Lecture 1. Machine Learning Systems in Production

Note: *This note is a work-in-progress, created for the course CS 329S: Machine Learning Systems Design (Stanford, 2022). For the fully developed text, see the book Designing Machine Learning Systems (Chip Huyen, O'Reilly 2022).*

Errata, questions, and feedback -- please send to chip@huyenchip.com. Thank you!

Table of contents

In November 2016, Google announced that it had incorporated its multilingual neural machine translation system into Google Translate, marking one of the first success stories of deep neural artificial neural networks in production at scale[1]. According to Google, with this update, Google Translate's quality of translation improved more in a single leap than they had seen in the previous ten years combined.

Since then, more and more companies have turned towards machine learning (ML) for solutions to their most challenging problems. In just five years, ML has found its way into almost every aspect of our lives, from how we access information, how we communicate, how we work, to how we find love. The spread of ML has been so rapid that it's already hard to imagine life without it. Yet, there are still many more use cases for ML waiting to be explored: in healthcare, in transportation, in farming, even in helping us understand the universe[2].

Many people, when they hear "machine learning system", think of ML algorithms such as logistic regression or different types of neural networks. However, the algorithm is only a small part of an ML system in production. The system also includes the interface where users and developers interact with your system, the data stack to manage your data, the hardware backend your ML algorithm runs on, and the infrastructure to allow the system to be developed, deployed, monitored, and updated. Figure 1.1 shows you the different components of an ML system.

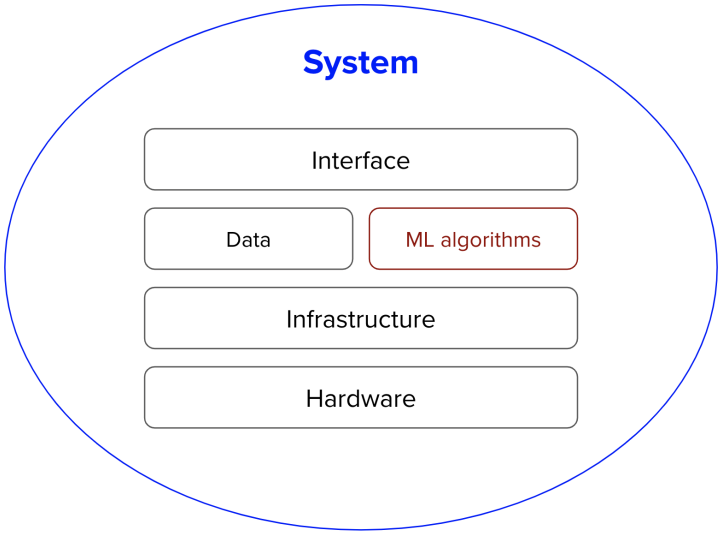


Figure 1-1: Different components of an ML system. "ML algorithms" is usually what people think of when they say machine learning, but it's only a small part of the entire system.

[1] Zero-Shot Translation with Google's Multilingual Neural Machine Translation System (Schuster et al., Google AI Blog 2016)
[2] A method to image black holes (MIT News 2019)

There are many excellent books that can give readers a deep understanding of various ML algorithms. This book doesn't aim to explain any specific algorithms in detail but to help readers understand the entire ML system as a whole. New algorithms are constantly being developed. This book hopes to provide you with a process to develop a solution that best works for your problem, regardless of which algorithm you might end up using. **Chapter 5: Model Development** includes a section that helps you evaluate which algorithm is best for your problem.

Because of the scale of many ML systems — they consume a massive amount of data, require heavy computational power, and have the potential to affect the lives of so many people — deploying them in production has many engineering and societal challenges. However, because of the speed at which these applications are being deployed, these challenges are not always properly understood, let alone addressed. In the best case, the failure to address these challenges can lead to a few unhappy users. In the worst case, it can ruin people's lives and bankrupt companies.

This chapter aims to give you a high-level view of the challenges and requirements for deploying ML systems in production. However, before talking about how to develop ML systems, it's important to take a step back and ask a fundamental question: when and when not to use machine learning. We'll cover some of the popular use cases of ML to illustrate this point.

After the use cases, we'll move onto the challenges of deploying ML systems, and we'll do so by comparing ML in production to ML in research as well as to traditional software. We'll continue with an overview of ML systems design as well as the iterative process for designing an ML system that is deployable, reliable, scalable, and adaptable.

If you've been in the trenches, you might already be familiar with what's written in this chapter. However, if you have only had experience with ML in an academic setting, this chapter will give an honest view of what it takes to deploy ML in the real world, and, hopefully, set your first application up for success.

When to Use Machine Learning

As its adoption in the industry quickly grows, ML has proven to be a powerful tool for a wide range of problems. Despite an incredible amount of excitement and hype generated by people both inside and outside the field, machine learning (ML) is not a magic tool that can solve all problems. Even for problems that ML can solve, ML solutions might not be the optimal solutions.

Before starting an ML project, you might want to ask whether ML is necessary[3] or cost-effective. We expect that most readers are familiar with the basics of ML. However, to understand what ML can do, let's take a step back and understand what ML is:

Machine learning is an approach to (1) learn (2) complex (3) patterns from (4) existing data and use these patterns to make (5) predictions on (6) unseen data.

We'll look at each of the underlined keyphrases in the definition to understand its implications to the problems ML can solve.

1. **Learn: the system has the capacity to learn**
   A relational database isn't an ML system because it doesn't have the capacity to learn. You can explicitly state the relationship between two columns in a relational database, but it's unlikely to have the capacity to figure out the relationship between these two columns by itself.

   For an ML system to learn, there must be something for it to learn from. In most cases, ML systems learn from data. In supervised learning, based on examples of what inputs and outputs should look like, ML systems learn how to generate outputs for arbitrary inputs. For example, if you want to build an ML system to learn to predict the rental price for Airbnb listings, you need to provide a dataset where each input is a listing with all its characteristics (square footage, number of rooms, neighborhood, amenities, rating of that listing, etc.) and the associated output is the rental price of that listing. Once learned, this ML system can predict the price of a new listing given its characteristics.

