<https://medium.com/@mumbaiyachori/ad-click-prediction-machine-learning-system-design-6e553d7ccc1c>

# Ad Click Prediction | Machine learning system design

<https://github.com/prakhargurawa/machine-learning-systems-design>

1. Project setup
2. Data pipeline
3. Modeling: selecting, training, and debugging
4. Serving: testing, deploying, and maintaining

A technique often used by the winners of machine learning competitions, including the famed $1M Netflix Prize and many Kaggle competitions, is [ensembling](https://en.wikipedia.org/wiki/Ensemble_learning): combining "multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone."

Project setup 🡪 Ingestion 🡪 data storing (optional)🡪 data exploration 🡪 prep and train 🡪 model and serve 🡪 monitor and continual learning 🡪 business anaylasis

Designing a machine learning system is an iterative process. There are generally four main components of the process: project setup, data pipeline, modeling (selecting, training, and debugging your model), and serving (testing, deploying, maintaining).

**Text, letter

Description automatically generated**

**Project Setup** [ Goals, User experience, Performance constraints, Evaluation, Personalization, Project constraints]

**Data pipeline** [Data availability, user data, storage, data preprocessing and representation,challenges,privacy, biases]

**Model Selection**[Problem definition, Problem type selection, Model selection, Training, Re-training, Hyperparam tune, Debugging, Scaling]

For example, for the task of classification, before using a transformer-based model with 300 million parameters, see if a decision tree works. For fraud detection, before wielding complex neural networks, try one of the many popular non-neural network approaches such as k-nearest neighbor classifier.

It's almost the norm now for machine learning engineers and researchers to train their models on multiple machines (CPUs, GPUs, TPUs). Modern machine learning frameworks make it easy to do distributed training. The most common parallelization method with multiple workers is data parallelism: you split your data on multiple machines, train your model on all of them, and accumulate gradients. This gives rise to a couple of issues.

With data parallelism, each worker has its own copy of the model and does all the computation necessary for the model. Model parallelism is when different components of your model can be evaluated on different machines. For example, machine 0 handles the computation for the first two layers while machine 1 handles the next two layers, or some machines can handle the forward pass while several others handle the backward pass. In theory, nothing stops you from using both data parallelism and model parallelism. However, in practice, it can pose a massive engineering challenge.

Serving

Training and serving aren't two isolated processes.  You need to consider the performance/interpretability tradeoffs. Making a model more complex might increase its performance but make the results harder to interpret.

<https://medium.com/airbnb-engineering/using-machine-learning-to-predict-value-of-homes-on-airbnb-9272d3d4739d>

Additionally, ML infra created a new framework that will automatically translate Jupyter notebooks into Airflow pipelines.

Video Recommendation

<https://www.educative.io/courses/machine-learning-system-design/g7p515EBD5r>

Diagram

Description automatically generated

* Each user has a list of video watches (videos, minutes\_watched).

Diagram

Description automatically generated

Diagram

Description automatically generated

**System Design Machine Learning by by Chip Huyen Notes**

Chapters [4](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch04.html#training_data) to [6](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch06.html#model_development_and_offline_evaluatio) cover the pre-deployment phase of an ML project: from creating the training data and engineering features to developing and evaluating your models in a development environment. This is the phase where expertise in both ML and the problem domain are especially needed.

Chapters 7 to 9 cover the deployment and post-deployment phase of an ML project. We’ll learn through a story many readers might be able to relate to that having a model deployed isn’t the end of the deployment process. The deployed model will need to be monitored and continually updated to changing environments and business requirements.

Chapters [3](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch03.html#data_engineering_fundamentals) and [10](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch10.html#infrastructure_and_tooling_for_mlops) focus on the infrastructure needed to enable stakeholders from different backgrounds to work together to deliver successful ML systems. [Chapter 3](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch03.html#data_engineering_fundamentals) focuses on data systems, whereas [Chapter 10](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch10.html#infrastructure_and_tooling_for_mlops) focuses on compute infrastructure and ML platforms.

Chapter 1. Overview of Machine Learning Systems

Graphical user interface

Description automatically generated

Graphical user interface, table

Description automatically generated with medium confidenceIn production, data, if available, is a lot more messy. It’s noisy, possibly unstructured, constantly shifting. It’s likely biased, and you likely don’t know how it’s biased. Labels, if there are any, might be sparse, imbalanced, or incorrect. Changing project or business requirements might require updating some or all of your existing labels. If you work with users’ data, you’ll also have to worry about privacy and regulatory concerns.

You or someone in your life might already be a victim of biased mathematical algorithms without knowing it. Your loan application might be rejected because the ML algorithm picks on your zip code, which embodies biases about one’s socioeconomic background. Your resume might be ranked lower because the ranking system employers use picks on the spelling of your name.

