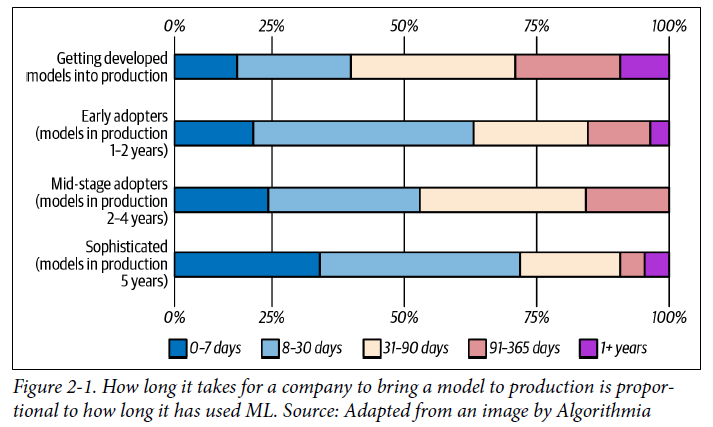
# Designing Machine Learning Systems - Chip Huyen

## Chapter 2 – Intro to ML System Design

* **ML systems design** takes a ***system* approach to MLOps 🡪 consider** an **ML system *holistically*** to **ensure** that **all components** (business requirements, data stack, infrastructure, deployment, monitoring, etc.) + their **stakeholders** can **work together** to **satisfy the specified objectives** **requirements**
* Before developing an ML system, must understand *WHY* this system is needed
* If this system is built for a business, it **must be driven by business objectives**, which will **need to be translated into ML objectives** to guide the development of ML models
* Once everyone is on board with the objectives for an ML system, **need to set requirements to guide the development** of this system
* **4 requirements**: **reliability**, **scalability**, **maintainability**, and **adaptability**.
* We then have the **iterative process for designing systems** to meet those requirements
* With all objectives, requirements, + processes in place, *before using ML algorithms to solve your problem*, you **first need to frame your problem into a task that ML can solve**
* *The difficulty of your job can change significantly depending on how you frame your problem*
* Because **ML = a data-driven approach**, there is a debate about **which is more important: data or intelligent algorithms?**

### Business and ML Objectives

* We **first need to consider the objectives of the proposed ML projects**
* **Data scientists tend to care about the ML objectives**: **metrics** **they can measure about performance** of their ML models (accuracy, F1 score, inference latency, etc.)
* Might spend a ton of resources (data, compute, engineering time) in improving their model’s accuracy from 94% to just 94.2%
* **Most companies don’t care about the fancy ML metrics** (don’t care about increasing model accuracy from 94% to 94.2% **unless it moves some *business* metrics**
* Data scientists can become too focused on hacking ML metrics without paying attention to business metrics
* *Managers, however, only care about business metrics +, after failing to see how an ML project can help push their business metrics, kill projects prematurely (+ possibly let go of the data science team involved)*
* While most companies want to convince you otherwise**, the sole purpose of most businesses is to maximize profits for shareholders** (Milton Friedman)
* The **ultimate goal of any project within a business is to increase profits, either directly** (increasing sales (conversion rates) + cutting costs) **or indirectly** (higher customer satisfaction, increasing time spent on a website)
* **For an ML project to succeed** *within a business organization*, it’s **crucial to tie the performance of an ML system to the overall business performance**
* *What business performance metrics is the new ML system supposed to influence? (e.g., Amount of ads revenue? Number of monthly active users?)*
* Ex: E-commerce site that cares about purchase-through rate (PTR) wants to move the recommender system from **batch prediction** to **online prediction**, since you reason that online prediction will enable recommendations more relevant to users right now, which can lead to a higher PTR
* You can even do an **experiment** to show that online prediction can improve your recommender system’s predictive accuracy by X%, and, historically on your site, each % increase in the recommender system’s predictive accuracy led to a certain increase in PTR
* 1 of the reasons why predicting ad click-through rates (CTR) + fraud detection are among the most popular use cases for ML today is that **it’s easy to map ML models’ performance to business metrics**: every increase in CTR results in *actual* ad revenue, + every fraudulent transaction stopped results in *actual* money saved.
* **Many companies create their own metrics to map business metrics to ML metrics**.
* Ex: Netflix measures performance of their recommender system using take-rate = number of quality plays divided by the number of recommendations a user sees
* The higher the take-rate, the better the recommender system
* Netflix *also* put a recommender system’s take-rate in the context of their *other* business metrics like total streaming hours and subscription cancellation rate
* They found that a higher take-rate also results in higher total streaming hours + lower subscription cancellation rates
* **The effect of an ML project on business objectives can be hard to reason about**
* Ex: An ML model that gives customers more personalized solutions can make them happier, which makes them spend more money on your services
* But the *same* ML model can also solve their problems faster, which makes them spend *less* money on your services
* **To gain a definite answer on the question of how ML metrics influence business metrics, experiments are often needed**
* **Many companies** do that with experiments like A/B testing + **choose the model that leads to better business metrics, *regardless of whether this model has better ML metrics***
* Yet, **even rigorous experiments might NOT be sufficient to understand the relationship between an ML model’s outputs and business metrics**
* Ex: A cybersecurity company that detects + stops security threats, + **ML is just a component in a complex process**
* ML model used to detect anomalies in traffic pattern, which then go through a logic set (e.g., a series of if-else statements) that categorizes whether they constitute potential threats
* These potential threats are then reviewed by security experts to determine whether they are actual threats
* Then, actual threats will then go through *another, different* process aimed at stopping them
* When this process fails to stop a threat, it **might be impossible to figure out whether the ML component has anything to do with it**
* **Many companies like to say they use ML in their systems because “being AI-powered” alone already helps attract customers, *regardless of whether the AI part actually does anything useful***
* **When evaluating ML solutions *through the business lens*, it’s important to be realistic about the expected returns**
* Due to all the hype surrounding ML, generated both by the media + by practitioners with a vested interest in ML adoption, some companies might have the notion that ML can magically transform their businesses overnight 🡪 ***Magically: possible. Overnight: no***
* Many companies have seen payoffs from ML
* ML helped Google search better, sell more ads at higher prices, improve translation quality, + build better Android applications), *but this gain hardly happened overnight*, asGoogle has been investing in ML for decades.
* **ROI in ML depends a lot on the *maturity stage* of adoption** 🡪 The **longer you’ve adopted ML**, the **more efficient your pipeline will run**, the **faster your development cycle** will be, the **less engineering time** you’ll need, + the **lower your cloud bills** will be, which **all lead to higher returns**



