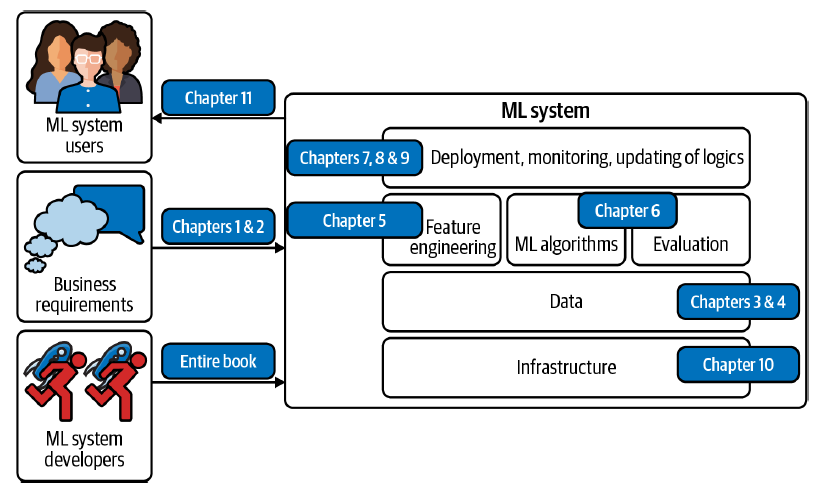
# Designing Machine Learning Systems - Chip Huyen

## Chapter 1 – Overview of ML Systems

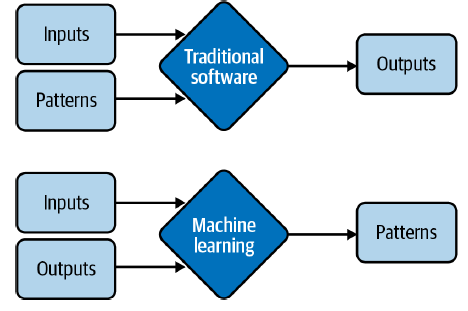
* Many when they hear “ML system” think of just the ML *algorithms* being used, such as logistic regression or different types of NN’s
* However, **the algorithm is only a small part of an ML system in production**
* ML systems also include the **business requirements** that gave birth to the ML project in the first place, the **interface** where users and developers interact with your system, the **data stack**, and the **logic** for developing, monitoring, + updating your models, as well as the **infrastructure** that enables the delivery of that logic



* Algorithms might become outdated quickly as new algorithms are constantly being developed, but the framework proposed in this book should still work with new algorithms
* “Ops” in **MLOps** comes from **DevOps**, short for **Developments and Operations**
* To **operationalize** something means to **bring it into production**, which includes **deploying, monitoring, and maintaining** it
* **MLOps** = a set of tools and best practices for bringing ML into production
* **ML systems design takes a system approach to MLOps**, which means that it **considers an ML system holistically** to ensure that all the components + their stakeholders can work together to satisfy the specified objectives and requirements

### When to Use Machine Learning

* ML has proven to be a powerful tool for a wide range of problems
* **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 or cost-effective**
* **“Is ML is *sufficient?*” 🡪 the answer is always NO**
* Examine what ML solutions generally *do*: ML is an **approach to** (1) **learn** (2) **complex patterns** from (3) **existing data**and *use* these patterns to make (4) **predictions**on (5) **unseen data**
* **1) Learn: the system has the capacity to learn**
* A relational database (RDB) *isn’t* an ML system because it doesn’t have the capacity to *learn*
* Can explicitly state the relationship between 2 columns in an RDB, but it’s **unlikely to have the capacity to figure out the relationship between these 2 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 example input and output pairs, ML systems learn how to generate outputs for arbitrary inputs
* Ex: 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 relevant 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 should be able to predict the price of a new listing given its characteristics
* **2) Complex patterns: there are patterns to learn, and they are complex**
* **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
* **Patterns** are *different* from **distributions** 🡪 We know the distribution of the outcomes of a fair die, but *there are no patterns in the way outcomes are generated*
* 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 or ML algorithms might not be sufficient to capture them**
* Ex: There might be a pattern in how Elon Musk’s tweets affect crypto prices
* However, you wouldn’t know until you’ve rigorously trained and evaluated ML models on his tweets
* **And, even if *all* your models fail to make reasonable predictions of crypto prices, it *doesn’t* mean there’s no pattern.**
* Consider a website like Airbnb with a lot of house listings; each listing comes with a ZIP
* 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 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 manually specify**, + ML is a good solution for this
* Instead of telling your system how to calculate the price from a list of characteristics, **provide the prices and characteristics, and let your ML system figure out the pattern**
* The **difference between ML solutions and the lookup table solution, as well as general traditional software solutions** is shown in below