Fairness, Interpretability, Data, Computation priorities

# Chapter 2. Introduction to Machine Learning Systems Design

# Requirements for ML Systems

## Reliability Scalability Maintainability Adaptability

## Diagram Description automatically generated

*Step 1. Project scoping*

A project starts with scoping the project, laying out goals, objectives, and constraints. Stakeholders should be identified and involved. Resources should be estimated and allocated. We already discussed different stakeholders and some of the foci for ML projects in production in [Chapter 1](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch01.html#overview_of_machine_learning_systems). We also already discussed how to scope an ML project in the context of a business earlier in this chapter. We’ll discuss how to organize teams to ensure the success of an ML project in [Chapter 11](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch11.html#the_human_side_of_machine_learning).

*Step 2. Data engineering*

A vast majority of ML models today learn from data, so developing ML models starts with engineering data. In [Chapter 3](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch03.html#data_engineering_fundamentals), we’ll discuss the fundamentals of data engineering, which covers handling data from different sources and formats. With access to raw data, we’ll want to curate training data out of it by sampling and generating labels, which is discussed in [Chapter 4](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch04.html#training_data).

*Step 3. ML model development*

With the initial set of training data, we’ll need to extract features and develop initial models leveraging these features. This is the stage that requires the most ML knowledge and is most often covered in ML courses. In [Chapter 5](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch05.html#feature_engineering), we’ll discuss feature engineering. In [Chapter 6](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch06.html#model_development_and_offline_evaluatio), we’ll discuss model selection, training, and evaluation.

*Step 4. Deployment*

After a model is developed, it needs to be made accessible to users. Developing an ML system is like writing—you will never reach the point when your system is done. But you do reach the point when you have to put your system out there. We’ll discuss different ways to deploy an ML model in [Chapter 7](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch07.html#model_deployment_and_prediction_service).

*Step 5. Monitoring and continual learning*

Once in production, models need to be monitored for performance decay and maintained to be adaptive to changing environments and changing requirements. This step will be discussed in Chapters [8](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch08.html#data_distribution_shifts_and_monitoring) and [9](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch09.html#continual_learning_and_test_in_producti).

*Step 6. Business analysis*

Model performance needs to be evaluated against business goals and analyzed to generate business insights. These insights can then be used to eliminate unproductive projects or scope out new projects. This step is closely related to the first step.

### Multiclass versus multilabel classification

## 

Figure 2-5. Given the problem of predicting the app a user will most likely open next, you can frame it as a classification problem. The input is the user’s features and environment’s features. The output is a distribution over all apps on the phone.

This is a bad approach because whenever a new app is added, you might have to retrain your model from scratch, or at least retrain all the components of your model whose number of parameters depends on N. A better approach is to frame this as a regression task.

A screenshot of a computer

Description automatically generated with medium confidence

loss = ɑ quality\_loss + β engagement\_loss

*quality\_model*

Minimizes *quality\_loss* and outputs the predicted quality of each post

*engagement\_model*

Minimizes *engagement\_loss* and outputs the predicted number of clicks of each post

You can combine the models’ outputs and rank posts by their combined scores:

ɑ quality\_score + β engagement\_score

Now you can tweak α and β without retraining your models!

# Chapter 3. Data Engineering Fundamentals

These two different types of data require different processing paradigms, which we’ll discuss in the section [“Batch Processing Versus Stream Processing”](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch03.html#batch_processing_versus_stream_processi).

# Data Sources

One source is user input data, data explicitly input by users.

Another source is system-generated data.

There are also internal databases, generated by various services and enterprise applications in a company.

Then there’s the wonderfully weird world of third-party data.

Table

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

# Chapter 4. Training Data

Sampling is an integral part of the ML workflow that is, unfortunately, often overlooked in typical ML coursework. Sampling happens in many steps of an ML project lifecycle, such as sampling from all possible real-world data to create training data; sampling from a given dataset to create splits for training, validation, and testing; or sampling from all possible events that happen within your ML system for monitoring purposes. In this section, we’ll focus on sampling methods for creating training data, but these sampling methods can also be used for other steps in an ML project lifecycle.

There are two families of sampling: nonprobability sampling and random sampling.

## Nonprobability Sampling

Nonprobability sampling is when the selection of data isn’t based on any probability criteria. Here are some of the criteria for nonprobability sampling:

*Convenience sampling*

Samples of data are selected based on their availability. This sampling method is popular because, well, it’s convenient.

*Snowball sampling*

Future samples are selected based on existing samples. For example, to scrape legitimate Twitter accounts without having access to Twitter databases, you start with a small number of accounts, then you scrape all the accounts they follow, and so on.

*Judgment sampling*

Experts decide what samples to include.

