Quality outputs such as histograms, variance, completeness)

Alerting should be set up on model accuracy metrics to catch any sudden regressions that may occur. This has been seen on projects where old models have suddenly failed against new data (fraud risking can become less accurate as new attack vectors are discovered), or an auto ML solution has generated buggy model code. Some ideas on alerting are:

- % decrease in precision or recall.
- variance change in model score or outputs.
- changes in dependent user outputs e.g. number of search click throughs for a recommendation engine.

Experience report

For a fraud detection application, we adopted the usual best practices of cross validation training set with an auto-ML library for model selection. The auto-ML approach yielded a good performing model to start, albeit rather inscrutable for a fraud detection setting. Our primary objective was to build that path to live for the fraud scoring application. We followed up shortly thereafter with building model performance monitoring joining live out-of-sample scores with fraud outcomes based on precision, recall and f1 measures tracked in Grafana. Observability is vital to detect model regression - when the live performance degrades consistently below what the model achieved during training and validation.

It became clear that we were in an adversarial situation in which bad actors would change their attack patterns, which was reflected in data drift of the model inputs and consequent concept drift.

> The effort invested in developing the model pipeline and performance monitoring allowed us to detect this drift rapidly and quickly iterate with more interpretable models and better features.

Austin Poulton
Data scientist

Equal Experts, South Africa

Monitor data quality

Data quality is fundamental for all ML products. If the data suffers from substantial quality issues the algorithm will learn the wrong things from the data. So we need to monitor that the values we're receiving for a given feature are valid.

Some common data quality issues we see are:

- missing values - fields are missing values.
- out of bound values - e.g. negative values or very low or high values.
- default values - e.g. fields set to zero or dates set to system time (1 jan 1900).
- format changes - e.g. a field which has always been an integer changes to float.
- changes in identifiers for categorical fields - e.g. GB becomes UK for a country identifier.

When training or retraining you need a strategy for handling data records with quality issues. The simplest approach is to filter out all records which do not meet your quality criteria, but this may remove important records. If you take this approach you should certainly look at what data is being discarded and find ways to resolve, if possible.

Other approaches are possible - for missing or incorrect fields we often follow the standard practice of imputing missing or clearly incorrect values. Where we impute values we typically record this in an additional column.

In cases where you can't disentangle a data error from a real entry (e.g. data sets where Jan 1900 could be a real data point) you may have to filter out good data points or investigate individually.

Automate the model lifecycle

As with any modern software development process, we eliminate manual steps where possible, to reduce the likelihood of errors happening. For ML solutions we make sure there is a defined process for moving a model into production and refreshing as needed. (Note that we do not apply this automation to the initial development and prototyping of the algorithms as this is usually an exploratory and creative activity.)

For an algorithm which has been prototyped, and accepted into production the life-cycle is:

- Ingest the latest data.
- Create training and test sets.
- Run the training.
- Check performance meets the required standard.
- Version and redeploy the model.

In a fully automated lifecycle this process is repeated either on a schedule or triggered by the arrival of more recent data with no manual steps.

There are a variety of tools and techniques to help with this. Some of the tools we have found useful include:

- MLFlow
- AWS Sagemaker
- GCP Vertex AI

Ingest data
Create training and test sets
Run training
Check performance
Version control model
Deploy

Experience report

When creating a pricing estimation service for one of our clients we were working from a blank canvas in terms of ML architecture. We knew that the model was going to be consumed by a web application so we could deploy as a microservice, and that data came in weekly batches with no real-time need for training.

We used a combination of S3 with versioning as our model store, and S3 event notifications, Lambdas, Fargate and Amazon Load Balancer to automate the data ingest, provisioning and update of two models, using CloudWatch to log the operations. The process is fully automated and triggered by the arrival of a weekly data drop into an S3 bucket.

> We took a lean approach using standard AWS services to create a platform able to ingest new data, retrain the model and serve the model as an API endpoint.