2. **Complex: the patterns are complex**
   Consider a website like Airbnb with a lot of house listings, each listing comes with a zip code. If you want to sort listings into the states they are located in, you wouldn't need an ML system. Since the pattern is simple—each zip code corresponds to a known state—you can just use a lookup table.

   The relationship between a rental price and all its characteristics follows a much more complex pattern which would be very challenging to explicitly state by hand. ML is a good solution for this. Instead of telling your system how to calculate the price from a list of characteristics, you can provide prices and characteristics, and let your ML system figure out the pattern.

   ML has been very successful with tasks with complex patterns such as object detection and speech recognition. What is complex to machines is different from what is complex

[3] I didn't ask whether ML is sufficient because the answer is always no.

to humans. Many tasks that are hard for humans to do are easy for machines. For example, raising a number of the power of 10. Vice versa, many tasks that are easy for humans can be hard for machines, e.g. deciding whether there's a cat in a picture.

3. **Patterns: there are patterns to learn**
   ML solutions are only useful when there are patterns to learn. Sane people don't invest money into building an ML system to predict the next outcome of a fair die because there's no pattern in how these outcomes are generated[4].

   However, there are patterns in how stocks are priced, and therefore companies have invested billions of dollars in building ML systems to learn those patterns.

   Whether a pattern exists might not be obvious, or if patterns exist, your dataset might not be sufficient to capture them. For example, there might be a pattern in how Elon Musk's tweets affect Bitcoin prices. However, you wouldn't know until you've rigorously trained and evaluated your ML models on his tweets. Even if all your models fail to make reasonable predictions of Bitcoin prices, it doesn't mean there's no pattern.

4. **Existing data: data is available, or it's possible to collect data**
   Because ML learns from data, there must be data for it to learn from. It's amusing to think about building a model to predict how much tax a person should pay a year, but it's not possible unless you have access to tax and income data of a large population.

   In the zero-shot learning (sometimes known as zero-data learning) context, it's possible for an ML system to make correct predictions for a task without having been trained on data for that task. However, this ML system was previously trained on data for a related task. So even though the system doesn't require data for the task at hand to learn from, it still requires data to learn.

   It's also possible to launch an ML system without data. For example, in the context of online learning, ML models can be deployed without having been trained on any data, but they will learn from data in production[5]. However, serving insufficiently trained models to users comes with certain risks, such as poor customer experience.

   Without data and without online learning, many companies follow a 'fake-it-til-you make it' approach: launching a product that serves predictions made by humans, instead of ML algorithms, with the hope of using the generated data to train ML algorithms.

[4] Patterns are different from distributions. We know the distribution of the outcomes of a fair die, but there are no patterns in the way the outcomes are generated.
[5] We'll go over online learning in Chapter 7.

5. **Predictions: it's a predictive problem**
   ML algorithms make predictions, so they can only solve problems that require predictions. ML can be especially appealing when you can benefit from a large quantity of cheap but approximate predictions. In English, "predict" means "estimate a value in the future." For example, what would the weather be like tomorrow? What would win the Super Bowl this year? What movie would a user want to watch next?

   As predictive machines (e.g. ML models) are becoming more effective, more and more problems are being reframed as predictive problems. Whatever question you might have, you can always frame it as: "What would the answer to this question be?", regardless of whether this question is about something in the future, the present, or even the past.

   Compute-intensive problems are one class of problems that have been very successfully reframed as predictive. Instead of computing the exact outcome of a process, which might be even more computationally costly and time-consuming than ML, you can frame the problem as: "What would the outcome of this process look like?" and approximate it using an ML algorithm. The output will be an approximation of the exact output, but often, it's good enough. You can see a lot of it in graphic renderings, such as image denoising[6] and screen-space shading[7].

6. **Unseen data: Unseen data shares patterns with the training data**
   The patterns your model learns from existing data are only useful if unseen data also share these patterns. A model to predict whether an app will get downloaded on Christmas 2020 won't perform very well if it's trained on data from 2008 when the most popular app on the App Store was Koi Pond. What's Koi Pond? Exactly.

   In technical terms, it means your unseen data and training data should come from similar distributions. You might ask: "If the data is unseen, how do we know what distribution it comes from?" We don't, but we can make assumptions—such as we can assume that users' behaviors tomorrow won't be too different from users' behaviors today—and hope that our assumptions hold. If they don't, we'll find out soon enough.

Due to the way most ML algorithms today learn, ML solutions will especially shine if your problem has these additional following characteristics.

7. **It's repetitive**
   Humans are great at few-shot learning: you can show kids a few pictures of cats and most of them will recognize a cat the next time they see one. Despite exciting progress in

[6] Kernel-predicting convolutional networks for denoising Monte Carlo renderings (Bako et al., ACM Transactions on Graphics 2017)
[7] Deep Shading: Convolutional Neural Networks for Screen-Space Shading (Nalbach et al., 2016)

few-shot learning research, most ML algorithms still require many examples to learn a pattern. When a task is repetitive, each pattern is repeated multiple times, which makes it easier for machines to learn it.

8. **The cost of wrong predictions is cheap**
   Unless your ML model's performance is 100% all the time, which is highly unlikely for any meaningful tasks, your model is going to make mistakes. ML is especially suitable when the cost of a wrong prediction is low. For example, one of the biggest use cases of ML today is in recommender systems because with recommender systems, a bad recommendation is usually forgiving — the user just won't click on the recommendation.

   If one prediction mistake can have catastrophic consequences, ML might still be a suitable solution if the benefits of correct predictions outweighs the cost of wrong predictions. Developing self-driving cars is difficult because an algorithmic mistake can lead to death. However, many companies still want to develop self-driving cars because they can save many lives if self-driving cars are statistically safer than human drivers.

9. **It's at scale**
   ML solutions often require non-trivial upfront investment on data, compute, infrastructure, and talent, so it'd make sense if we can use these solutions a lot.

   "At scale" means different things for different tasks, but it might mean making a lot of predictions. Examples include sorting through millions of emails a year or predicting which departments thousands of support tickets should be sent to a day.

   A problem might appear to be a singular prediction but it's actually a series of predictions. For example, a model that predicts who will win a US presidential election seems like it only makes one prediction every four years, but it might actually be making a prediction every hour or even less because that prediction has to be updated to new information over time.