*Quota sampling*

You select samples based on quotas for certain slices of data without any randomization. For example, when doing a survey, you might want 100 responses from each of the age groups: under 30 years old, between 30 and 60 years old, and above 60 years old, regardless of the actual age distribution.

Nonprobability sampling can be a quick and easy way to gather your initial data to get your project off the ground. However, for reliable models, you might want to use probability-based sampling, which we will cover next.

## Simple Random Sampling

## Stratified Sampling

## Weighted Sampling

## Reservoir Sampling

# Labeling

## Handling the Lack of Labels

Table

Description automatically generated

Table

Description automatically generated

Active learning is a method for improving the efficiency of data labels. The hope here is that ML models can achieve greater accuracy with fewer training labels if they can choose which data samples to learn from. Active learning is sometimes called query learning—though this term is getting increasingly unpopular—because a model (active learner) sends back queries in the form of unlabeled samples to be labeled by annotators (usually humans).

It’s good practice to keep track of the origin of each of your data samples as well as its labels, a technique known as data lineage. Data lineage helps you both flag potential biases in your data and debug your models. For example, if your model fails mostly on the recently acquired data samples, you might want to look into how the new data was acquired. On more than one occasion, we’ve discovered that the problem wasn’t with our model, but because of the unusually high number of wrong labels in the data that we’d acquired recently.

# Class Imbalance

# The second reason is that class imbalance makes it easier for your model to get stuck in a nonoptimal solution by exploiting a simple heuristic instead of learning anything useful about the underlying pattern of the data.

# The third reason is that class imbalance leads to asymmetric costs of error—the cost of a wrong prediction on a sample of the rare class might be much higher than a wrong prediction on a sample of the majority class.

### Data-level methods: Resampling

Data-level methods modify the distribution of the training data to reduce the level of imbalance to make it easier for the model to learn. A common family of techniques is resampling. Resampling includes oversampling, adding more instances from the minority classes, and undersampling, removing instances of the majority classes. The simplest way to undersample is to randomly remove instances from the majority class, whereas the simplest way to oversample is to randomly make copies of the minority class until you have a ratio that you’re happy with. [Figure 4-10](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch04.html#illustrations_of_how_undersampling_and) shows a visualization of oversampling and undersampling.

# Diagram Description automatically generated with medium confidence

# A popular method of oversampling low-dimensional data is SMOTE (synthetic minority oversampling technique).[39](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch04.html#idm46868210052544) It synthesizes novel samples of the minority class through sampling convex combinations of existing data points within the minority class.[40](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch04.html#ch01fn116)

#### Cost-sensitive learning

Graphical user interface, text, application

Description automatically generated

#### Class-balanced loss

Graphical user interface, text, application

Description automatically generated

In practice, ensembles have shown to help with the class imbalance problem.[**47**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch04.html#ch01fn122) However, we don’t include ensembling in this section because class imbalance isn’t usually why ensembles are used.

# Data Augmentation

Data augmentation is a family of techniques that are used to increase the amount of training data. Traditionally, these techniques are used for tasks that have limited training data, such as in medical imaging. However, in the last few years, they have shown to be useful even when we have a lot of data—augmented data can make our models more robust to noise and even adversarial attacks.

Graphical user interface, text, application

Description automatically generated

## Perturbation

Perturbation is also a label-preserving operation, but because sometimes it’s used to trick models into making wrong predictions, I thought it deserves its own section.

Adding noisy samples to training data can help models recognize the weak spots in their learned decision boundary and improve their performance.[**51**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch04.html#ch01fn125) Noisy samples can be created by either adding random noise or by a search strategy. Moosavi-Dezfooli et al. proposed an algorithm, called DeepFool, that finds the minimum possible noise injection needed to cause a misclassification with high confidence.[**52**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch04.html#ch01fn126) This type of augmentation is called adversarial augmentation.[**53**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch04.html#ch01fn127)

## Data Synthesis

Graphical user interface, text, application

Description automatically generated

ML algorithms work well in situations when the data distribution is more balanced, and not so well when the classes are heavily imbalanced. Unfortunately, problems with class imbalance are the norm in the real world. In the following section, we discussed why class imbalance made it hard for ML algorithms to learn. We also discussed different techniques to handle class imbalance, from choosing the right metrics to resampling data to modifying the loss function to encourage the model to pay attention to certain samples.

# Chapter 5. Feature Engineering

# Learned Features Versus Engineered Features

Diagram

Description automatically generated

## Handling Missing Values

### Deletion

On top of that, removing rows of data can create biases in your model, especially if the missing values are at random (MAR).

### Imputation

One common practice is to fill in missing values with their defaults. For example, if the job is missing, you might fill it with an empty string “”. Another common practice is to fill in missing values with the mean, median, or mode (the most common value).