Shaun McGee

Product & delivery

Equal Experts, USA

Create a walking skeleton / steel thread

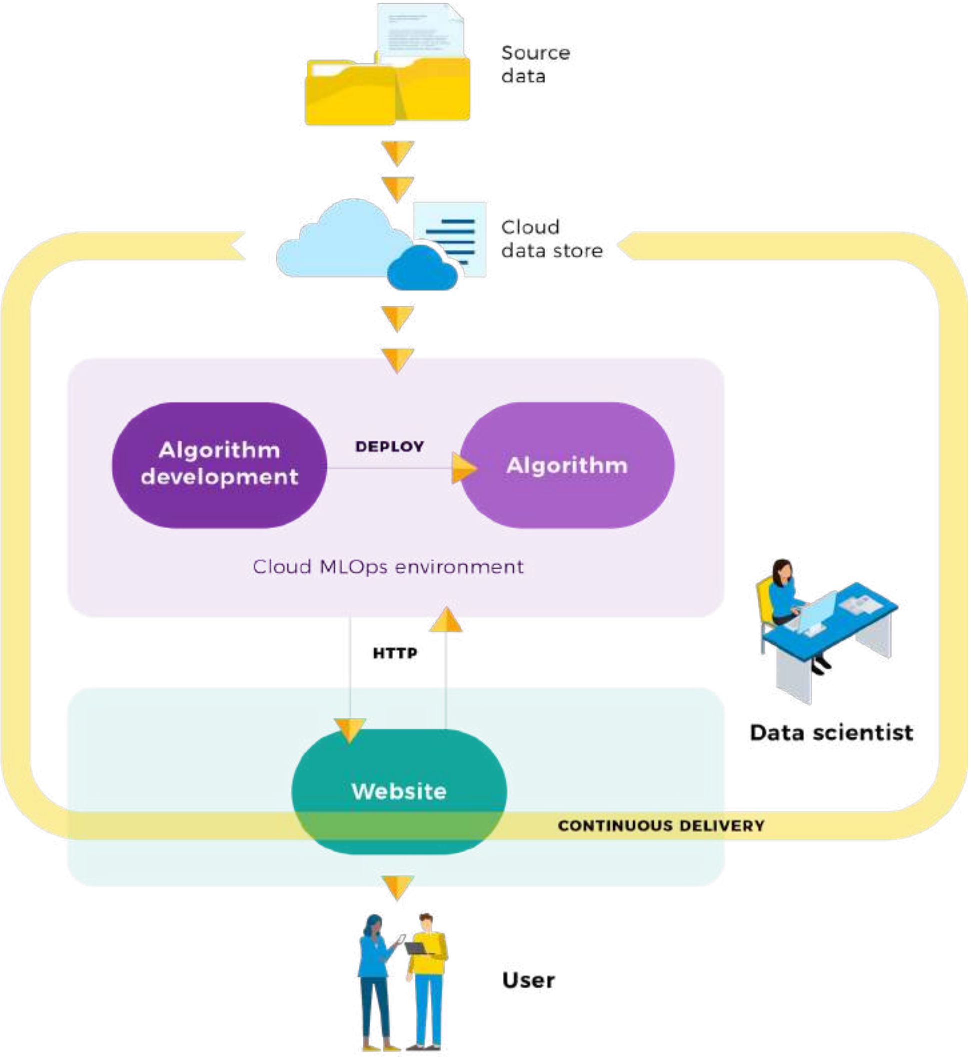
One of the challenges of operationalising a model is integration. Having a model ready to be called as an API is one task, having the external systems calling it is a completely separate and often complex task.

Usually, the team that is operationalising the model doesn't do the changes on the external systems, it's the teams responsible for those systems. These inter-team dependencies are always hard to manage and can directly affect the delivery of a machine learning project.

In addition, the complexity of integration depends directly on the systems that will integrate with the model. Imagine that the goal of a model is to be used by a big, old monolithic system that is very hard to change or to add features, with a cumbersome deploy process. Without testing, the integration will be time consuming and it will require effort from other teams that need to prioritise in their backlogs.

Based on the previous description, the best practice to minimise the impact of these external dependencies on the ML project, is to deploy a skeleton of a model. A skeleton model can be a dumb model that returns always one constant prediction, and with this dumb model the external teams can start to ingegrate since the start of the project.

One key aspect of the integration is that it should have a feature flag that indicates when the model should be used, so the skeleton model can be integrated, also being called but without affecting the behaviour of the external systems.



Experience report

In one of our ML projects, we created a model to predict the behaviour of a user on a webapp. The webapp was composed of a monolith, which was deployed once a week. The webapp team always had a large backlog of features so prioritising external integrations was always a slow process.