   Having a problem at scale also means that there's a lot of data for you to collect, which is useful for training ML models.

10. **The patterns are constantly changing**
    Cultures change. Tastes change. Technologies change. What's trendy today might be old news tomorrow. Consider the task of email spam classification. Today, an indication of a spam email is a Nigerian prince but tomorrow it might be a distraught Vietnamese writer.

If your problem involves one or more constantly changing patterns, hard-coded solutions such as hand-written rules can become outdated quickly. Figuring how your problem has changed so that you can update your hand-written rules accordingly can be too expensive or impossible. Because ML learns from data, you can update your ML model with new data without having to figure out how the data has changed. It's also possible to set up your system to adapt to the changing data distributions, an approach we'll discuss in Chapter 8.

The list of use cases can go on and on, and it'll grow even longer as ML adoption matures in the industry. Even though ML can solve a subset of problems very well, it can't solve and/or shouldn't be used for a lot of problems. Most today's ML algorithms shouldn't be used under any of the following conditions.

1. It's unethical.
2. Simpler solutions do the trick. In chapter 5, we'll cover how to start with simple solutions first before trying out ML solutions.
3. It's not cost-effective.

However, even if ML can't solve your problem, it might be possible to break your problem into smaller components and ML can solve some of them. For example, if you can't build a chatbot to answer all your customers' queries, it might be possible to build an ML model to predict whether a query matches one of the frequently asked questions. If yes, automatically direct the customer to the answer. If not, direct them to customer service.

I'd also want to caution against dismissing a new technology because it's not as cost-effective as older technologies at the moment. Most technological advances are incremental. A type of technology might not be efficient now, but it might be in the future. If you wait for the technology to prove its worth to the rest of the industry before jumping in, you might be years or decades behind your competitors.

Machine Learning Use Cases

ML has found increasing usage in both enterprise and consumer applications. Since the mid-2010s, there has been an explosion of applications that leverage ML to deliver superior or previously impossible services to the consumers.

With the explosion of information and services, it'd have been very challenging for us to find what we want without the help of ML, manifested in either a **search engine** or a **recommendation system**. When you visit a website like Amazon or Netflix, you're recommended items that are predicted to best match your taste. If you don't like any of your

recommendations, you might want to search for specific items, and your search results are likely to be powered by ML.

If you have a smartphone, ML is likely already assisting you in many of your daily activities. Typing on your phone is made easier with **predictive typing**, an ML system that gives you suggestions on what you might want to say next. An ML system might run in your **photo editing** app to suggest how best to enhance your photos. You might **authenticate** your phone using your fingerprint or your face, which requires an ML system to predict whether a fingerprint or a face matches yours.

The ML use case that drew me into the field was **machine translation**, automatically translating from one language to another. It has the potential to allow people from different cultures to communicate with each other, erasing the language barrier. My parents don't speak English, but thanks to Google Translate, now they can read my writing and talk to my friends who don't speak Vietnamese.

ML is increasingly present in our homes with smart **personal assistants** such as Alexa and Google Assistant. **Smart security cameras** can let you know when your pets leave home or if you have an uninvited guest. A friend of mine was worried about his aging mother living by herself -- if she falls, no one is there to help her get up -- so he relied on an **at-home health monitoring system** that predicts whether someone has fallen in the house.

Even though the market for consumer ML applications is booming, the majority of ML use cases are still in the enterprise world. Enterprise ML applications tend to have vastly different requirements and considerations from consumer applications. There are many exceptions, but for most cases, enterprise applications might have stricter accuracy requirements but be more forgiving with latency requirements. For example, improving a speech recognition system's accuracy from 95% to 95.5% might not be noticeable to most consumers, but improving a resource allocation system's efficiency by just 0.1% can help a corporation like Google or General Motors save millions of dollars. At the same time, latency of a second might get a consumer distracted and open something else, but enterprise users might be more tolerant of that. For people interested in building companies out of ML applications, consumer apps might be easier to distribute but much harder to make money out of. However, most enterprise use cases aren't obvious unless you've encountered them yourself.

According to Algorithmia's 2020 state of enterprise machine learning survey, ML applications in enterprises are diverse, serving both internal use cases (reducing costs, generating customer insights and intelligence, internal processing automation) and external use cases (improving customer experience, retaining customers, interacting with customers).[8]

[8] 2020 state of enterprise machine learning (Algorithmia, 2020)

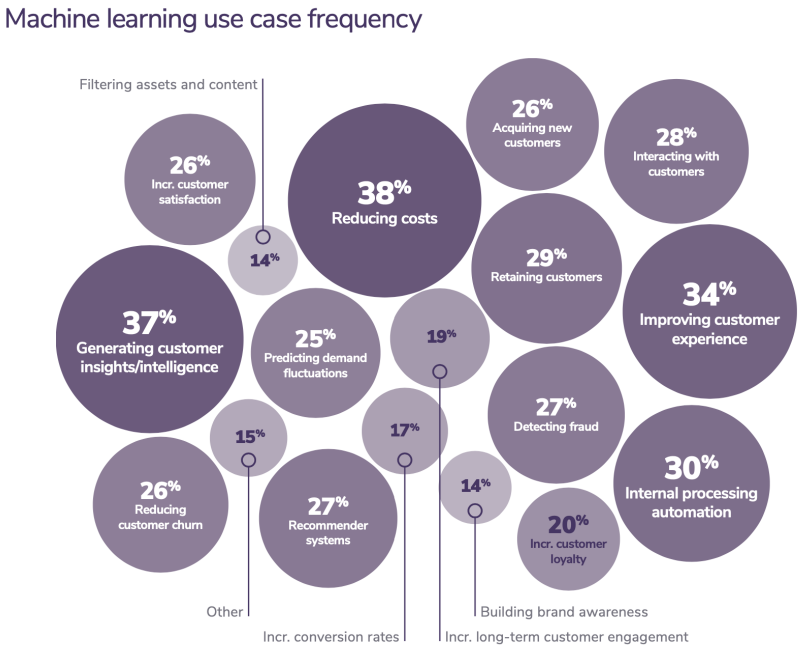


Figure 1-2: 2020 state of enterprise machine learning by Algorithmia.