* Instead of requiring hand-specified patterns to calculate outputs, **ML solutions learn patterns from inputs and outputs**
* For this reason, ML is also called **Software 2.0**.5
* 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 to humans**
* Many tasks that are hard for humans to do are easy for machines (Ex: raising a number of the power of 10)
* On the other hand, **many tasks easy for humans can be hard for machines** (Ex: deciding whether there’s a cat in a picture
* **3) 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**
* In the **zero-shot learning** (or **zero-data learning**) context, it’s **possible for an ML system to make good predictions for a task without having been trained on data for a specific task**
* *However*, this ML system was previously trained on data for *other* tasks, *often related to the task in consideration*
* 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**
* Ex: In the context of **continual learning**, **ML models can be deployed without having been trained on any data**, but they **will learn from incoming data in production**
* However, **serving insufficiently trained models to users comes with certain risks, such as poor customer experience**
* Without data and without continual learning, many companies follow a “fake-it til- you make it” approach: launching a product that serves predictions made by *humans*, instead of ML models, with the hope of using the generated data to train ML models *later*
* **4) Predictions**: **it’s a predictive problem**
* **ML models make predictions**, so they **can only solve problems that require predictive answers**
* ML can be **especially appealing** when you can **benefit** from a **large quantity of cheap but approximate predictions**
* 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 an *exact* outcome of a process (which might be even more computationally costly + time-consuming than ML), you can frame a problem as: “What would the outcome of this process look like?” and approximate it using an ML model
* **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 and screen-space shading
* **5) Unseen data**: **unseen data shares patterns with the training data**
* The **patterns a 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 have a model that performs poorly, which we might be able to find out with **monitoring** and **test** in production
* Due to the way most ML algorithms today learn, **ML solutions will especially shine if your problem has these additional following characteristics**
* **6. It’s repetitive**
* Humans = great at **few-shot learning**: show kids a few pictures of cats + most will recognize a cat the next time they see one
* Despite exciting progress in 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**
* **7. The cost of wrong predictions is cheap**
* Unless your ML model’s performance is 100% all the time (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**
* Ex: 1 of the biggest use cases of ML today = recommender systems, because a bad recommendation is usually forgiving (the user just won’t click on the recommendation)
* **If 1 prediction mistake can have catastrophic consequences, ML *might* still be a suitable solution if, *on average*, the benefits of correct predictions outweigh the cost of wrong predictions**
* Developing self-driving cars is challenging because an algorithmic mistake can lead to death
* However, many companies still want to develop self-driving cars because they have the potential to save many lives once self-driving cars are statistically safer than human drivers
* **8. It’s at scale**
* **ML solutions often require nontrivial up-front 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, **in general**, it **means making a lot of predictions**
* Ex: Sorting through millions of emails a year, or predicting which departments thousands of support tickets should be routed to a day
* **A problem might appear to be a singular prediction, but it’s actually a *series* of predictions**
* Ex: A model that predicts who will win a US presidential election seems like it only makes 1 prediction every 4 years, but it might actually be making a prediction every hour or even more frequently because that **prediction has to be continually updated to incorporate new information**
* **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**
* **9. The patterns are constantly changing**
* If your problem involves 1 or more constantly changing patterns, **hardcoded solutions such as handwritten rules can become outdated quickly**
* **Figuring how your problem has changed so that you can update your handwritten 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**, like in a **continual learning approach**
* 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 of today’s ML algorithms shouldn’t be used under any of the following conditions:
* It’s **unethical**
* **Simpler solutions do the trick** (4 phases of ML model development = 1st phase should be non-ML solutions)
* 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 use ML to solve some of them**
* Ex: 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 FAQ’s
* If yes, 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 existing technologies at the moment**
* **Most technological advances are incremental 🡪** A type of technology might not be efficient now, but it might be over time with more investments
* **If you wait for the technology to prove its worth to the rest of the industry before jumping in, you might end up years or decades behind 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 consumers.