> To make integration easier we created a mock API at the start of the project. This decoupled the integration from the model development, meaning that a lot of the work could happen in parallel, significantly reducing the time to the first working version.

Miguel Duarte

Data engineer

Equal Experts, EU

Appropriately optimise models for inference

The computational profile of models can be very different during a model's training phase (i.e. when it's in development) and when a model is used for inference (i.e. deployed and making predictions in production). How you optimise your model can have quite dramatic cost implications when running it.

> During **training**, we show the in-development model a huge amount of training examples, and then use optimisation techniques like gradient descent and automatic differentiation to adjust the models' internal weights by a small amount, and then repeat this process many times. This involves both a lot of data movement, and keeping track of internal optimisation variables that are only relevant during the training phase. For very large models, we can parallelise training using techniques such as data and model parallelism and split the computations over multiple devices (e.g. GPUs). It may make sense to use specialised hardware such as GPUs and TPUs.

> During **inference** we show the model a single input (or a small batch of inputs) and ask it to make a prediction against just that input, once. In this phase we need to optimise a model to minimise latency (i.e. take as little time as possible to produce an answer), and must choose strategies like whether to batch up requests to a model or make them one at a time.

For small ML models, or models which don't receive a large amount of traffic, optimising the inference phase of the model may not be worth the cost - avoid premature optimisation!

Several tools and techniques exist that help us produce leaner models:

> **Pruning** considers whether some model complexity can be shed without much impact in performance. Many popular ML frameworks such as TensorFlow and PyTorch have tools built in to help with this process.

> **Quantisation of model weights** means finding ways to represent the internals of a model so that they use less memory and allow operations to be processed in parallel by modern hardware, but give nearly the same performance as the unoptimised model. Quantisation is also supported by some modern ML frameworks out of the box.

> **Deep learning compilers** such as TVVM can apply optimisations to the computations that a model performs.

> Use **vendor-specific tools** such as NVIDIA's TensorRT, Intel's OpenVino or Graphcore's Poplar tools when running on specific hardware

> At the extreme end of performance optimisation, heavy users can consider **specialised hardware** offered by the likes of Google, NVIDIA, Cerebras, Graphcore, SambaNova and others. This is an exciting and rapidly growing market! Many of these offerings are available in large cloud providers (e.g. NVIDIA GPUs, Google TPUs, Graphcore IPUs).

> Being able to **scale deployments** of your model dynamically to meet the demands of traffic to your service, and free up resources when demand is low. For models hosted as endpoints in a cloud environment a simple solution can be placing a load balancer in front of the model endpoint.

Our recommendation is to track the training and in-production running costs of a model carefully, and to regularly review whether the effort of better optimisation makes sense for you. To avoid ending up in a situation where you can't deploy new versions of your model rapidly, any optimisation you invest in needs to be repeatable and automatic (a core part of the model development process).

Experience report

I was asked to help a team improve the ability of their algorithms to scale. The purpose of the algorithm was to create an index that could find the same or very similar items from a text description and an image. It was a very cool algorithm, which used some of the latest deep-learning techniques but was just taking too long to add new items to the index.

I took an end to end look at the processing so I could understand the latencies, and found several points that could be improved. Some of the optimisations were small things but some of the more important ones were:

- The models had to be run on GPUs, which were often shared with other jobs, so I implemented a GPU acquisition algorithm to lock and release the resources the algorithm needed.
- The algorithm accessed lots of data from GCP BigQuery - introducing partitioning made it much quicker to get to the data it needed.
- Introducing a two phase approach of an initial quick filter, followed by applying the complex algorithm only where matches might occur reduced matching times.
- The initial code featured a race condition which sometimes occurred. Four lines of code were enough to implement a simple locking condition to stop this happening.

Putting all these changes together resulted in the code executing at less than 10% of the time than previously, which meant that the new data could be processed in the right time frames and the backlog of items to be indexed could be removed as well.

Emrah Gozcu

ML / Data engineer

Equal Experts, UK

Explore

Our playbooks are collections of observations that we have made many times in different sectors and clients. However, there are some emerging technologies and approaches which we have only applied in one or two places to date, but which we think are really promising. **We think they will become recommended practices in the future** – or are at least worth experimenting with. For now we are recommending you explore them at least.

