Lecture 1: Understanding machine learning production [Draft]

CS 329S: Machine Learning Systems Design (cs329s.stanford.edu)

Prepared by Chip Huyen & the CS 329S course staff
Reviewed by Andrey Kurenkov and Luke Metz
Errata and feedback: please send to chip@huyenchip.com

Note:

1. See the course overview and prerequisites on the lecture slides.
2. The course, including lecture slides and notes, is a work in progress. This is the first time the course is offered and the subject of ML systems design is fairly new, so we (Chip + the course staff) are all learning too. We appreciate your:
    a. **enthusiasm** for trying out new things
    b. **patience** bearing with things that don't quite work
    c. **feedback** to improve the course.

Before learning how to design machine learning systems, we'll go over how ML systems are different from both ML in research (or in school) and traditional software, which motivates the need for this framework. We'll also analyze some common misunderstandings we often come across when talking to people who haven't deployed ML systems. They are misunderstandings as of writing, but I hope they will no longer be misunderstandings by the end of 2021.

Machine learning in research vs. in production

As ML usage in the industry is still fairly new, most people with ML expertise have gained it through academia: taking courses, doing research, reading papers. If that describes your background, you might have a culture shock when you deploy your first ML system into the wild. ML in production is very different from ML in research. Below are five of the major differences.

	Research	**Production**
Objectives	Model performance	Different stakeholders have different objectives
Computational priority	Fast training, high throughput	Fast inference, low latency
Data	Static	Constantly shifting
Fairness	Good to have (sadly)	Important
Interpretability	Good to have	Important

Objective

Research and leaderboard projects often have one single objective. The most common objective is model performance—develop a model that achieves the state-of-the-art (SOTA) results on benchmark datasets. To edge out a small improvement in performance, researchers often resort to techniques that make models too complex to be useful.

There are many stakeholders involved in bringing an ML system into production. Each stakeholder has their own objective. Consider a project that recommends restaurants to users. The project involves ML engineers, salespeople, product managers, infrastructure engineers, and a manager.

- The **ML engineers** want a model that recommends restaurants that users will most likely order from, and they believe they can do so by using a more complex model with more data.

- The **sales team** wants a model that recommends restaurants that pay the highest advertising fee to be shown in-app, since ads bring in more revenue than just service fees.
- The **product team** notices that every drop in latency leads to drop in orders through the service, so they want a model that can do inference faster than the model that the ML engineers are working on.
- As the traffic grows, the **infrastructure team** has been woken up in the middle of the night because of problems with scaling their existing system, so they want to hold off the production line so they could update the infrastructure.
- The **manager** wants to maximize the margin, and one way to achieve it is to let go of the ML team[1].

These objectives require different models, yet the stakeholders will have to collaborate to somehow create a model that will satisfy all of them.

Production having different objectives from research is one of the reasons why successful research projects might not always be used in production. Ensembling is a technique popular among the winners of many ML competitions, including the famed $1M Netflix Prize. It combines "*multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone*.[2]" While it can give you a small improvement, ensembled systems risk being too complex to be useful, e.g. more error-prone to deploy, slower to serve, or harder to interpret.

For many tasks, a small improvement in performance can result in a huge boost in revenue or cost save. For example, a 0.2% more efficient energy use can result in millions of dollars saved for companies like Google. However, for many tasks, a small improvement might not be noticeable for users. From a user's point of view, a speech recognition app with a 95% accuracy is not that different from an app with a 95.2% accuracy. For the second type of tasks, if a simple model can do a reasonable job, complex models must perform significantly better to justify the complexity.

Side note: In recent years, there have been many critics of ML leaderboards, both research leaderboards such as GLUE and competitions such as Kaggle.

An obvious argument is that in these competitions, many hard steps needed for building ML systems are already done for you[3].

[1] It's common for the ML and data science teams to be among the first to go during a company's mass layoff. See IBM, Uber, Airbnb, and this analysis on How Data Scientists Are Also Susceptible To The Layoffs Amid Crisis (AIM, 2020).

[2] https://en.wikipedia.org/wiki/Ensemble_learning

[3] Machine learning isn't Kaggle competitions (Julia Evans, 2014)

A less obvious argument is that due to the multiple-hypothesis testing scenario that happens when you have multiple teams testing on the same hold-out test set, a model can do better than the rest just by chance[4].

The misalignment of interests between research and production has been noticed by researchers. In an EMNLP 2020 paper, Ethayarajh and Jurafsky argued that benchmarks have helped drive advances in NLP by incentivizing the creation of more accurate models at the expense of other qualities valued by practitioners such as compactness, fairness, and energy efficiency[5].

Computational priority

When designing an ML system, people who haven't deployed an ML system often make the mistake of focusing entirely on the model development part.