### Requirements for ML Systems

* We **can’t say we’ve successfully built an ML system without knowing *what requirements the system has to satisfy***
* Specified requirements for an ML system **vary from use case to use case**
* ***Most* systems should have these 4 characteristics: reliability, scalability, maintainability, and adaptability**

#### Reliability

* The ML system should **continue to perform the correct function at the desired level of performance even in the face of adversity** (hardware or software faults, even human error)
* **“Correctness” might be difficult to determine for ML systems**
* Ex: An ML system might call a predict function (model.predict()) correctly, but predictions are wrong
* *How do we know a prediction is wrong if we don’t have ground truth labels to compare with?*
* With traditional software systems, you often get a warning, such as a system crash or runtime error or 404
* However, **ML systems can fail silently**
* End users don’t even know the system has failed + might have kept on using it as if it were working
* Ex: Using Google Translate to translate a sentence into a language you don’t know, it might be very hard for you to tell if the translation is wrong

#### Scalability

* There are ***multiple* ways an ML system can grow**
* **1) Grow in complexity**
* Last year you used a logistic regression model that fit into an AWS free tier instance with 1 GB of RAM, but this year, you switched to a 100-millionparameter NN that requires 16 GB of RAM to generate predictions
* **2) Grow in traffic volume**
* When you started deploying your ML system, you only served 10K prediction requests daily
* However, as your company’s user base grows, the number of prediction requests your ML system serves daily fluctuates between 1 million and 10 million
* **3) Grow in ML model count**
* Initially, you might have only 1 model for 1 use case, such as detecting trending hashtags on Twitter
* However, over time, you want to add more features to this use case, so you’ll add 1 more model to filter out NSFW content + *another* model to filter out tweets generated by bots
* ***This growth pattern is especially common in ML systems that target enterprise use cases***
* Initially, a startup might serve only 1 enterprise customer, which means they only have 1 model
* However, as this startup gains more customers, they *might have 1 model for each customer* (ex: 8,000 models in production for their 8,000 enterprise customers)
* **Whichever way your system grows, there should be reasonable ways of dealing with that growth**
* When talking about **scalability** a majority of **people think of** **resource scaling**, which **consists of up-scaling (expanding the resources to handle growth)** and **down-scaling** (**reducing the resources when not needed**)
* Ex: At peak, a system might require 100 GPUs, however, most of the time, it 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**, which **can be tricky to implement**
* Ex: Amazon autoscaling feature failed on Prime Day, causing their system to crash for an hour, costing them between $72 million and $99 million.
* However, **handling growth isn’t *just* resource scaling, but also artifact management**
* Managing 100 hundred models is very different from managing 1 model
* With 1 model, you can, perhaps, *manually* monitor its performance, *manually* update it with new data, + you can just have a file that helps reproduce it whenever needed
* However, with 100 models, **both the monitoring + retraining aspect will need to be automated**, + you’ll need a way to manage code generation so that you can adequately reproduce a model when you need to