Feature stores

Data is central to any ML system - it's needed both online and offline, for exploration and realtime prediction. One of the challenges in operationalising any ML algorithm is ensuring that any data used to train the model is also available in production. It is not simply the raw data that is used by the model - in most cases the raw data needs to be transformed in some way to create a data feature. (See Provide an Environment which Allows Data Scientists to create and test models for a description of Features and Feature Engineering.)

Creating a feature can be a time-consuming activity and you need it to be available for both offline and online activities. Furthermore, a feature you have created for one purpose may well be relevant for another task. A feature store is a component that manages the ingestion of raw data (from databases, event streams etc.) and turns it into features which can be used both to train models and as an input to the operational model. It takes the place of the data warehouse and the operational data pipelines - providing a batch API or query mechanism for retrieval of feature data-sets for model training, as well as a low latency API to provide data for real-time predictions.

The benefits are that:

- You do not need to create a separate data pipeline for the online inference
- Exactly the same transforms are used for training as for online inference

Experience report

In my experience MLOps is not just about tooling. It's a culture - a mindset for bringing data scientists, engineers and content experts together - but let's just focus on some tooling for now!

One of the marks of successfully getting machine learning into operations is that tedious and difficult tasks are automated, and it is easy for developers and data scientists to work together. I've recently been using Google Vertex AI as the framework for managing machine learning models at an online retailer. Prior to using Vertex AI, there were several teams doing ML operations in different ways. Some were using Airflow and Kubernetes, others were using hand-rolled in-house builds and data stores.

We have used Vertex AI to create a shared toolset for managing the model lifecycle, with standardised components to do the typical things you need to do:

- Workflow management/ orchestration
- Model serving
- Model repository
- Feature store

I have found the feature store to be really useful. Our models need to use aggregated features like average lead times for products, and the Vertex AI feature store is a good place to calculate and store them. Using the feature store means that I know that the data is processed the same way for training as when applied to the model in production. It saves us time because we don't have to create separate data pipelines for the deployed model in operation. It also has other advantages - keeping data in the feature store makes it easier to query how these aggregated features have changed over time. I think they will become a standard part of most ML environments.

Bas Geerdink

ML specialist

Equal Experts, EU

Pitfalls

User Trust and Engagement

A common pitfall when surfacing a machine learning score or algorithmic insight is that end users don't understand or trust the new data points. This can lead to them ignoring the insight, no matter how accurate or useful it is.

This usually happens when ML is conducted primarily by data scientists in isolation from users and stakeholders, and can be avoided by:

> Engaging with users from the start - understand what problem they expect the model to solve for them and use that to frame initial investigation and analysis

> Demo and explain your model results to users as part of your iterative model development - take them on the journey with you.

> Focus on explainability - this may be of the model itself. our users may want feedback on how it's arrived at its decision (e.g. surfacing the values of the most important features used to provide a recommendation), or it may be guiding your users on how to take action on the end result (e.g. talking through how to threshold against a credit risk score)

> Users will prefer concrete domain based values over abstract scores or data points, so feed this consideration into your algorithmic selection.

> Give access to model monitoring and metrics once you are in production - this will help maintain user trust if they wish to check in on model health if they have any concerns.

> Provide a feedback mechanism - ideally available directly alongside the model result. This allows the user to confirm good results and raise suspicious ones, and can be a great source of labelling data. Knowing their actions can have a direct impact on the model provides trust and empowerment.

Experience report

We had a project tasked with using machine learning to find fraudulent repayment claims, which were being investigated manually inside an application used by case workers. The data science team initially understood the problem to be one of helping the case workers know which claims were fraud, and in isolation developed a model that surfaced an score of 0 - 100 overall likelihood of fraud.

The users didn't engage with this score as they weren't clear about how it was being derived, and they still had to carry out the investigation to confirm the fraud. It was seldom used.

A second iteration was developed that provided a score on the bank account involved in the repayment instead of an overall indicator. This had much higher user engagement because it indicated a jumping off point for investigation and action to be taken.