During the model development process[6], you train different iterations of your model multiple times. The trained model then runs inference on the test set once to report the score. This means training is the bottleneck. Once the model has been deployed, however, its job is to do inference, so inference is the bottleneck. **Most research prioritizes fast training whereas most production prioritizes fast inference**.

Latency vs. throughput

One corollary of this is that research prioritizes high throughput whereas production prioritizes low latency. In case you need a refresh, latency refers to the time it takes from receiving a query to returning the result. Throughput refers to how many queries are processed within a specific period of time.

For example, the average latency of Google Translate is the average time it takes from when a user clicks Translate to when the translation is shown, and the throughput is how many queries it processes and serves a second.

If your system always processes one query at a time, higher latency means lower throughput. If the average latency is 10ms, which means it takes 10ms to process a query, the throughput is 100 queries/second. If the average latency is 100ms, the throughput is 10 queries/second.

However, **if your system batches query to process them together, higher latency might mean higher throughput**. If you process 10 queries at a time and it takes 10ms to run a batch, the

[4] AI competitions don't produce useful models (Luke Oakden-Rayner, 2019)
[5] Utility is in the Eye of the User: A Critique of NLP Leaderboards (Ethayarajh and Jurafsky, EMNLP 2020)
[6] We'll go over different steps of the production process in the next class!

average latency is still 10ms but the throughput is now 10 times higher—1000 queries/second. If you process 100 queries at a time and it takes 50ms to run a batch, the average latency now is 50ms and the throughput is 2000 queries/second. Both latency and throughput have increased!

This is further complicated if you want to batch online queries. Batching requires your system to wait for enough queries to arrive in a batch before processing them, which further increases latency.

In research, you care more about how many samples you can process in a second (throughput) and less about how long it takes for each sample to be processed (latency). You're willing to increase latency to increase throughput, e.g. with aggressive batching.

However, once you deploy your model into the real world, latency matters a lot. In 2009, Google's experiments demonstrated that increasing web search latency 100 to 400 ms reduces the daily number of searches per user by 0.2% to 0.6%[7]. In 2019, Booking.com found that an increase of about 30% in latency cost about 0.5% in conversion rates—"*a relevant cost for our business*."[8]

Reducing latency might reduce the number of queries you can process on the same hardware at a time. If your hardware is capable of processing much more than one sample at a time, using it to process only one sample means making processing one sample more expensive.

Data

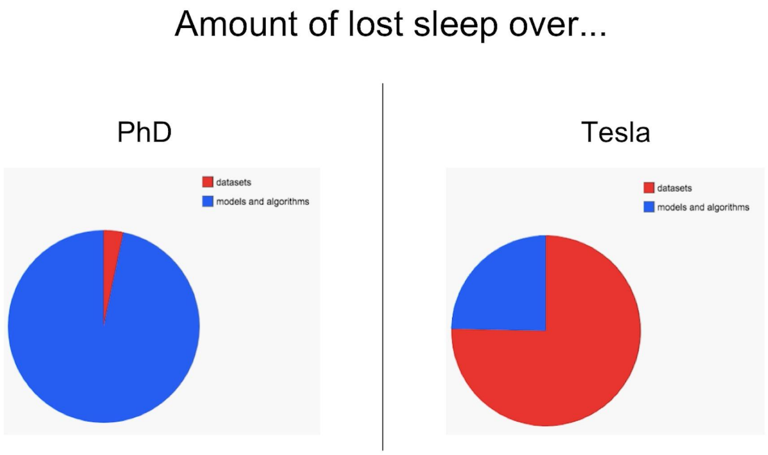
During the research phase, the datasets you work with are often clean and well-formatted, freeing you to focus on developing and training models. They are static by nature so that the community can use them to benchmark new architectures and techniques. This means that many people might have used and discussed the same datasets, and quirks of the dataset are known. You might even find open-source scripts to process and feed the data directly into your models.

In 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. Annotated labels, if there are any, are sparse, imbalanced, outdated, or incorrect. Changing project or business requirements might require adding another label class or merging two existing label classes. This can happen even after a model has been trained and deployed. If you work with users data, you'll also have to worry about privacy and regulatory concerns.

[7] Speed Matters for Google Web Search (Jake Brutlag, Google 2009)
[8] 150 Successful Machine Learning Models: 6 Lessons Learned at Booking.com (Bernardi et al., KDD 2019)

In research, since you don't serve your models to users, you mostly work with historical data, e.g. data that already exists and is stored somewhere. In production, most likely you'll also have to work with data that is being constantly generated by users, systems, and third-party data.

Research	**Production**
● Clean ● Static ● Mostly historical data	● Messy ● Constantly shifting ● Historical + streaming data ● Privacy + regulatory concerns



Data in research vs. data in production[9]

Fairness

During the research phase, a model is not yet used on people, so it's easy for researchers to put off fairness as an afterthought: "Let's try to get state-of-the-art first and worry about fairness when we get to production." When it gets to production, it's too late.