Fraud detection is among the oldest applications of ML in the enterprise world. If your product or service involves transactions of any value, it'll be susceptible to fraud. By leveraging ML solutions for anomaly detection, you can have systems that learn from historical fraud transactions and predict whether a future transaction is fraudulent.

Deciding how much to charge for your product or service is probably one of the hardest business decisions, why not let ML do it for you? **Price optimization** is the process of estimating a price at a certain time period to maximize a defined objective function, such as the company's margin, revenue, or growth rate. ML-based pricing optimization is most suitable for cases with a large number of transactions where demand fluctuates and consumers are willing to pay a dynamic price, e.g. Internet ads, flight tickets, accommodation bookings, ride-sharing, events.

To run a business, it's important to be able to **forecast customer demand** so that you can prepare a budget, stock inventory, allocate resources, and update pricing strategy. For example, if you run a grocery store, you want to stock enough so that customers find what they're looking for, but you don't want to overstock, because if you do, your groceries might go bad and you lose money.

Acquiring a new user is expensive. As of 2019, the average cost for an app to acquire a user who'll make an in-app purchase is $86.61[9]. The acquisition cost for Lyft is estimated at $158/rider[10]. This cost is so much higher for enterprise customers. Customer acquisition cost is hailed by investors as a startup killer[11]. **Reducing customer acquisition costs** by a small amount can result in a large increase in profit. This can be done through better identifying potential customers, showing better-targeted ads, giving discounts at the right time, etc.—all of which are suitable tasks for ML.

After you've spent so much money acquiring a customer, it'd be a shame if they leave. The cost of acquiring a new user is approximated to be 5 to 25 times more expensive than retaining an existing one. **Churn prediction** is predicting when a specific customer is about to stop using your products or services so that you can take appropriate actions to win them back. Churn prediction can be used not only for customers but also for employees.

To prevent customers from leaving, it's important to keep them happy by addressing their concerns as soon as they arise. **Automated support ticket classification** can help with that. Previously, when a customer opens a support ticket or sends an email, it needs to first be processed then passed around to different departments until it arrives at the inbox of someone who can address it. An ML system can analyze the ticket content and predict where it should go, which can shorten the response time and improve customer satisfaction. It can also be used to classify internal IT tickets.

Another popular use case of ML in enterprise is **brand monitoring**. The brand is a valuable asset of a business[12]. It's important to monitor how the public and how your customers perceive your brand. You might want to know when/where/how it's mentioned, both explicitly (e.g. when someone mentions "Google") or implicitly (e.g. when someone says "the search giant") as well as the sentiment associated with it. If there's suddenly a surge of negative sentiment in your brand mentions, you might want to do something about it as soon as possible. Sentiment analysis is a typical ML task.

A set of ML use cases that has generated much excitement recently is in health care. There are ML systems that can **detect skin cancer** and **diagnose diabetes**. Even though many healthcare applications are geared towards consumers, because of their strict requirements with accuracy and privacy, they are usually provided through a healthcare provider such as a hospital or used to assist doctors in providing diagnosis.

---

[9] Average mobile app user acquisition costs worldwide from September 2018 to August 2019, by user action and operating system (Statista, 2019)
[10] Valuing Lyft Requires A Deep Look Into Unit Economics (Forbes, 2019)
[11] Startup Killer: the Cost of Customer Acquisition (David Skok, 2018)
[12] Apple, Google, Microsoft, Amazon each has a brand estimated to be worth in the order of hundreds of millions dollars (Forbes, 2020)

Understanding Machine Learning Systems

Understanding ML systems will be helpful in designing and developing them. In this section, we'll start with the question: how important is data for building intelligent systems? We'll then go over how ML systems are different from both ML in research (or as often taught in school) and traditional software, which motivates the need for this book.

Mind vs. Data

Progress in the last decade shows that the success of an ML system depends largely on the data it was trained on. Instead of focusing on improving ML algorithms, most companies focus on managing and improving their data.

Despite the success of models using massive amounts of data, many are skeptical of the emphasis on data as the way forward. In the last three years, at every academic conference I attended, there were always some debates among famous academics on the power of mind vs. data. *Mind* might be disguised as inductive biases or intelligent architectural designs. *Data* might be grouped together with computation since more data tends to require more computation.

In theory, you can both pursue intelligent design and leverage large data and computation, but spending time on one often takes time away from another.

On the mind over data camp, there's Dr. Judea Pearl, a Turing Award winner best known for his work on causal inference and Bayesian networks. The introduction to his book, "The book of why", is entitled "Mind over data," in which he emphasizes: "*Data is profoundly dumb*." In one of his more controversial posts on Twitter, he expressed his strong opinion against ML approaches that rely heavily on data and warned that data-centric ML people might be out of job in 3-5 years.

[QUOTE]

> *"ML will not be the same in 3-5 years, and ML folks who continue to follow the current data-centric paradigm will find themselves outdated, if not jobless. Take note."*[13]

[/QUOTE]

There's also a milder opinion from Professor Christopher Manning, Director of the Stanford Artificial Intelligence Laboratory, who argued that huge computation and a massive amount of data with a simple learning algorithm create incredibly bad learners. The structure allows us to design systems that can learn more from fewer data[14].

---

[13] Tweet by Dr. Judea Pearl (2020)

[14] Deep Learning and Innate Priors (Chris Manning vs. Yann LeCun debate).

Many people in ML today are on the data over mind camp. Professor Richard Sutton, a professor of computing science at the University of Alberta and a distinguished research scientist at DeepMind, wrote a great blog post in which he claimed that researchers who chose to pursue intelligent designs over methods that leverage computation will eventually learn a bitter lesson.