Users were engaged throughout development of the second iteration, and encouraged to bring it into their own analytical dashboards instead of having it forced into the case working application. Additionally, whenever a bank account score was surfaced, it was accompanied by the values of all features used to derive it. The users found this data just as useful as the score itself for their investigations.

Shital Desai
Product owner

Equal Experts, UK

Explainability

ML models come in many flavours. Some models are naturally easy to explain. Rules-based models or simple statistical ones can be easily inspected and intuitively understood. The typical machine learning approaches are usually harder to understand. At the extreme, deep learning models are very complex and need specialised approaches to understand their inner workings.

It is important to know if explainability to end users is a requirement up front, because this will influence the model you decide to go with. In some use cases, there is a regulatory need and explainability is an essential requirement e.g. for credit risking it is essential to be able to explain why an applicant has been denied a loan. In other cases the model will simply not be accepted by end users if they cannot understand how a decision has been reached.

Explainability goes hand in hand with simplicity, and a simple model may well perform worse than a complex one. It's is common to find that an explainable model performs less well in terms of accuracy. This is fine! Accuracy alone is not always the only measure of a good model.

In our experience, engaging the end user and explaining how the model is making decisions often leads to a better model overall. The conversations you have with end users who understand their domain and data often result in the identification of additional features that, when added, improve model performance. In any event, explainability is often a useful part of developing the model and can help to identify model bias, reveal unbalanced data, and to ensure the model is working in the intended way.

If you find you have a complex model and need an explanatory solution, these tools can help:

- Yellowbrick
- SHAP - good for use during model development
- The What If Tool - good for counterfactual analysis
- Google's AI Explanations - good for using on deployed models (tensorflow models, tabular, text, image data, AutoML)
- Their related white paper

Experience report

In one of the engagements that I was involved in, we had to take an ML initiative from an existing partner. During discovery we found that the stakeholders in the business units did not understand the machine model that had been built and their questions related to the output (predictions) were answered with deep technical jargon that they could not comprehend. This resulted in the business units at best using the output grudgingly without any trust or at worst completely ignoring the output.

One of the first things we did when we took over the operations of the system was to translate the model outputs into outcomes and visuals that explained what the model was predicting in business terms. This was done during the initial iterations of building the model.

Three significant changes happened in how the data team was able to collaborate with the business units:

1 *The stakeholders understood what the model was trying to do. They were able to superimpose the output of the models on their own deep understanding of the business. They either concurred with the model outputs or challenged them. The questions that they raised helped the data team to look for errors in their data sources/ assumptions or explore additional data/features, thereby improving the output.*

2 *The business units also understood better the need for high quality data that affect the model outputs. They took steps to fix processes that either were incomplete in data collection or were ambitious resulting in confused data collection.*

3 *As the stakeholders were involved very early in the model building process, they considered themselves to be co-creators of the model rather than just consumers. This resulted in enthusiastic adoption of outputs including acceleration of any process changes needed to leverage the work.*

We were able to deploy five models in production over a period of six months that were being used to generate business outcomes compared to one model that went live in the earlier attempt, after 18 months.

Oshan Modi
Data scientist

Equal Experts, India

Avoid notebooks in production

Like many people we both love and hate notebooks such as Jupyter. Data science and the initial stages of model/algorithm development are creative processes, requiring lots of visualisations and quick pivoting between modelling approaches. For this rapid analysis of data and prototyping of algorithms, notebooks are excellent tools and they are the tool of choice for many data scientists. However they have a number of features which make them difficult to use in production.

- Notebook files contain both code and outputs - these can be large (e.g. images) and also contain important business or even personal data. When used in conjunction with version control such as Git, data is by default committed to the repo. You can work round this but it is all too easy to inadvertently pass data to where it shouldn't be. It also means that it is difficult/impossible to see exactly what changes have been made to the code from one commit to the next.
- Notebook cells can run out of order - meaning that different results are possible from the same notebook - depending on what order you run the cells in.
- Variables can stay in the kernel after the code which created them has been deleted.
- Variables can be shared between notebooks using magic commands.
- Not all python features work in a notebook e.g. multi-processing will not function in Jupyter
- The format of notebooks does not lend itself easily to testing - there are no intuitive test frameworks for notebooks.

In some cases we have used tools like papermill to run notebooks in production, but most of the time moving to standard modular code after an initial prototype has been created will make it more testable, easier to move into production and will probably speed up your algorithm development as well.