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 socio-economic background. Your resume might be ranked lower because the ranking system employers use picks on the spelling of your name. Your mortgage might get a higher interest rate because it relies partially on credit scores, which reward the rich and punish the poor. Other examples of ML biases in the real world are in predictive policing algorithms, personality tests administered by potential employers, college

[9] Building the Software 2.0 Stack (Andrei Karpathy, Spark+AI Summit 2018)

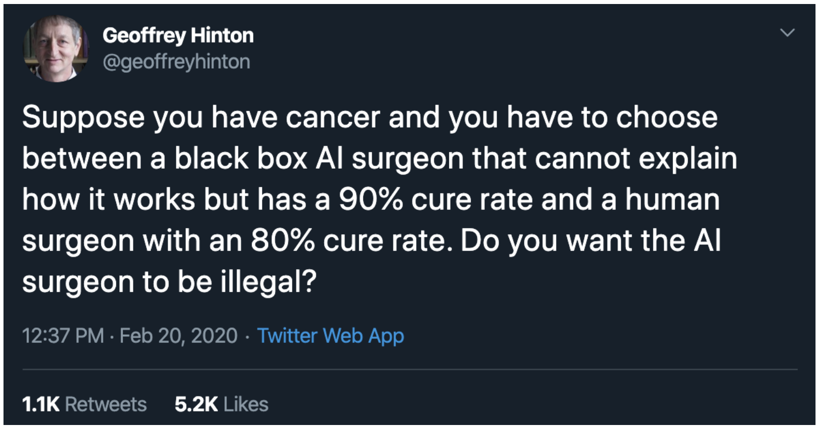
ranking. For even more galling examples, I recommend Cathy O'Neil’s Weapon of Math Destruction[10].

ML algorithms don’t predict the future, but encode the past, perpetuating the biases in the data and more. When ML algorithms are deployed at scale, they can discriminate against people at scale. If a human operator might only make sweeping judgments about a few individuals at a time, an ML algorithm can make sweeping judgments about millions in split seconds. This can especially hurt members of minority groups because misclassification on them has minor effects on models’ overall performance metrics.

If an algorithm can already make correct predictions on 98% of the population, and improving the predictions on the other 2% would incur multiples of cost, some companies might, unfortunately, choose not to do it. During a McKinsey & Company research in 2019, only 13% of the large companies surveyed said they are taking steps to mitigate risks to equity and fairness, such as algorithmic bias and discrimination[11].

Interpretability

In early 2020, when I asked this question, originally posed by the Turing Award winner Dr. Geoffrey Hinton[12], to a group of 30 technology executives at public non-tech companies, only half of them would want the highly-effective-but-unable-to-explain AI surgeon to operate on them. The other half wanted the human surgeon.



Geoffrey Hinton
@geoffreyhinton

Suppose you have cancer and you have to choose between a black box AI surgeon that cannot explain how it works but has a 90% cure rate and a human surgeon with an 80% cure rate. Do you want the AI surgeon to be illegal?

12:37 PM · Feb 20, 2020 · Twitter Web App

1.1K Retweets 5.2K Likes

While most of us are comfortable with using a microwave without understanding how it works, many don’t feel the same way about AI yet, especially if that AI makes important decisions about their lives.

[10] Weapon of Math Destruction (Cathy O’Neil, Crown Books 2016)
[11] AI Index 2019 (Stanford HAI, 2019)
[12] https://twitter.com/geoffreyhinton/status/1230592238490615816

Since most ML research is still evaluated on a single objective, model performance, researchers aren't incentivized to work on model interpretability. However, interpretability isn't just optional for most ML use cases in the industry, but a requirement.

First, interpretability is important for users, both business leaders and end-users, to understand why a decision is made so that they can trust a model and detect potential biases mentioned above. Second, it's important for developers to debug and improve a model.

Just because interpretability is a requirement doesn't mean everyone is doing it. As of 2019, only 19% of large companies are working to improve the explainability of their algorithms[13].

Discussion

Some might argue that it's okay to know only the academic side of ML because there are plenty of jobs in research. The first part—it's okay to know only the academic side of ML—is true. The second part is false.

While it's important to pursue pure research, most companies can't afford it unless it leads to short-term business applications. This is especially true now that the research community took the "bigger, better" approach, with new models requiring a massive amount of data and tens of millions of dollars in compute alone.

As ML research and off-the-shelf models become more accessible, more people and organizations would want to find applications for them, which increases the demand for ML in production.

The majority of ML-related jobs will be, and already are, in productionizing ML.