## Scaling

Chart, histogram

Description automatically generated

## Discretization

Discretization is the process of turning a continuous feature into a discrete feature. This process is also known as quantization or binning. This is done by creating buckets for the given values. For annual income, you might want to group them into three buckets as follows:

* Lower income: less than $35,000/year
* Middle income: between $35,000 and $100,000/year
* Upper income: more than $100,000/year

## Encoding Categorical Features

In production, your model crashes because it encounters a brand it hasn’t seen before and therefore can’t encode. New brands join Amazon all the time. To address this, you create a category UNKNOWN with the value of 2,000,000 to catch all the brands your model hasn’t seen during training. Your model doesn’t crash anymore, but your sellers complain that their new brands are not getting any traffic. It’s because your model didn’t see the category UNKNOWN in the train set, so it just doesn’t recommend any product of the UNKNOWN brand. You fix this by encoding only the top 99% most popular brands and encode the bottom 1% brand as UNKNOWN. This way, at least your model knows how to deal with UNKNOWN brands.

## Feature Crossing

Feature crossing is the technique to combine two or more features to generate new features. This technique is useful to model the nonlinear relationships between features.

## Discrete and Continuous Positional Embeddings

First introduced to the deep learning community in the paper [“Attention Is All You Need”](https://oreil.ly/eXk16) (Vaswani et al. 2017), positional embedding has become a standard data engineering technique for many applications in both computer vision and NLP. We’ll walk through an example to show why positional embedding is necessary and how to do it.

### Scaling before splitting

As discussed in the section [“Scaling”](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch05.html#scaling), it’s important to scale your features. Scaling requires global statistics—e.g., mean, variance—of your data. One common mistake is to use the entire training data to generate global statistics before splitting it into different splits, leaking the mean and variance of the test samples into the training process, allowing a model to adjust its predictions for the test samples. This information isn’t available in production, so the model’s performance will likely degrade.

* The more features you have, the more opportunities there are for data leakage.
* Too many features can cause overfitting.
* Too many features can increase memory required to serve a model, which, in turn, might require you to use a more expensive machine/instance to serve your model.
* Too many features can increase inference latency when doing online prediction, especially if you need to extract these features from raw data for predictions online. We’ll go deeper into online prediction in [Chapter 7](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch07.html#model_deployment_and_prediction_service).
* Useless features become technical debts. Whenever your data pipeline changes, all the affected features need to be adjusted accordingly. For example, if one day your application decides to no longer take in information about users’ age, all features that use users’ age need to be updated.

## Feature Importance

Chart

Description automatically generated

# Chapter 6. Model Development and Offline Evaluation

Model development is an iterative process. After each iteration, you’ll want to compare your model’s performance against its performance in previous iterations and evaluate how suitable this iteration is for production.

## Evaluating ML Models

### Six tips for model selection

#### Avoid the state-of-the-art trap

#### Start with the simplest models

#### Avoid human biases in selecting models

#### Evaluate good performance now versus good performance later

Chart, line chart

Description automatically generated

#### Evaluate trade-offs

#### Understand your model’s assumptions

## Ensembles

Ensembling methods are less favored in production because ensembles are more complex to deploy and harder to maintain. However, they are still common for tasks where a small performance boost can lead to a huge financial gain, such as predicting click-through rate for ads.

### Bagging

Bagging, shortened from bootstrap aggregating, is designed to improve both the training stability and accuracy of ML algorithms.[**4**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch06.html#ch01fn153) It reduces variance and helps to avoid overfitting. Given a dataset, instead of training one classifier on the entire dataset, you sample with replacement to create different datasets, called bootstraps, and train a classification or regression model on each of these bootstraps. Diagram

Description automatically generated

### Boosting

Boosting is a family of iterative ensemble algorithms that convert weak learners to strong ones.

Diagram

Description automatically generated

### Stacking

Stacking means that you train base learners from the training data then create a meta-learner that combines the outputs of the base learners to output final predictions, as shown in [Figure 6-5](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch06.html#a_visualization_of_a_stacked_ensemble_f).

Diagram

Description automatically generated

## Experiment Tracking and Versioning

### Experiment tracking

A large part of training an ML model is babysitting the learning processes. Many problems can arise during the training process, including loss not decreasing, overfitting, underfitting, fluctuating weight values, dead neurons, and running out of memory. It’s important to track what’s going on during training not only to detect and address these issues but also to evaluate whether your model is learning anything useful.