With the explosion of information and services, it would have been very challenging

for us to find what we want without the help of ML, manifested in either a *search*

*engine* or a *recommender 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 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 opening something else, but enterprise users might be more tolerant

of high latency. For people interested in building companies out of ML applications, consumer apps might be easier to distribute but much harder to monetize. 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)

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 opened a support ticket or sent an email,

it needed to first be processed then passed around to different departments until it

arrived at the inbox of someone who could 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.13 It’s important to monitor how the public and 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 address 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 health-care applications are geared toward consumers, because of their strict

requirements with accuracy and privacy, they are usually provided through a healthcare

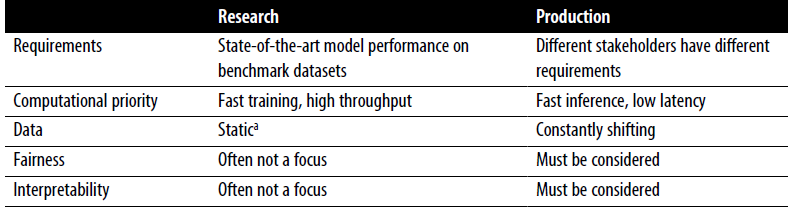
provider such as a hospital or used to assist doctors in providing diagnosis.

### Understanding Machine Learning Systems

* **Understanding ML systems will be helpful in designing and developing them**
* ML systems = different from both ML in research/as often taught in school and traditional software

#### Machine Learning in Research Versus in Production

* As ML usage in the industry is still fairly new, **most people with ML expertise have gained it through academia**: courses, research, reading academic papers.
* **ML in production is very different from ML in research: See 5 of the major differences**