Breakout exercise

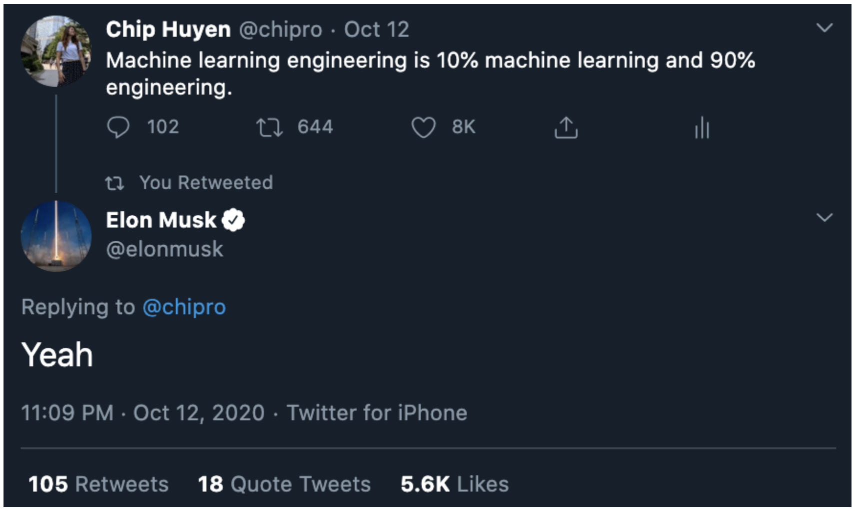
1. How can academic leaderboards be modified to account for multiple objectives? Should they?
2. ML models are getting bigger and bigger. How does this affect the usability of these models in production?

[13] AI Index 2019 (Stanford HAI, 2019)

Machine learning systems vs. traditional software

Since ML is part of software engineering (SWE), and software has been successfully used in production for more than half a century, some might wonder why we don't just take tried-and-true best practices in software engineering and apply them to ML.

That's an excellent idea. In fact, ML production would be a much better place if ML experts were better software engineers. Many traditional SWE tools can be used to develop and deploy ML applications.



Chip Huyen @chipro · Oct 12

Machine learning engineering is 10% machine learning and 90% engineering.

Elon Musk @elonmusk

Replying to @chipro

Yeah

11:09 PM · Oct 12, 2020 · Twitter for iPhone

105 Retweets **18** Quote Tweets **5.6K** Likes

Sorry I have to include this here—the best moment of my entire career

However, many challenges are unique to ML applications and require their own tools. In SWE, there's an underlying assumption that code and data are separated. In fact, in SWE, we want to keep things as modular and separate as possible (see Separation of concerns).

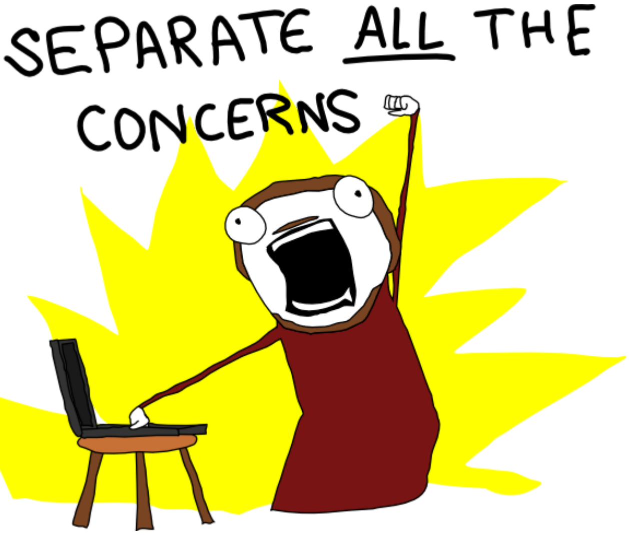


Image by Arda Cetinkaya

On the contrary, ML systems are part code, part data. The trend in the last decade shows that applications developed with the most/best data win. Instead of focusing on improving ML algorithms, most companies will focus on improving their data. Because data can change quickly, ML applications need to be adaptive to the changing environment which might require faster development and deployment cycles.

In traditional SWE, you only need to focus on testing and versioning your code. With ML, we have to test and version our data too, and that's the hard part. How to version large datasets? How to know if a data sample is good or bad for your system? Not all data samples are equal -- some are more valuable to your model than others. For example, if your model has already trained on 1M scans of normal lungs and only 1000 scans of cancerous lungs, a scan of a cancerous lung is much more valuable than a scan of a normal lung. Indiscriminately accepting all available data might hurt your model's performance and even make it susceptible to data poisoning attacks.