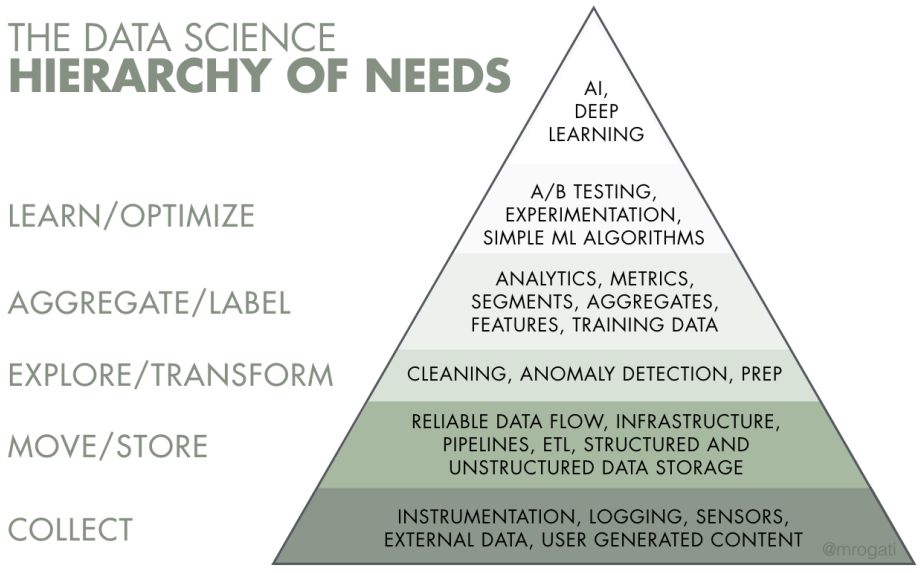
[QUOTE]

"The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. ... Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation."[15]

[/QUOTE]

When asked how Google search was doing so well, Peter Norvig, Google's Director of Search, emphasized the importance of having a large amount of data over intelligent algorithms in their success: "*We don't have better algorithms. We just have more data.*"[16]

Dr. Monica Rogati, Former VP of Data at Jawbone, argued that data lies at the foundation of data science, as shown in Figure 1-3. If you want to use data science, a discipline of which machine learning is a part of, to improve your products or processes, you need to start with building out your data, both in terms of quality and quantity. Without data, there's no data science.



[15] The Bitter Lesson (Richard Sutton, 2019)
[16] The Unreasonable Effectiveness of Data (Alon Halevy, Peter Norvig, and Fernando Pereira, Google 2009)

Figure 1-3: The data science hierarchy of needs (Monica Rogati, 2017[17])

The debate isn't about whether *finite* data is necessary, but whether it's sufficient. The term *finite* here is important, because if we had infinite data, we can just look up the answer. Having a lot of data is different from having infinite data.

Regardless of which camp will prove to be right eventually, no one can deny that data is essential, for now. Both the research and industry trends in the recent decades show the success of machine learning relies more and more on the quality and quantity of data. Models are getting bigger and using more data. Back in 2013, people were getting excited when the One Billion Words Benchmark for Language Modeling was released, which contains 0.8 billion tokens[18]. Six years later, OpenAI's GPT-2 used a dataset of 10 billion tokens. And another year later, GPT-3 used 500 billion tokens. The growth rate of the sizes of datasets is shown in Figure 1-4.

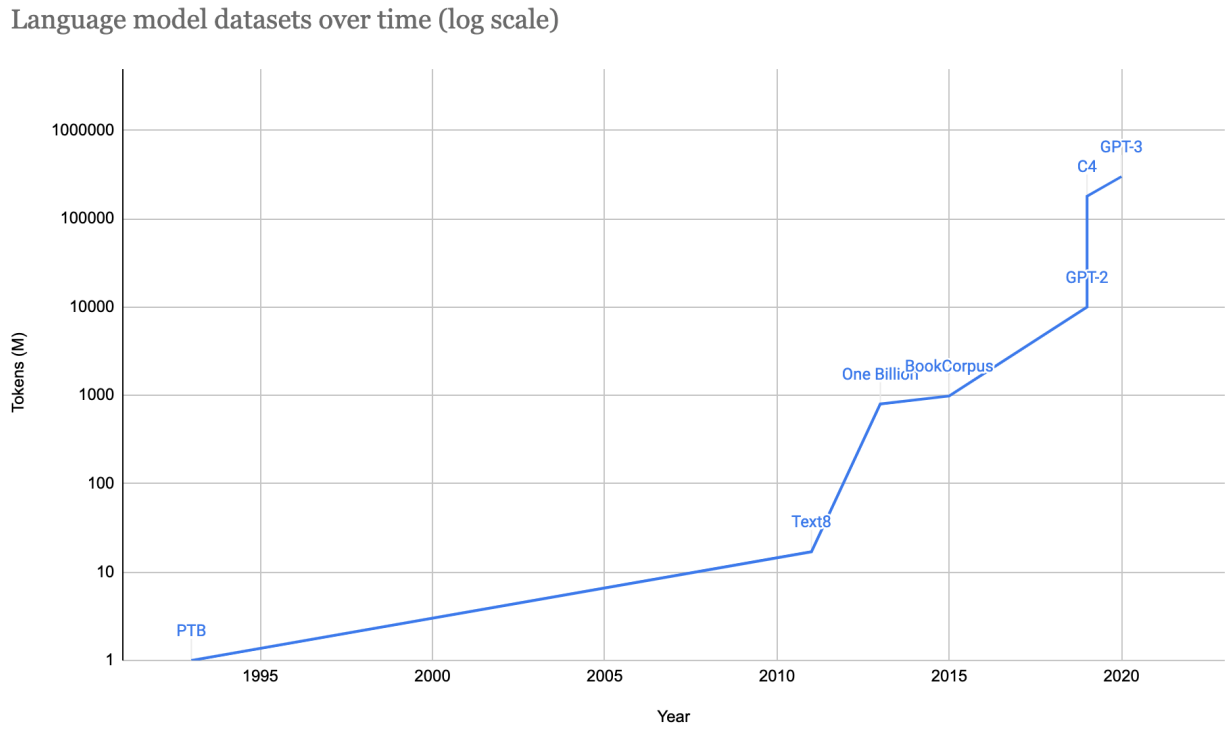


Figure 1-4: The size of the datasets used for language models over time (log scale)

Even though much of the progress in deep learning in the last decade was fueled by an increasingly large amount of data, more data doesn't always lead to better performance for your model. More data at lower quality, such as data that is outdated or data with incorrect labels, might even hurt your model's performance.