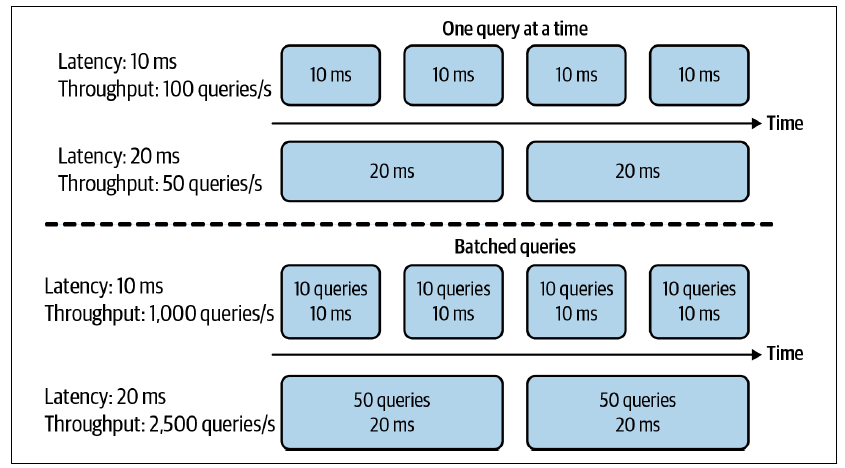


##### Different stakeholders and requirements

* **Research and leaderboard project members often align on 1 *single* objective**, the most common being **model performance** = **develop a model that achieves the state-of-the-art 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 with their **own requirements**
* Having **different, often conflicting, requirements can make it difficult to design, develop, and select an ML model that satisfies all the requirements**
* Consider a mobile app that recommends restaurants to users
* The app makes money by charging restaurants a 10% service fee on each order, which means expensive orders give the app more money than cheap orders
* The project involves ML engineers, salespeople, product managers, infrastructure engineers, and a manager:
* 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
* Sales team: Wants a model that recommends the more expensive restaurants since these restaurants bring in more service fees.
* Product team: Notices that every increase in latency leads to a drop in orders through the service, so they want a model that can return the recommended restaurants in < 100 ms.
* ML platform team: As traffic grows, this 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 on model updates to prioritize improving the ML platform
* Manager: Wants to maximize the margin, and one way to achieve this might be to let go of the ML team (It’s not unusual for the ML and DS teams to be among the first to go during a company’s mass layoff)
* “Recommending the restaurants that users are most likely to click on” and “recommending the restaurants that will bring in the most money for the app” are *two different objectives*
* **1 way to develop an ML system that satisfies different objectives: develop one model for each objective and combine their predictions**
* Imagine for now that we have 2 different models: Model A that recommends restaurants users are most likely to click on, and model B that recommends restaurants that will bring in the most money for the app
* A and B might be *very* different models. Which model should be deployed to the users?
* To make the decision more difficult, neither A nor B satisfies the requirement set forth by the product team: they can’t return restaurant recommendations in < 100 milliseconds.
* **When developing an ML project, it’s important for ML engineers to understand requirements from ALL stakeholders involved and *how strict these requirements are***.
* Ex: If being able to return recommendations within 100 ms. is a must-have requirement (the company finds that if your model takes > 100 ms. to recommend restaurants, 10% of users lose patience and close the app), then *neither* model A nor model B will work
* However, if it’s just a nice-to-have requirement, you might still want to consider model A or model B
* **Production having different requirements from research is one of the reasons why successful research projects might not always be used in production**
* Ex: **Ensembling is popular among the winners of many ML competitions, yet it’s not widely used in production.**
* **Ensembling combines multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone**
* While it **can give your ML system a small performance improvement**, ensembling **tends to make a system too complex to be useful in production** (e.g., **slower to make predictions or harder to interpret the results**)
* **For many tasks, a small improvement in performance can result in a huge boost in revenue or cost savings**
* EX: a 0.2% improvement in the click-through rate (CTR) for a product recommender 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**
* **For this type of task, if a simple model can do a reasonable job, complex models must perform *significantly* better to justify the complexity**

##### Computational priorities

* **When designing an ML system**, people who haven’t deployed an ML system often make the **mistake of focusing too much on the model development part + not enough on the model deployment + maintenance part.**
* During the model development process, you might train many different models, + each model does multiple passes over the training data
* Each trained model then generates predictions on the validation data once to report the scores.
* The validation data is usually much smaller than the training data
* **During model development, *training* is the bottleneck**
* **Once the model has been deployed**, however, its **job is to generate predictions**, so ***inference* is the bottleneck**
* **Research usually prioritizes fast training, whereas production usually prioritizes fast inference**
* 1 corollary of this is that **research prioritizes high throughput whereas production prioritizes low latency**
* **Latency** = the **time it takes from receiving a query to returning the result**
* **Throughput** = **how many queries are processed within a specific period of time**
* Some books make the distinction between latency and **response time**
* According to Kleppmann’s *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** + **queueing delays**. **Latency** is the **duration that a request is waiting to be handled**, during which it is latent, awaiting service”
* 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**
* Ex: Average latency of Google Translate is the average time it takes from when a user clicks Translate to when the translation is shown, + the throughput is how many queries it processes and serves a second
* **If your system always processes 1 query at a time, higher latency means lower throughput**
* If the average latency is 10 ms (takes 10 ms to process a query), throughput is 100 queries/second
* If the average latency is 100 ms, 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 + it takes 10 ms to run a *batch*, average latency is still 10 ms but throughput is now 10 times higher: 1,000 queries/second
* If you process 50 queries at a time and it takes 20 ms to run a batch, average latency now is 20 ms and throughput is 2,500 queries/second
* *Both latency and throughput have increased!*
* **The difference in latency and throughput trade-off for processing queries 1 at a time and processing queries in batches is illustrated below**



* This is **even more complicated if you want to batch online queries**
* **Batching requires your ML 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, for example, with aggressive batching**
* **However, once you deploy your model into the real world, latency matters a lot**
* Akamai study: a 100 ms delay can hurt conversion rates by 7%
* Booking.com found that an increase of about 30% in latency cost about 0.5% in conversion rates (“a relevant cost for our business”)
* 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**
* **To reduce latency in production, you might have to reduce the number of queries you can process on the same hardware at a time**
* *If your hardware is capable of processing many more queries at a time, using it to process fewer queries means underutilizing your hardware, increasing the cost of processing each query.*
* When thinking about latency, it’s important to **keep in mind that latency is NOT an individual number but a *distribution***
* It’s tempting to simplify this 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 due to outliers
* It’s **usually better to think in percentiles**, as they **tell you something about a certain percentage of your requests**
* The most common percentile is the 50th percentile, abbreviated as p50, also known as the median
* If the median is 100 ms, half of the requests take longer than 100 ms, and half of the requests take less than 100 ms.
* **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, + p99
* **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.**
* Ex: Amazon = customers with the slowest requests are often those who have the most data on their accounts because they have made many purchases (i.e., they’re the most valuable customers)
* It’s a **common practice to use high percentiles to specify the performance requirements for your system**
* Ex: A PM might specify that the 90th percentile or 99.9th percentile latency of a system must be below a certain number