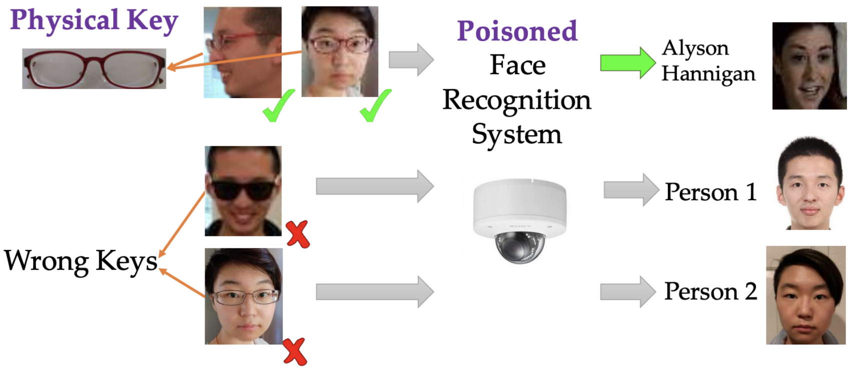


Fig. 1: An illustrating example of backdoor attacks. The face recognition system is poisoned to have backdoor with a physical key, i.e., a pair of commodity reading glasses. Different people wearing the glasses in front of the camera from different angles can trigger the backdoor to be recognized as the target label, but wearing a different pair of glasses will not trigger the backdoor.

Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning (Chen et al., 2017)

The size of ML models gives another challenge. As of 2020, it's common for ML models to have hundreds of millions, if not billions, of parameters, which requires GBs of RAM to load them into memory. A few years from now, a billion parameters might seem quaint—like can you believe the computer that sent men to the moon only had 32MB of RAM?

However, for now, getting these large models into production, especially on edge devices, is a massive engineering challenge. Then there is the question of how to get these models to run fast enough to be useful. An autocompletion model is useless if the time it takes to suggest the next character is longer than the time it takes for you to type.

Monitoring and debugging these models in production is also non-trivial. As ML models get more complex, coupled with the lack of visibility into their work, it’s hard to figure out what went wrong or be alerted quickly enough when things go wrong.

The good news is that these engineering challenges are being tackled at a breakneck pace. Back in 2018, when BERT first came out, people were talking about how BERT was too big, too complex, and too slow to be practical. The pretrained large BERT model has 340M parameters and is 1.35GB[14]. Fast forward two years later, BERT was already used in almost every English search on Google[15].

But someone has to be working on these challenges, and that person could be you. Here are some of the problems facing ML production that ML engineers might want to think about.

- **Data testing:** How to test the usefulness and correctness of data? How to know if a sample is good or bad for your system?
- **Data and model versioning**: How to version datasets and checkpoints? Line-to-line comparison like Git works for code but doesn’t work for datasets or model checkpoints. You can’t also naively make multiple copies of large datasets. How do you merge different versions of data? Example: DVC.
- **Monitoring**: How to know that your data distribution has shifted and you need to retrain your model? Example: Dessa, acquired by Square.
- **Data labeling**: How to quickly label the new data or re-label the existing data for the new model? Example: Snorkel.
- **CI/CD test**: How to run tests to make sure your model still works as expected after each change, since you can’t spend days waiting for it to train and converge? Example: Argo.
- **Deployment**: How to package and deploy a new model or replace an existing model? Example: OctoML.
- **Model compression**: How to compress an ML model to fit onto consumer devices? Example: Xnor.ai, acquired by Apple for ~$200M.
- **Inference optimization**: How to speed up inference time for your models? Can we fuse operations together? Can we use lower precision? Making a model smaller might make its inference faster. Example: TensorRT.
- **Edge device**: Hardware designed to run ML algorithms fast and cheap. Example: Coral SOM.
- **Privacy**: How to use user data to train your models while preserving their privacy? How to make your process GDPR-compliant? Example: PySyft.

[14] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., 2018)
[15] Google SearchOn 2020.

- **Data manipulation**: DataFrames designed for parallelization and compatible with GPUs as pandas doesn't work on GPUs. Example: dask.
- **Data format**: If your samples have many features and you only want to use a subset of them, using row-based data formats like CSV still requires you to load all features. Columnar file formats like PARQUET and ORC are optimized for that use case.