* The *loss curve* corresponding to the train split and each of the eval splits.
* The *model performance metrics* that you care about on all nontest splits, such as accuracy, F1, perplexity.
* The log of *corresponding sample, prediction, and ground truth label*. This comes in handy for ad hoc analytics and sanity check.
* The *speed* of your model, evaluated by the number of steps per second or, if your data is text, the number of tokens processed per second.
* *System performance metrics* such as memory usage and CPU/GPU utilization. They’re important to identify bottlenecks and avoid wasting system resources.
* The values over time of any *parameter and hyperparameter* whose changes can affect your model’s performance, such as the learning rate if you use a learning rate schedule; gradient norms (both globally and per layer), especially if you’re clipping your gradient norms; and weight norm, especially if you’re doing weight decay.

### Versioning

Imagine this scenario. You and your team spent the last few weeks tweaking your model, and one of the runs finally showed promising results. You wanted to use it for more extensive tests, so you tried to replicate it using the set of hyperparameters you’d noted down somewhere, only to find out that the results weren’t quite the same.

##### Code versioning tools allow users to revert to a previous version of the codebase by keeping copies of all the old files. However, a dataset used might be so large that duplicating it multiple times might be unfeasible.

##### DEBUGGING ML MODELS

Here are some of the things that might cause an ML model to fail:

*Theoretical constraints*

As discussed previously, each model comes with its own assumptions about the data and the features it uses. A model might fail because the data it learns from doesn’t conform to its assumptions. For example, you use a linear model for the data whose decision boundaries aren’t linear.

*Poor implementation of model*

The model might be a good fit for the data, but the bugs are in the implementation of the model. For example, if you use PyTorch, you might have forgotten to stop gradient updates during evaluation when you should. The more components a model has, the more things that can go wrong, and the harder it is to figure out which goes wrong. However, with models being increasingly commoditized and more and more companies using off-the-shelf models, this is becoming less of a problem.

*Poor choice of hyperparameters*

With the same model, one set of hyperparameters can give you the state-of-the-art result but another set of hyperparameters might cause the model to never converge. The model is a great fit for your data, and its implementation is correct, but a poor set of hyperparameters might render your model useless.

*Data problems*

There are many things that could go wrong in data collection and preprocessing that might cause your models to perform poorly, such as data samples and labels being incorrectly paired, noisy labels, features normalized using outdated statistics, and more.

*Poor choice of features*

There might be many possible features for your models to learn from. Too many features might cause your models to overfit to the training data or cause data leakage. Too few features might lack predictive power to allow your models to make good predictions.

*Start simple and gradually add more components*

Start with the simplest model and then slowly add more components to see if it helps or hurts the performance. For example, if you want to build a recurrent neural network (RNN), start with just one level of RNN cell before stacking multiple together or adding more regularization. If you want to use a BERT-like model (Devlin et al. 2018), which uses both a masked language model (MLM) and next sentence prediction (NSP) loss, you might want to use only the MLM loss before adding NSP loss.

Currently, many people start out by cloning an open source implementation of a state-of-the-art model and plugging in their own data. On the off-chance that it works, it’s great. But if it doesn’t, it’s very hard to debug the system because the problem could have been caused by any of the many components in the model.

*Overfit a single batch*

After you have a simple implementation of your model, try to overfit a small amount of training data and run evaluation on the same data to make sure that it gets to the smallest possible loss. If it’s for image recognition, overfit on 10 images and see if you can get the accuracy to be 100%, or if it’s for machine translation, overfit on 100 sentence pairs and see if you can get to a BLEU score of near 100. If it can’t overfit a small amount of data, there might be something wrong with your implementation.

*Set a random seed*

There are so many factors that contribute to the randomness of your model: weight initialization, dropout, data shuffling, etc. Randomness makes it hard to compare results across different experiments—you have no idea if the change in performance is due to a change in the model or a different random seed. Setting a random seed ensures consistency between different runs. It also allows you to reproduce errors and other people to reproduce your results.

## Distributed Training

### Data parallelism

If your model updates the weight using the gradient from each machine separately—asynchronous SGD—gradient staleness might become a problem because the gradients from one machine have caused the weights to change before the gradients from another machine have come in

Diagram, schematic

Description automatically generated

### Model parallelism

Diagram, schematic

Description automatically generated

Timeline

Description automatically generated

# Model Offline Evaluation

## Baselines

*Random baseline Simple heuristic Zero rule baseline Human baseline Existing solutions*

## Evaluation Methods

### Perturbation tests

### Invariance tests

### Directional expectation tests

### Model calibration

# Chapter 7. Model Deployment and Prediction Service

A screenshot of a phone

Description automatically generated with medium confidence

# Example of how to use FastAPI to turn your predict function

# into a POST endpoint

@app.route('/predict', methods=['POST'])

def predict():

X = request.get\_json()['X']

y = MODEL.predict(X).tolist()

return json.dumps({'y': y}), 200

# Machine Learning Deployment Myths

## Myth 1: You Only Deploy One or Two ML Models at a Time

## A picture containing diagram Description automatically generated Myth 2: If We Don’t Do Anything, Model Performance Remains the Same

## Myth 3: You Won’t Need to Update Your Models as Much

## Myth 4: Most ML Engineers Don’t Need to Worry About Scale

# Batch Prediction Versus Online Prediction

* Batch prediction, which uses only batch features.
* Online prediction that uses only batch features (e.g., precomputed embeddings).
* Online prediction that uses both batch features and streaming features. This is also known as streaming prediction.