##### Data

* **Research**, the **datasets you work with are often clean and well-formatted, freeing you to focus on developing 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**
* Might even find open-source scripts to process and feed the data directly into your models
* **Production** data, *if available*, is a lot **more messy**
* It’s **noisy**, **possibly unstructured**, **constantly shifting**
* It’s **likely biased**, and you likely **don’t know how it’s biased**
* **Labels**, *if there are any*, might be **sparse, imbalanced, or incorrect**
* **Changing project or business requirements might require updating some or all existing labels**.
* If you work with **user data**, you’ll also have to worry about **privacy and regulatory concerns**
* In **research**, 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**

##### Fairness

* **During research**, 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*
* If you optimize 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**
* Biased mathematical algorithms 🡪 loan application rejected based on ZIP (embodies biases about one’s socioeconomic background), resume might be ranked lower because the ranking system picks on the spelling of names, mortgage might get a higher interest rate because it relies partially on credit scores, which favor the rich and punish the poor
* Other examples of ML biases in the real world = predictive policing algorithms, personality tests administered by potential employers, and college rankings.
* **ML algorithms don’t predict the future, but encode the past, thus 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** (can especially hurt members of **minority groups** because **misclassification on them could only have a minor effect 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
* McKinsey 2019 research study: only 13% of large companies surveyed said they’re taking steps to mitigate risks to equity and fairness, such as algorithmic bias and discrimination

##### Interpretability

* 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 industry, but a *requirement***
* 1) Important for users, both business leaders + end users, to **understand why a decision is made so that they can trust a model and detect potential biases mentioned previously**
* 2) It’s **important for developers to be able to debug and improve a model.**
* *Just because interpretability is a requirement doesn’t mean everyone is doing it*
* 2019: only 19% of large companies are working to improve the explainability of their algorithms

##### Discussion

* Some might argue that it’s OK to know only the academic side of ML (true) because there are plenty of jobs in research (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.
* **The vast majority of ML-related jobs will be, and already are, in productionizing ML**

#### Machine Learning Systems Versus Traditional Software

* Since ML is part of 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**
* ML production would be a much better place if ML experts were better SWE’s
* **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**
* **SWE =** underlying assumption that **code and data are separated** (In fact, in SWE, we ***want* to keep things as modular and separate as possible** [**separation of concerns**])
* On the contrary, **ML systems are part code, part data, and part artifacts created from the two**
* Trend in the last decade 🡪 **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**
* **Traditional SWE** 🡪 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**
* Ex: If your model has already trained on 1M scans of normal lungs and only 1K 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 a model’s performance and even make it susceptible to data poisoning attacks**
* The **size of ML models is another challenge**
* 2022: common for ML models to have hundreds of millions, if not billions, of parameters, which requires gigabytes of RAM to load them into memory
* A few years from now, a billion parameters might seem quaint—like
* **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 one to type
* **Monitoring and debugging these models in production is also nontrivial**
* 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

### Summary

* Opening chapter aimed to give readers an understanding of what it takes to bring ML into the real world
* 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 solutions 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 ML can’t solve**, it’s possible that **ML can be one part of the solution**
* This chapter also highlighted the differences between ML in research and ML in production + the differences include the stakeholder involvement, computational priority, the properties of data used, the gravity of fairness issues, + 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
* ML systems are complex, consisting of many different components
* **Data scientists + MLE’s working with ML systems in production will likely find that focusing *only* on the ML algorithms part is *far from enough***
* **It’s important to know about other aspects of the system, including the data stack, deployment, monitoring, maintenance, infrastructure, etc.**