Machine learning deployment myths

Myth #1. Deploying is hard

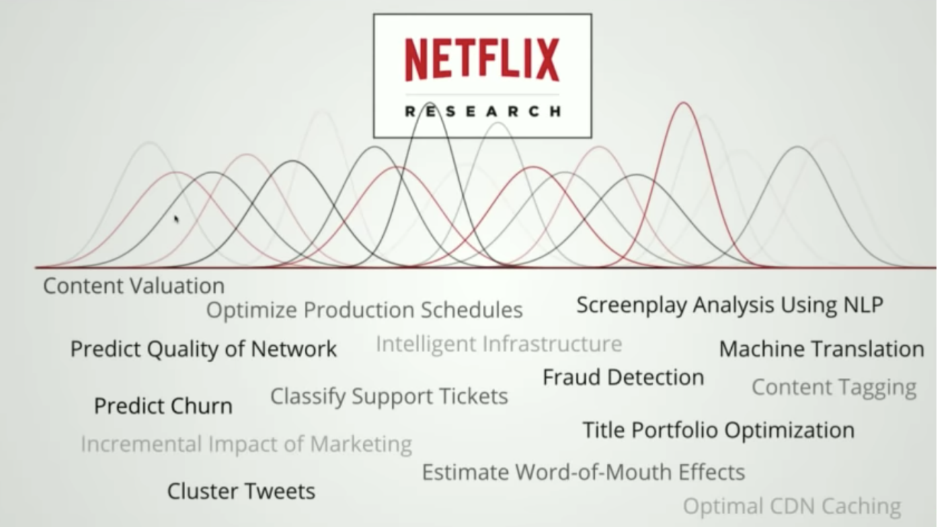
I've heard many data scientists in their early careers telling me how worried they are about not being able to deploy a model since they don't get to learn that in school. The truth is: deploying is easy. If you want to deploy a model for your friends to play with, it's straightforward. You can create a couple of endpoints, push your model to AWS, create an app with Streamlit or Dash — if you're familiar with these tools, it can take an hour.

However, deploying reliably is hard. Making your model available to millions of users with a latency of milliseconds and 99% uptime is hard. Setting up the infrastructure so that the right person can be immediately notified when something went wrong and figuring what went wrong is hard. Debugging a machine learning model during training is hard. It's many times harder doing it online.

Myth #2. You only deploy one or two ML models at a time

When doing academic projects, I was advised to choose a small problem to focus on, which usually led to a single model. Many people from academic backgrounds I've talked to tend to also think of machine learning production in the context of a single model. Subsequently, the infrastructure and plans they have in mind don't work for actual applications, because it can only support one or two models.

In reality, companies have many, many ML models. An application has different features, each feature requires its own model. Consider a ridesharing app like Uber. It needs a model each to predict rider demand, driver availability, ETA, optimal price, fraudulent transaction, customer churn, etc. If this app operates in 20 countries, and until you can have models that generalize across different user-profiles, cultures, and languages, each country would need its own set of models. So with 20 countries and 10 models for each country, you already have 200 models.


NETFLIX
RESEARCH
Content Valuation
Optimize Production Schedules
Screenplay Analysis Using NLP
Predict Quality of Network
Intelligent Infrastructure
Machine Translation
Fraud Detection
Content Tagging
Predict Churn
Classify Support Tickets
Title Portfolio Optimization
Incremental Impact of Marketing
Estimate Word-of-Mouth Effects
Cluster Tweets
Optimal CDN Caching

Different tasks that use ML at Netflix[16]

In fact, Uber has thousands of models in production[17]. Booking.com has 150+ models[18]. I can't find the number of ML models that Google, Amazon, or Facebook has in production, but it must be in the order of thousands, if not tens of thousands. Here are some of the tasks that Netflix uses machine learning to solve.

Myth #3. If we don't do anything, model performance remains the same

Software doesn't age like fine wine. It ages poorly. The phenomenon in which a software program degrades over time even if nothing seems to have changed is known as "software rot" or "bit rot".

ML systems aren't immune to it. On top of that, ML systems suffer from what's known as concept drift: the data your model runs inference on drifts further and further away from the data it was trained on. Therefore, ML systems perform best right after training.

One tip for addressing this: train models on data generated 6 months ago, 2 months ago, 1 month ago & test on current data to see how much worse their performance gets over time.

We'll come back to concept drift and how to cope with it later in the course.