[17] The AI Hierarchy of Needs (Monica Rogati, 2017)
[18] 1 Billion Word Language Model Benchmark (Chelba et al., 2013)

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 academic papers. If that describes your background, it might be a steep learning curve for you to understand the challenges of deploying ML systems in the wild and navigate an overwhelming set of solutions to these challenges. ML in production is very different from ML in research. Table 1-1 shows 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[19]	Constantly shifting
Fairness	Good to have (sadly)	Important
Interpretability	Good to have	Important

Table 1-1: Key differences between ML in research and ML in production.

Stakeholders and their objectives

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.

[19] A subfield of research focuses on continual learning: developing models to work with changing data distributions. We'll cover continual learning in Chapter 7.

- 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[20] 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[21].

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*.[22]" 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 savings. For example, a 0.2% improvement in the click-through-rate for a product recommendation system can result in millions of dollars increase in revenue for an ecommerce site. 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 bar: 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[23].

---

[20] Inference here means "generate predictions".
[21] 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).
[22] Ensemble learning (Wikipedia)
[23] 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[24].

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[25].

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, 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.

[WARNING]
Terminology clash
Some books make the distinction between latency and response time. According to Martin Kleppmann in his foundational book *Designing Data-Intensive Applications*, "*the response time is what the client sees: besides the actual time to process the request (the service time), it includes network delays and queueing delays. Latency is the duration that a request is waiting to be handled — during which it is latent, awaiting service.*"

[24] AI competitions don't produce useful models (Luke Oakden-Rayner, 2019)
[25] Utility is in the Eye of the User: A Critique of NLP Leaderboards (Ethayarajh and Jurafsky, EMNLP 2020)

In this book, to simplify the discussion and to be consistent with the terminology used in the ML community, we use latency to refer to the response time, so the latency of a request measures the time from when the request is sent to the time a response is received.
[/WARNING]

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, because most modern distributed systems batch queries to process them together, often concurrently, higher latency might also mean higher throughput. If you process 10 queries at a time and it takes 10ms to run a batch, the 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 2017, an Akamai study found that 100ms delay can hurt conversion rates by 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*."[26] In 2016, Google found that more than half of mobile users will leave a page if it takes more than 3 seconds to load. Users today are even less patient.

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.

[26] 150 Successful Machine Learning Models: 6 Lessons Learned at Booking.com (Bernardi et al., KDD 2019)

When thinking about latency it's important to keep in mind that it's not just an individual number but a distribution. It's tempting to simplify a distribution by using a single number like the average (arithmetic mean) latency of all the requests within a time window, but this number can be misleading. Imagine you have 10 requests whose latency numbers are 100ms, 102ms, 100ms, 100ms, 99ms, 104ms, 110ms, 90ms, 3000ms, 95ms. The average latency of these 10 requests is 390ms, which makes your system seem slower than it actually is. What might happen is that there was a network error that made one request much slower, and you should investigate that troublesome request.

It's usually better to think in percentiles as they tell you about the percentages of your users who are experiencing something. The most common percentile is the 50th percentile, abbreviated as p50. It's also known as median. If the median is 100ms, half of the requests take longer than 100ms, and half of the requests take less than 100ms.

Higher percentiles also help you discover outliers, which might be symptoms of something wrong. Typically, the percentiles you'll want to look at are p90, p95, and p99. The 90th percentile (p90) for the 10 requests above is 3000ms, which is an outlier.

Higher percentiles are important to look at because even though they account for a small percentage of your users, sometimes they can be the most important users. For example, on Amazon website, the customers with the slowest requests are often those who have the most data on their accounts because they have made many purchases — that is, they're the most valuable customers[27].

It's a common practice to use high percentiles to specify the performance requirements for your system, for example, a product manager might specify that the 90th percentile or 99.9th percentile latency of a system must be below a certain number.

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

[27] Designing Data-Intensive Applications (Martin Kleppmann, O'Reilly 2017)

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.

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.

Figure 1-5 is a great graphic by Andrej Karpathy, head of AI at Tesla, that illustrates the data problems he encountered during his PhD compared to his time at Tesla.

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

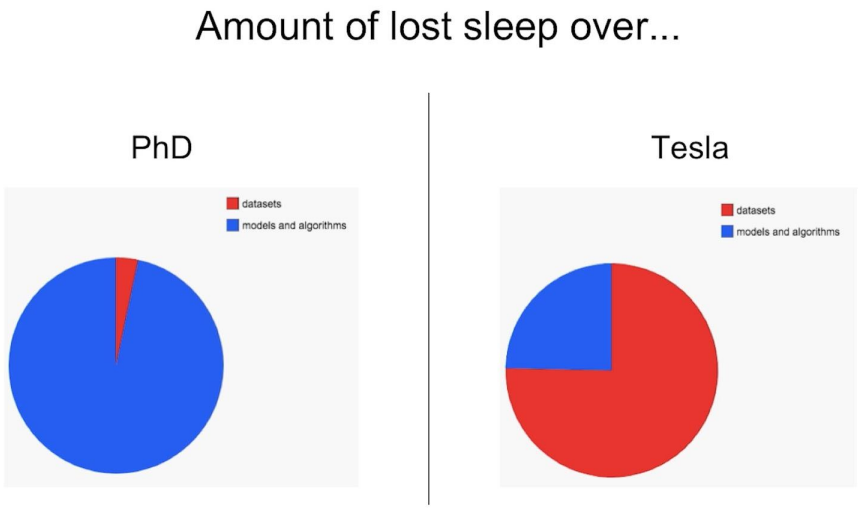


Figure 1-5: Data in research vs. data in production by Andrej Karpathy[28]

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. On top of that, as of 2021,

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

fairness isn't yet a metric for researchers to optimize on. If you optimize your models for better accuracy or lower latency, you can show that your models beat state-of-the-art. But there's no equivalent state-of-the-art for fairness metrics.

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, and college ranking.

In 2019, "*Berkeley researchers found that both face-to-face and online lenders rejected a total of 1.3 million creditworthy black and Latino applicants between 2008 and 2015*." When the researchers "*used the income and credit scores of the rejected applications but deleted the race identifiers, the mortgage application was accepted"*[29]. For even more galling examples, I recommend Cathy O'Neil's *Weapons of Math Destruction*[30].

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[31].