Diagram

Description automatically generated Diagram

Description automatically generated *Batch features*

The mean preparation time of this restaurant in the past

*Streaming features*

In the last 10 minutes, how many other orders they have, and how many delivery people are available

Diagram

Description automatically generated Table

Description automatically generated In many applications, online prediction and batch prediction are used side by side for different use cases. For example, food ordering apps like DoorDash and UberEats use batch prediction to generate restaurant recommendations—it’d take too long to generate these recommendations online because there are many restaurants. However, once you click on a restaurant, food item recommendations are generated using online prediction.

* A (near) real-time pipeline that can work with incoming data, extract streaming features (if needed), input them into a model, and return a prediction in near real time. A streaming pipeline with real-time transport and a stream computation engine can help with that.
* A model that can generate predictions at a speed acceptable to its end users. For most consumer apps, this means milliseconds.

Diagram

Description automatically generated Diagram

Description automatically generated

# Model Compression

The process of making a model smaller is called model compression, and the process to make it do inference faster is called inference optimization. Originally, model compression was to make models fit on edge devices. However, making models smaller often makes them run faster.

## Low-Rank Factorization

## Shape, arrow Description automatically generated with medium confidence Knowledge Distillation

Knowledge distillation is a method in which a small model (student) is trained to mimic a larger model or ensemble of models (teacher).

## Pruning

Pruning was a method originally used for decision trees where you remove sections of a tree that are uncritical and redundant for classification.[**25**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch07.html#ch01fn222) As neural networks gained wider adoption, people started to realize that neural networks are over-parameterized and began to find ways to reduce the workload caused by the extra parameters.

Pruning, in the context of neural networks, has two meanings. One is to remove entire nodes of a neural network, which means changing its architecture and reducing its number of parameters. The more common meaning is to find parameters least useful to predictions and set them to 0. In this case, pruning doesn’t reduce the total number of parameters, only the number of nonzero parameters. The architecture of the neural network remains the same. This helps with reducing the size of a model because pruning makes a neural network more sparse, and sparse architecture tends to require less storage space than dense structure. Experiments show that pruning techniques can reduce the nonzero parameter counts of trained networks by over 90%, decreasing storage requirements and improving computational performance of inference without compromising overall accuracy.[**26**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch07.html#ch01fn223) In [Chapter 11](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch11.html#the_human_side_of_machine_learning), we’ll discuss how pruning can introduce biases into your model.

## Quantization

Quantization is the most general and commonly used model compression method. It’s straightforward to do and generalizes over tasks and architectures.

Quantization reduces a model’s size by using fewer bits to represent its parameters. By default, most software packages use 32 bits to represent a float number (single precision floating point). If a model has 100M parameters and each requires 32 bits to store, it’ll take up 400 MB. If we use 16 bits to represent a number, we’ll reduce the memory footprint by half. Using 16 bits to represent a float is called half precision.

Instead of using floats, you can have a model entirely in integers; each integer takes only 8 bits to represent.

# ML on the Cloud and on the Edge

Putting your models on the edge is also appealing when handling sensitive user data. ML on the cloud means that your systems might have to send user data over networks, making it susceptible to being intercepted.

Edge computing makes it easier to comply with regulations, like GDPR, about how user data can be transferred or stored.

## Compiling and Optimizing Models for Edge Devices

### Model optimization

After you’ve “lowered” your code to run your models into the hardware of your choice, an issue you might run into is performance. The generated machine code might be able to run on a hardware backend, but it might not be able to do so efficiently. The generated code may not take advantage of data locality and hardware caches, or it may not leverage advanced features such as vector or parallel operations that could speed code up.

A study by researchers at Stanford DAWN lab found that typical ML workloads using NumPy, pandas, and TensorFlow run 23 times slower in one thread compared to hand-optimized code

There are standard local optimization techniques that are known to speed up your model, most of them making things run in parallel or reducing memory access on chips. Here are four of the common techniques:

*Vectorization*

Given a loop or a nested loop, instead of executing it one item at a time, execute multiple elements contiguous in memory at the same time to reduce latency caused by data I/O.

*Parallelization*

Given an input array (or n-dimensional array), divide it into different, independent work chunks, and do the operation on each chunk individually.

*Loop tiling*[***46***](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch07.html#idm46868207978704)

Change the data accessing order in a loop to leverage hardware’s memory layout and cache. This kind of optimization is hardware dependent. A good access pattern on CPUs is not a good access pattern on GPUs.