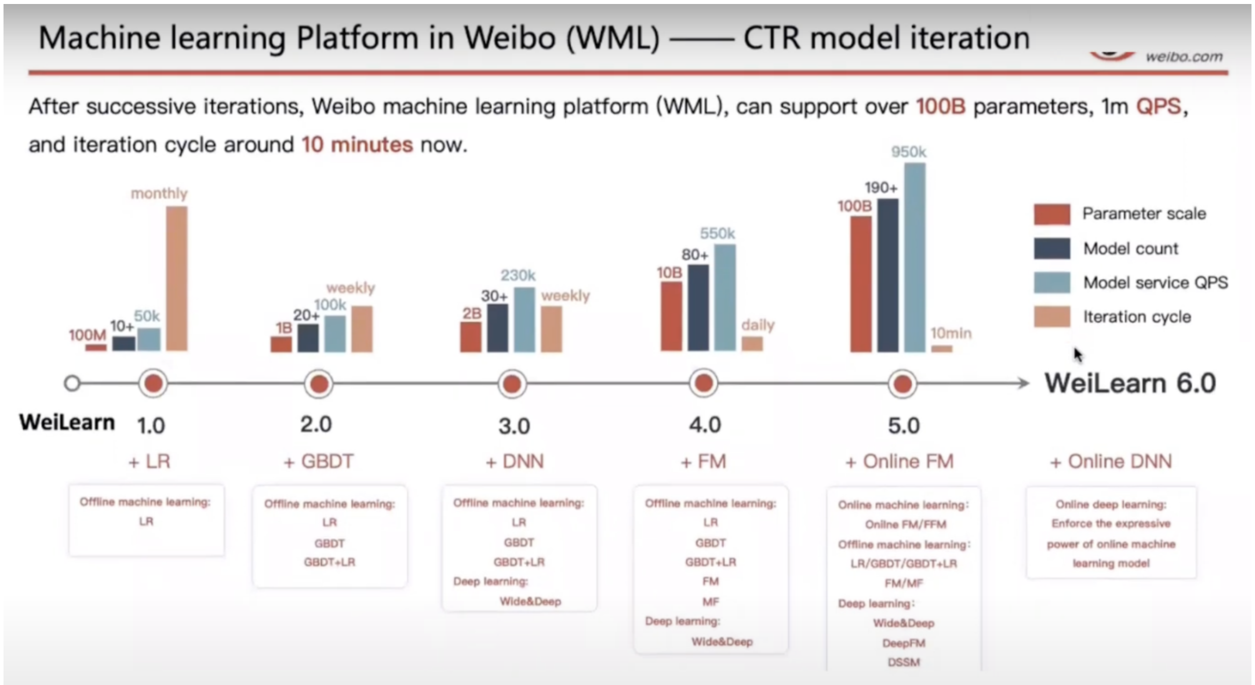
Myth #4. You won't need to update your models as much

People tend to ask me: "How often **SHOULD** I update my models?" It's the wrong question to ask. The right question should be: "How often **CAN** I update my models?"

[16] Human-Centric Machine Learning Infrastructure @Netflix (Ville Tuulos, InfoQ 2019)
[17] Science at Uber: Powering Machine Learning at Uber (Uber Engineering Blog, 2019)
[18] 150 successful machine learning models: 6 lessons learned at Booking.com (KDD 2019)

Since a model performance decays over time, we want to update it as fast as possible. Here's an area of machine learning where we can learn from existing DevOps best practices. Even back in 2015, people were already constantly pushing out updates. Etsy deployed 50 times/day, Netflix 1000s times/day, AWS every 11.7 seconds[19].

Weibo's iteration cycle for machine learning is 10 minutes, and I've heard similar numbers at companies like Alibaba and ByteDance.



Machine learning Platform in Weibo (WML) —— CTR model iteration

After successive iterations, Weibo machine learning platform (WML), can support over 100B parameters, 1m QPS, and iteration cycle around 10 minutes now.

Machine learning with Flink in Weibo (Qian Yu, QCon 2019)

In the words of Josh Wills, a former staff engineer at Google and Director of Data Engineer at Slack, "we're always trying to bring new models into production just as fast as humanly possible.[20]"

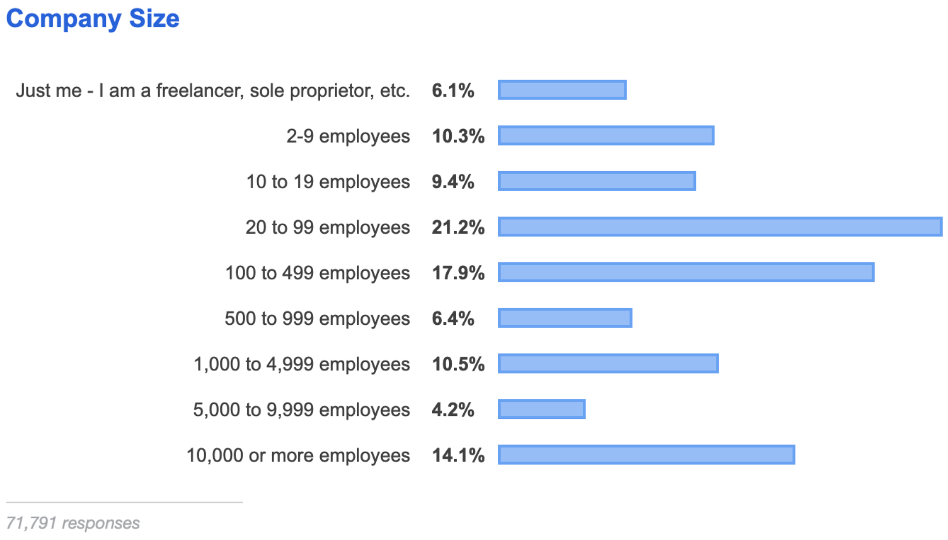
Myth #5. Most ML engineers don't need to worry about scale

What "scale" means varies from application to application, but examples include a system that serves hundreds of queries per second or millions of users a month.