Interpretability

In early 2020, the Turing Award winner Professor Geoffrey Hinton proposed a heatedly debated question about the importance of interpretability in ML systems.

[QUOTE]

---

[29] Mortgage discrimination: Black and Latino paying millions more in interest, study shows (CBS News, 2019)
[30] Weapon of Math Destruction (Cathy O'Neil, Crown Books 2016)
[31] AI Index 2019 (Stanford HAI, 2019)

"*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?*"[32]

[/QUOTE]

A couple of weeks later, when I asked this question 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.

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.

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[33].

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. Oftentimes, new models require 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.

---

[32] https://twitter.com/geoffreyhinton/status/1230592238490615816

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

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

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.

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).

On the contrary, ML systems are part code, part data, and part artifacts created from the two. 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 (see Figure 1-6).

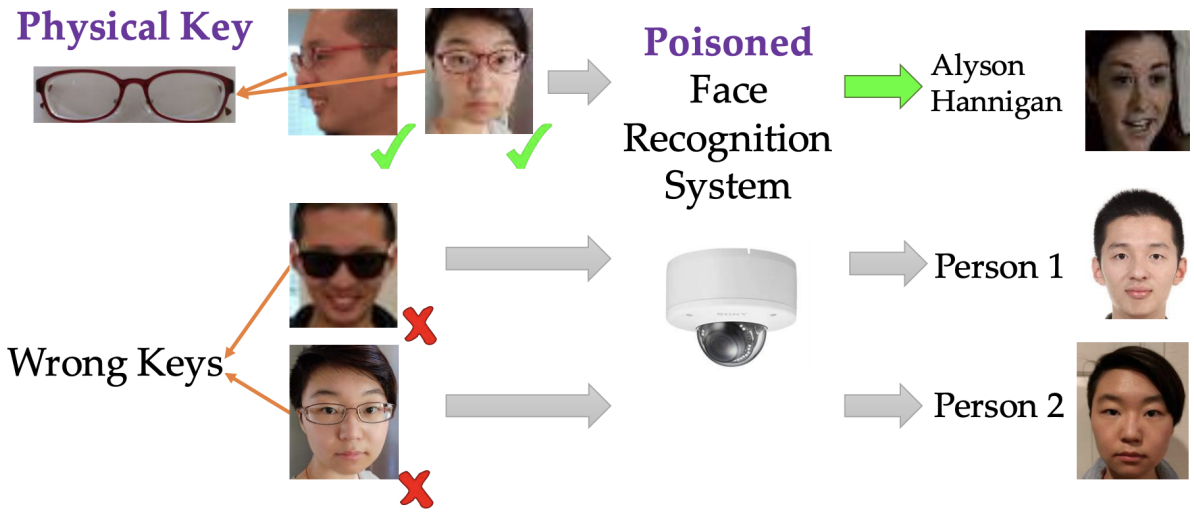


Figure 1-6: An example of how a face recognition system can be poisoned, using malicious data, to allow unauthorized people to pose as someone else.
Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning
(Chen et al., 2017)

The size of ML models gives another challenge. As of 2021, it's common for ML models to have hundreds of billions, if not trillions, 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[34], 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 the BERT (Bidirectional Encoder Representations from Transformers) paper 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[35]. Fast forward

[34] We'll cover edge devices in Chapter 6. Deployment.
[35] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., 2018)

two years later, BERT and its variants were already used in almost every English search on Google[36].

Designing ML Systems in Production

Now that we've discussed what it takes to develop and deploy an ML system, let's get to the fun part of actually designing one. This section aims to give you an overview of machine learning systems design. It starts by explaining what machine learning systems design is and covers the requirements for ML systems. We will then go over the iterative process for designing systems to meet those requirements.

ML systems design is the process of defining all the components of an ML system, including **interface, algorithms, data**, **infrastructure**, and **hardware**, so that the system satisfies **specified requirements**.

Requirements for ML Systems

Before building a system, it's essential to define requirements for that system. Requirements vary from use case to use case. However, most systems should have these four characteristics: reliable, scalable, maintainable, and adaptable.

We'll walk through each of these concepts in detail. Let's take a closer look at reliability first.

Reliability

The system should continue to perform the **correct function** at the **desired level of performance** even in the face of **adversity** (hardware or software faults, and even human error).

"Correctness" might be difficult to determine for ML systems. For example, your system might call the function ".predict()" correctly, but the predictions are wrong. How do we know if a prediction is wrong if we don't have ground truth labels to compare it with?

With traditional software systems, you often get a warning, such as a system crash or runtime error or 404. However, ML systems fail silently. End users don't even know that the system has failed and might have kept on using it as if it was working. For example, if you use Google Translate to translate a sentence into a language you don't know, it might be very hard for you to tell even if the translation is wrong.

[36] Google SearchOn 2020.

Scalability

As the system grows (in data volume, traffic volume, or complexity), there should be reasonable ways of dealing with that growth.

Scaling isn't just up-scaling[37] — expanding the resources to handle growth. In ML, it's also important to down-scale — reducing the resources when not needed. For example, at peak, your system might require 100 GPUs. However, most of the time, your system needs only 10 GPUs. Keeping 100 GPUs up all the time can be costly, so your system should be able to scale down to 10 GPUs.

An indispensable feature in many cloud services is autoscaling: automatically scaling up and down the number of machines depending on usage. This feature can be tricky to implement. Even Amazon fell victim to this when their autoscaling feature failed on Prime Day, causing their system to crash. An hour downtime was estimated to cost it between $72 million and $99 million[38].

Maintainability

There are many people who will work on an ML system. They are ML engineers, DevOps engineers, and subject matter experts (SMEs). They might come from very different backgrounds, with very different languages and tools, and might own different parts of the process. It's important to structure your project and set up your infrastructure in a way such that different contributors can work using tools that they are comfortable with, instead of one group of contributors forcing their tools onto other groups. When a problem occurs, different contributors should be able to work together to identify the problem and implement a solution without finger-pointing. We'll go more into this in chapter 7.

Adaptability

To adapt to changing data distributions and business requirements, the system should have some capacity for both discovering aspects for performance improvement and allowing updates without service interruption.