Experience report

I first came into contact with a Jupyter notebook while working on a predictive maintenance machine learning project, after a number of years as a production software developer. In this scenario, I found notebooks to be an invaluable resource. The ability to organise your code into segments with full markdown support and charts showing your thinking and output at each stage made demos and technical discussions simple and interactive. In addition, the tight integration with Amazon SageMaker and S3 meant I could work with relative freedom and with computing power on-tap while remaining in the client's estate.

However, as our proof of concept got more complicated, with a multi-stage ELT pipeline and varying data normalisation techniques etc, I found myself maintaining a block of core ELT code that was approaching 500 lines of untested spaghetti. I had tried, with some success, to functionalise it so it wasn't just one script and I could employ some DRY principles. However, I couldn't easily call the functions from one notebook to another so I resorted to copy and paste. Often I would make a small change somewhere and introduce a regression that made my algorithm performance drop off a cliff, resulting in losing half a day trying to figure out where I had gone wrong. Or maybe I'd restart my code in a morning and it wouldn't work because it relied on some globally scoped variable that I'd created and lost with my kernel the night before. If there were tests, I could have spotted these regressions and fixed them quickly, which would have saved me far more time in lost productivity than the tests would have taken to write in the first place.

> In retrospect, when I come to do work like this in the future, I would opt for a hybrid approach. I would write the initial code for each stage in a notebook where I could make changes in an interactive way and design an initial process that I was happy with. Then, as my code 'solidified', I would create an installable package in a separate GIT repository where I could make use of more traditional software development practices.

Using this approach has a number of advantages:

- You can import your code into any notebook by a simple pip install. You can use the same tested and repeatable ELT pipeline in a number of notebooks with differing algorithms with confidence.
- You can write and run tests and make use of CI tools, linting and all the other goodies software developers have created to make our code more manageable.
- Reduce your notebook's size, so that when you're doing presentations and demos you don't need 1,000 lines of boilerplate before you get to the good stuff.

The final advantage of this approach, in a world of deadlines where proof of concepts far too often become production solutions, is that you productionise your code as you go. This means that when the time comes that your code needs to be used in production, standardising it doesn't seem like such an insurmountable task.

Jake Saunders

Python developer

Equal Experts, UK

Poor security practices

Operationalising ML uses a mixture of infrastructure, code and data, all of which should be implemented and operated in a secure way. Our Secure Development playbook describes the practices we know are important for secure development and operations and these should be applied to your ML development and operations.

Some specific security pitfalls to watch out for in ML based solutions are:

> **Making the model accessible to the whole internet** - making your model endpoint publicly accessible may expose unintended inferences or prediction metadata that you would rather keep private. Even if your predictions are safe for public exposure, making your endpoint anonymously accessible may present cost management issues. A machine learning model endpoint can be secured using the same mechanisms as any other online service.

> **Exposure of data in the pipeline** - you will certainly need to include data pipelines as part of your solution. In some cases they may use personal data in the training. Of course these should be protected to the same standards as you would in any other development.

> **Embedding API Keys in mobile apps** - a mobile application may need specific credentials to directly access your model endpoint. Embedding these credentials in your app allows them to be extracted by third parties and used for other purposes. Securing your model endpoint behind your app backend can prevent uncontrolled access.

Don’t treat accuracy as the only or even the best way to evaluate your algorithm

We have spoken a lot about performance in this playbook but have deliberately shied away from specifying how it is calculated. How well your algorithm is working is context-dependent and understanding exactly the best way to evaluate it is part of the ML process. What we do know is that in most cases a simple accuracy measure - the percentage of correct classifications - is not the right one. You will obviously collect technical metrics such as precision (how many classifications are true) and recall (how many of the true examples did we identify) or more complex measures such as F scores, area under the curve etc. But these are usually not enough to gain user buy-in or define a successful algorithm on their own (see Business Impact is more than just Accuracy - Understand your baseline for an in-depth discussion of this.)

Use machine learning judiciously

Despite the hype, machine learning should not be the default approach to solve a problem. Complex problems, tightly tied to how our brains work like machine vision and natural language processing, are generally accepted as best tackled with artificial intelligence based on machine learning. Many real-world problems affecting a modern organisation are not of this nature and applying machine learning where it is not needed brings ongoing complexity, unpredictability and dependence on skills that are expensive to acquire and maintain. You could build a machine learning model to predict whether a number is even or odd - but you shouldn’t.