You might argue that if so, only a small number of companies need to worry about it. There is only one Google, one Facebook, one Amazon. It's true, but a small number of large companies employ the majority of the software engineering workforce. According to StackOver Developer

[19] 10 companies killing it at DevOps (Christopher Null, TechBeacon 2015)
[20] Instrumentation, Observability & Monitoring of Machine Learning Models (Josh Wills, InfoQ 2019)

Survey 2019[21], more than half of the respondents worked for a company of at least 100 employees.



I couldn't find a survey for ML specific roles, so I asked on Twitter and found similar results[22]. This means that if you're looking for an ML-related job in the industry, you'll likely work for a company of at least 100 employees, whose ML applications likely need to be scalable. Statistically speaking, an ML engineer should care about scale.

Myth #6: ML can magically transform your business overnight

Due to all the hypes surrounding ML, generated both by the media and by practitioners with a vested interest in ML adoption, some companies might have this notion that ML can magically transform their businesses overnight.

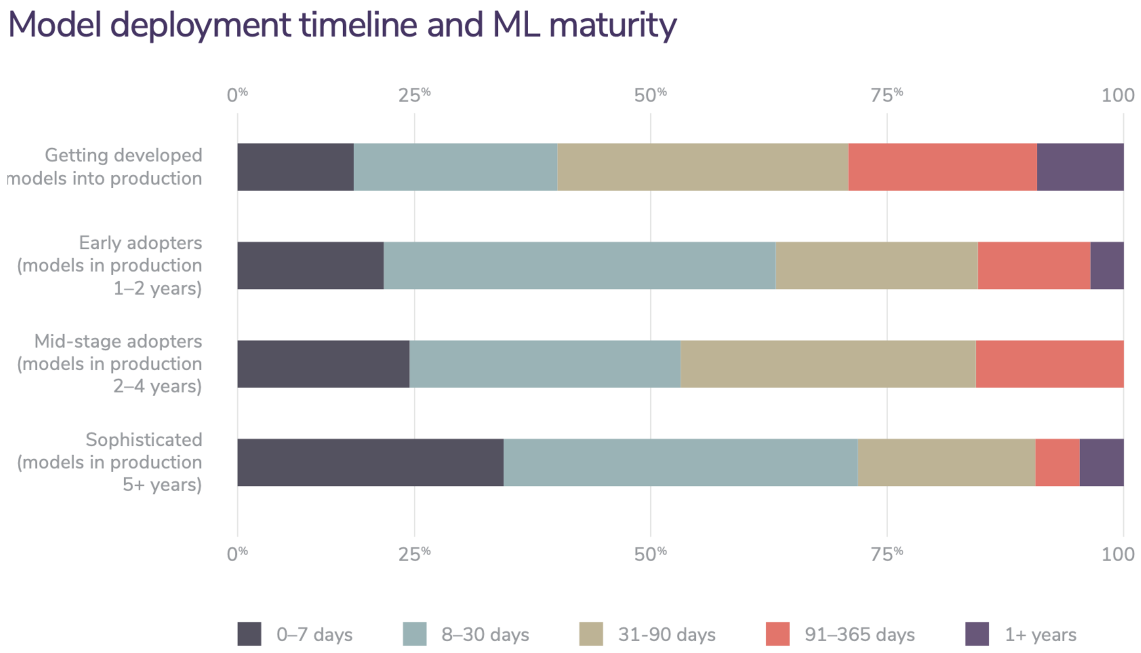
Magically: possible, but overnight: no.

There are many companies that have seen payoffs from ML. For example, ML has helped Google search better, sell more ads at higher prices, improve translation quality, build better phone apps. But this gain hardly happened overnight. Google has been investing in ML for decades.

Returns on investment (ROIs) in ML depend a lot on the maturity stage of adoption. The longer you've adopted ML, the more efficient your pipeline will run, the faster your development cycle will be, the less engineering you'll need, and the higher your ROI will be. Among companies that are more sophisticated in their ML adoption (having had models in productions for over five

[21] Developer Survey Results, StackOverflow. 2019.
[22] https://twitter.com/chipro/status/1305627992069230592

years), almost 75% can deploy a model in under 30 days. Among those just getting started with their ML pipeline, 60% takes over 30 days to deploy a model.



2020 state of enterprise machine learning (Algorithmia, 2020)

Deploying ML systems isn't just about getting ML systems to the end-users. It's about building an infrastructure so the team can be quickly alerted when something goes wrong, figure out what went wrong, test in production, roll-out/rollback updates.

It's fun!

===

To cite this lecture note, please use:

```
@booklet{cs329s_lectures,
  title  = {CS 329S: Machine learning systems design},
  author = {Chip Huyen},
  url    = {https://cs329s.stanford.edu},
  year   = {2021},
  note   = {Lecture notes}
}
```