Because ML systems are part code, part data, and data can change quickly, ML systems need to be able to evolve quickly. This is tightly linked to maintainability. We'll go more into this in chapter 7.

---

[37] Up-scaling and down-scaling are two aspects of "scaling out", which is different from "scaling up". Scaling out is adding more equivalently functional components in parallel to spread out a load. Scaling up is making a component larger or faster to handle a greater load.

[38] Wolfe, Sean. 2018. "Amazon's one hour of downtime on Prime Day may have cost it up to $100 million in lost sales." Business Insider. https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7.

Iterative Process

Developing an ML system is an iterative and, in most cases, never ending process[39]. You do reach the point where you have to put the system into production, but then that system will constantly need to be monitored and updated.

Before deploying my first ML system, I thought the process would be linear and straightforward. I thought all I had to do was to collect data, train a model, deploy that model, and be done. However, I soon realized that the process looks more like a cycle with a lot of back and forth between different steps.

For example, here is one workflow that you might encounter when building an ML model to predict whether an ad should be shown when users enter a search query[40].

1. Choose a metric to optimize. For example, you might want to optimize for impressions -- the number of times an ad is shown.
2. Collect data and obtain labels.
3. Engineer features.
4. Train models.
5. During error analysis, you realize that errors are caused by wrong labels, so you relabel data.
6. Train model again.
7. During error analysis, you realize that your model always predicts that an ad shouldn't be shown, and the reason is because 99.99% of the data you have is no-show (an ad shouldn't be shown for most queries). So you have to collect more data of ads that should be shown.
8. Train model again.
9. Model performs well on your existing test data, which is by now two months ago. But it performs poorly on the test data from yesterday. Your model has degraded, so you need to collect more recent data.
10. Train model again.
11. Deploy model.
12. Model seems to be performing well but then the business people come knocking on your door asking why the revenue is decreasing. It turns out the ads are being shown but few people click on them. So you want to change your model to optimize for clickthrough rate instead.
13. Go to step 1.

[39] Which, as an early reviewer pointed out, is a property of traditional software.
[40] Praying and crying not featured but present through the entire process.

Figure 1-7 shows an oversimplified representation of what the iterative process for developing ML systems in production looks like.

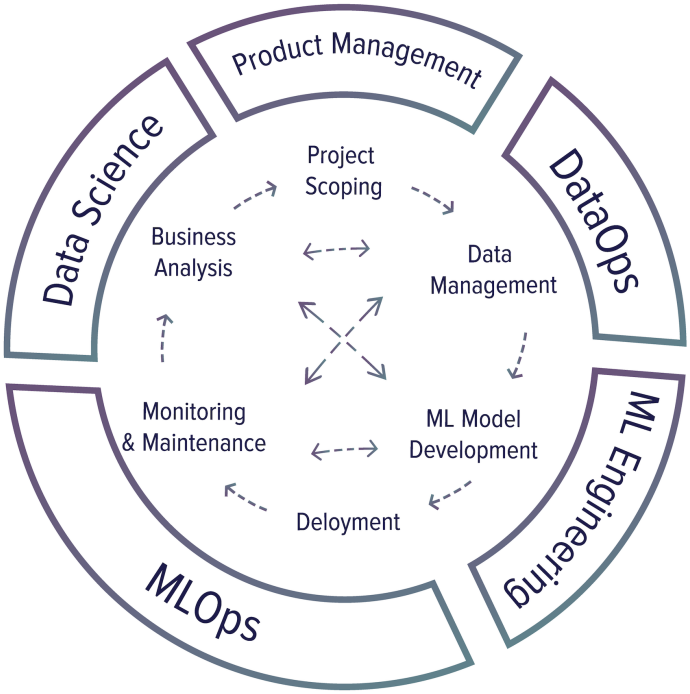


Figure 1-7: The process of developing an ML system looks more like a cycle with a lot of back and forth between steps.

While we'll take a deeper dive into what each of these steps mean in practice in later chapters, let's take a brief look at what happens during each of the steps.

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 focuses for ML projects in production earlier in this chapter. We'll discuss how to scope an ML project in the context of a business as well as how to organize teams to ensure the success of an ML project in Chapter 9: 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 2, we'll discuss the importance of data in ML and 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.

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 4, we'll discuss feature engineering and in chapter 5, 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 6.

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 chapter 7.

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. Because this step is closely related to the first step, it will also be discussed in chapter 9.

Summary

This opening chapter aims to give readers an understanding of what it takes to bring ML into the real world. ML systems are complex, consisting of many different components. Data scientists and ML engineers working with ML systems in production will likely find that focusing only on the ML algorithms part isn't enough. It's important to know about other aspects of the system, including data engineering, online vs. batch prediction, deployment, monitoring, maintenance, etc. This book aims to cover the other aspects of the system instead of just ML algorithms.

We started with a tour of the wide range of use cases of ML in production today. While most people are familiar with ML in consumer facing applications, the majority of ML use cases are for enterprise. We also discussed when ML would be appropriate. Even though ML can solve many problems very well, it can't solve all the problems and it's certainly not appropriate for all the problems. However, for problems that ML can't solve, it's possible that ML can solve part of them.

We continued to discuss a high-level debate that has consumed much of the ML literature: which is more important — data or intelligent algorithms. There are still many people who believe that having intelligent algorithms will eventually trump having a large amount of data. However, the success of systems including AlexNet, BERT, GPT showed that the progress of ML in the last decade relies on having access to a large amount of data. Regardless of whether data can overpower intelligent design, no one can deny the importance of data in ML. A nontrivial part of this book will be devoted to shedding light on various data questions.

This chapter highlighted the differences between ML in research and ML in production. The differences include the stakeholders involved, computational priority, the properties of data used, the gravity of fairness issues, and the requirements for interpretability. This section is the most helpful to those coming to ML production from academia. We also discussed how ML systems differ from traditional software systems, which motivated the need for this book.

Fortunately, complex ML systems are made up of simpler building blocks. Now that we've covered the high-level overview of an ML system in production, we'll zoom into its building blocks in the following chapters, starting with the fundamentals of data engineering in the next chapter. If any of the challenges mentioned in this chapter seems abstract to you, I hope that specific examples in the following chapters will make them more concrete.