*Operator fusion*

Fuse multiple operators into one to avoid redundant memory access. For example, two operations on the same array require two loops over that array. In a fused case, it’s just one loop. [Figure 7-13](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch07.html#an_example_of_an_operator_fusiondot_sou) shows an example of operator fusion.

Diagram

Description automatically generated

## ML in Browsers

When talking about browsers, many people think of JavaScript. There are tools that can help you compile your models into JavaScript, such as [TensorFlow.js](https://oreil.ly/3Afzv), [Synaptic](https://oreil.ly/SYiLq), and [brain.js](https://oreil.ly/83IIa). However, JavaScript is slow, and its capacity as a programming language is limited for complex logics such as extracting features from data.

A more promising approach is WebAssembly (WASM). WASM is an open standard that allows you to run executable programs in browsers. After you’ve built your models in scikit-learn, PyTorch, TensorFlow, or whatever frameworks you’ve used, instead of compiling your models to run on specific hardware, you can compile your model to WASM. You get back an executable file that you can just use with JavaScript.

# Chapter 8. Data Distribution Shifts and Monitoring

# Causes of ML System Failures

## Software System Failures

Software system failures are failures that would have happened to non-ML systems. Here are some examples of software system failures:

*Dependency failure*

A software package or a codebase that your system depends on breaks, which leads your system to break. This failure mode is common when the dependency is maintained by a third party, and especially common if the third party that maintains the dependency no longer exists.[**2**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch08.html#ch01fn251)

*Deployment failure*

Failures caused by deployment errors, such as when you accidentally deploy the binaries of an older version of your model instead of the current version, or when your systems don’t have the right permissions to read or write certain files.

*Hardware failures*

When the hardware that you use to deploy your model, such as CPUs or GPUs, doesn’t behave the way it should. For example, the CPUs you use might overheat and break down.[**3**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch08.html#ch01fn252)

*Downtime or crashing*

If a component of your system runs from a server somewhere, such as AWS or a hosted service, and that server is down, your system will also be down.

## ML-Specific Failures

### Production data differing from training data

### Edge cases

# Diagram Description automatically generated with medium confidence

## Types of Data Distribution Shifts

* P(X, Y) = P(Y|X)P(X)
* P(X, Y) = P(X|Y)P(Y)

P(Y|X) denotes the conditional probability of an output given an input—for example, the probability of an email being spam given the content of the email. P(X) denotes the probability density of the input. P(Y) denotes the probability density of the output. Label shift, covariate shift, and concept drift are defined as follows:

*Covariate shift*

When P(X) changes but P(Y|X) remains the same. This refers to the first decomposition of the joint distribution.

*Label shift*

When P(Y) changes but P(X|Y) remains the same. This refers to the second decomposition of the joint distribution.

*Concept drift*

When P(Y|X) changes but P(X) remains the same. This refers to the first decomposition of the joint distribution.[**21**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch08.html#ch01fn270)

Table

Description automatically generated Timeline

Description automatically generated

# Monitoring and Observability

Monitoring is all about metrics. Because ML systems are software systems, the first class of metrics you’d need to monitor are the operational metrics. These metrics are designed to convey the health of your systems. They are generally divided into three levels: the network the system is run on, the machine the system is run on, and the application that the system runs. Examples of these metrics are latency; throughput; the number of prediction requests your model receives in the last minute, hour, day; the percentage of requests that return with a 2xx code; CPU/GPU utilization; memory utilization; etc. No matter how good your ML model is, if the system is down, you’re not going to benefit from it.

## ML-Specific Metrics

Diagram, text

Description automatically generated

# Chapter 9. Continual Learning and Test in Production

If the model is retrained to adapt to the changing environment, evaluating it on a stationary test set isn’t enough.

# Continual Learning

Companies that employ continual learning in production update their models in micro-batches. For example, they might update the existing model after every 512 or 1,024 examples—the optimal number of examples in each micro-batch is task dependent. A picture containing text, sign

Description automatically generated

## Stateless Retraining Versus Stateful Training

However, continual learning isn’t about the retraining frequency, but the manner in which the model is retrained. Most companies do stateless retraining—the model is trained from scratch each time. Continual learning means also allowing stateful training—the model continues training on new data.[**2**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch09.html#ch01fn304) Stateful training is also known as fine-tuning or incremental learning. The difference between stateless retraining and stateful training is visualized in Timeline

Description automatically generated

## Why Continual Learning?

The first use case of continual learning is to combat data distribution shifts, especially when the shifts happen suddenly. Imagine you’re building a model to determine the prices for a ride-sharing service like Lyft.[**6**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch09.html#ch01fn308) Historically, the ride demand on a Thursday evening in this particular neighborhood is slow, so the model predicts low ride prices, which makes it less appealing for drivers to get on the road.