We typically recommend trying a non-machine learning based solution first. Perhaps a simple, rules-based system might work well enough to be sufficient. If nothing else, attempting to solve the problem with a non machine-learning approach will give you a baseline of complexity and performance that a machine learning-based alternative can be compared with.

Don’t forget to understand the at-inference usage profile

If you are deploying your algorithm as a microservice endpoint it’s worth thinking about how often and when it will be called. For typical software applications you may well expect a steady request rate. Whereas, for many machine learning applications it can be called as part of a large batch process leading to bursty volumes where there are no requests for five days then a need to handle 5 million inferences at once. A nice thing about using a walking skeleton (Create a Walking Skeleton / Steel Thread) is that you get an early understanding of the demand profile and can set up load balancing for appropriate provisioning.

Don’t make it difficult for data scientists to access data or use the tools they need

The data scientists who create an algorithm must have access to the data they need, in an environment that makes it easy for them to work on the models. We have seen situations where they can only work on data in an approved environment which does not have access to the data they need and they have no means of adding data they want to create their algorithms. Obviously they will not be able to work with these tools and will likely seek opportunities elsewhere to apply their skills.

Similarly, data science is a fast moving domain and great algorithms are open-sourced all the time - often in the form of Git repositories that can be put to use immediately to meet business needs. In a poorly designed analysis environment it is not possible to use these libraries, or they must go through an approval process which takes a long time.

In many cases these problems are a result of over-stringent security controls - whilst everyone needs to ensure that data is adequately protected, it is important that data architects do not become overzealous, and are able to pragmatically and rapidly find solutions that allow the data scientists to do their work efficiently.

In some situations, IT functions have taken a simplistic view that analytical model development is identical to code development, and therefore should be managed through the same processes as IT releases using mocked/obfuscated or small volume data in no-production environments. This shows a lack of understanding of how the shape and nuance of real data can impact on the quality of the model.

Not taking into consideration the downstream application of the model

> You will only get value from your investment in machine learning when it has been integrated into your business systems. Whilst it can be technically straightforward to provide an output, integrating into the business processes can take some time due to usability needs and security constraints.

Technical incompatibility or unrealistic accuracy expectations, if not addressed at the beginning of the project, can lead to delays, disappointment and other negative outcomes. For example, it is common to apply ML to tasks like 'propensity to buy' - finding people who may be interested in purchasing your product. If you did not take this downstream application into account from early on in the development, you might well provide the output in a form which is not usable such as an API endpoint, when a simple file containing a list or table supplied to an outbound call centre is all that is needed. Taking our recommendation to Create a Walking Skeleton/ Steel Thread is a great way to avoid this.

Contributors

We'd like to thank everyone within Equal Experts who has shared their wisdom and experiences with us, and have made this playbook possible.

Main Authors

Paul Brabban
Jon Carney
Simon Case
Scott Cutts
Claudio Diniz
Bas Geerdink
Thorben Louw
Jennifer Stark
Rajesh Thiagarajan

Thanks also to

Khalil Chourou
Adam Fletcher
Matteo Guzzo
Uttam Kini
Oshan Modi
Shaun McGee
Austin Poulton
Katharina Rasch
Jake Saunders
Isabell Britsch

Building a culture of innovation?

You need Experts by your side

A global network of technology shapers, we'll keep you innovating.

Engaging 3,000 consultants across 5 continents, we help create leading digital products and services. We decide with data, design for users, deliver at pace and scale sustainably.

Bringing undiluted expertise

We only engage senior consultants. In years of experience, our people have twice the industry average. Low-ego and curious, they share knowledge with your teams and across our network.

Keeping you innovating

We take the time to understand where you are now. We get excited about where you want to be. Building momentum with many small changes, we embed innovation into your every day.

Easy to work with

Valuing context, we never offer cookie-cutter solutions. Treating each other and our customers as equals, we solve problems with you. Infusing our culture and skills into your organisation, we build long-lasting value.

Experienced consultants know what they're doing – we free them to do what they do best. Find out what it's like to work with us get in touch.