## Continual Learning Challenges

### Fresh data access challenge

Text, chat or text message

Description automatically generated

Being able to pull fresh data isn’t enough. If your model needs labeled data to update, as most models today do, this data will need to be labeled as well. In many applications, the speed at which a model can be updated is bottlenecked by the speed at which data is labeled.

### Evaluation challenge

### Algorithm challenge

## Four Stages of Continual Learning

### Stage 1: Manual, stateless retraining

### Stage 2: Automated retraining

### Stage 3: Automated, stateful training

### Stage 4: Continual learning

# Test in Production

## Shadow Deployment

Shadow deployment might be the safest way to deploy your model or any software update. Shadow deployment works as follows:

1. Deploy the candidate model in parallel with the existing model.
2. For each incoming request, route it to both models to make predictions, but only serve the existing model’s prediction to the user.
3. Log the predictions from the new model for analysis purposes.

## A/B Testing

A/B testing is a way to compare two variants of an object, typically by testing responses to these two variants, and determining which of the two variants is more effective. We’ll use A/B testing to determine which model is better according to some predefined metrics. It is many ML engineers’ first response to how to evaluate ML models in production. A/B testing works as follows:

1. Deploy the candidate model alongside the existing model.
2. A percentage of traffic is routed to the new model for predictions; the rest is routed to the existing model for predictions. It’s common for both variants to serve prediction traffic at the same time. However, there are cases where one model’s predictions might affect another model’s predictions—e.g., in ride-sharing’s dynamic pricing, a model’s predicted prices might influence the number of available drivers and riders, which, in turn, influence the other model’s predictions. In those cases, you might have to run your variants alternatively, e.g., serve model A one day and then serve model B the next day.
3. Monitor and analyze the predictions and user feedback, if any, from both models to determine whether the difference in the two models’ performance is statistically significant.
4. To do A/B testing the right way requires doing many things right. In this book, we’ll discuss two important things. First, A/B testing consists of a randomized experiment: the traffic routed to each model has to be truly random. If not, the test result will be invalid. For example, if there’s a selection bias in the way traffic is routed to the two models, such as users who are exposed to model A are usually on their phones whereas users exposed to model B are usually on their desktops, then if model A has better accuracy than model B, we can’t tell whether it’s because A is better than B or whether “being on a phone” influences the prediction quality.
5. Second, your A/B test should be run on a sufficient number of samples to gain enough confidence about the outcome. How to calculate the number of samples needed for an A/B test is a simple question with a very complicated answer, and I’d recommend readers reference a book on A/B testing to learn more.

# Chapter 10. Infrastructure and Tooling for MLOps

Diagram

Description automatically generated

Graphical user interface, text, chat or text message

Description automatically generated

# Storage and Compute

Graphical user interface, application, Teams

Description automatically generated

# Development Environment

## Dev Environment Setup

## Standardizing Dev Environments

In the early days of our startup, we each worked from our own computer. We had a bash file that a new team member could run to create a new virtual environment—in our case, we use conda for virtual environments—and install the required packages needed to run our code. The list of the required packages was the good old requirements.txt that we kept adding to as we started using a new package. Sometimes, one of us got lazy and we just added a package name (e.g., torch) without specifying which version of the package it was (e.g., torch==1.10.0+cpu). Occasionally, a new pull request would run well on my computer but not another coworker’s computer,[**25**](https://learning.oreilly.com/library/view/designing-machine-learning/9781098107956/ch10.html#ch01fn362) and we usually quickly figured out that it was because we used different versions of the same package.

Then one day, my coworker got a new laptop. It was a MacBook with the then new M1 chip. He tried to follow our setup steps on this new laptop but ran into difficulty. It was because the M1 chip was new, and some of the tools we used, including Docker, weren’t working well with M1 chips yet. After seeing him struggling with setting the environment up for a day, we decided to move to a cloud dev environment. This means that we still standardize the virtual environment and tools and packages, but now everyone uses the virtual environment and tools and packages on the same type of machine too, provided by a cloud provider.

A much more popular option is to use a cloud dev environment with a local IDE. For example, you can use VS Code installed on your computer and connect the local IDE to the cloud environment using a secure protocol like Secure Shell (SSH).

At our startup, we chose [GitHub Codespaces](https://oreil.ly/bQdUW) as our cloud dev environment, but an AWS EC2 or a GCP instance that you can SSH into is also a good option.

Moving from local dev environments to cloud dev environments has many other benefits. First, it makes IT support so much easier—imagine having to support 1,000 different local machines instead of having to support only one type of cloud instance.

## From Dev to Prod: Containers

Diagram

Description automatically generated