#### Maintainability

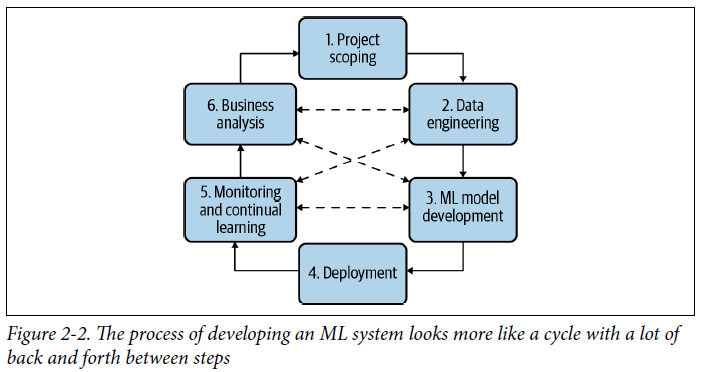
* Many people work on an ML system 🡪 MLE’s, DevOps engineers, + SMEs, **all of whom might come from very different backgrounds, with very different programming languages and tools, + might own different parts of the process.**
* It’s **important to structure workloads + set up infrastructure so that different contributor groups can work using tools they’re comfortable with**, instead of 1 group forcing tools onto other groups
* **Code should be documented. Code, data, and artifacts should be versioned**
* **Models should be sufficiently reproducible so that even when the original authors are not around, other contributors can have sufficient contexts to build on their work**
* When a **problem occurs**, different **contributors should be able to work together to identify the problem and implement a solution without finger-pointing**

#### Adaptability

* **To adapt to shifting data distributions + business requirements**, an ML system should **have some capacity for both discovering aspects for performance improvement + allowing updates without service interruption**
* Because **ML systems are part code, part data, + data can change quickly**, ML systems **need to be able to evolve quickly** 🡪 *This is tightly linked to maintainability*

### Iterative Process

* **Developing an ML system is an iterative and, in most cases, *never-ending* process** (also a property of traditional software)
* Once a system is **put into production**, it’ll need to be ***continually* monitored and updated**
* Example workflow for building an ML model to predict whether an ad should be shown when users enter a search query:
* **1. Choose a metric to optimize** (ex: optimize for impressions = 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 errors are caused by wrong labels, so you **relabel the data**
* **6. Train the model *again***
* **7.** During **error analysis**, you realize your model ***always* predicts that an ad *shouldn’t* be shown,** + the reason is because 99.99% of the data you have contains NEGATIVE labels (ads that shouldn’t be shown) 🡪 So you have to **collect more data of ads that *should* be shown**
* **8. Train the model again**
* **9. The model performs well** on your **existing test data**, which is by now 2 months old, but **performs poorly on the data from yesterday** 🡪 **model is now stale**, so you need to **update it on more recent data**
* **10. Train the model again**
* **11. Deploy the model**
* **12. The model seems to be performing well**, but then the business-people ask why **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 ad CTR instead**
* **13. Go to step 1**
* The figure below shows an oversimplified representation of what the iterative process for developing ML systems in production looks like from the perspective of a data scientist or an MLE



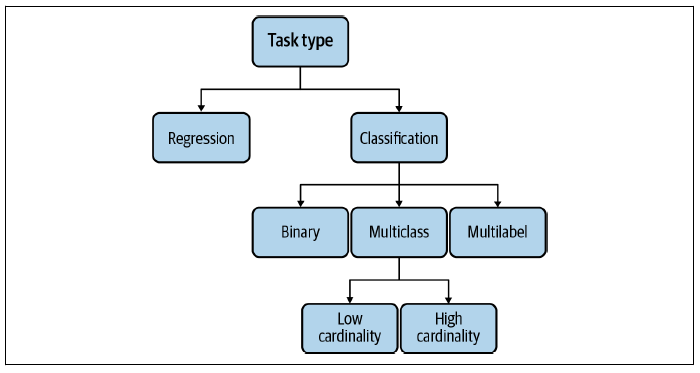
* *This process looks different from the perspective of an ML platform engineer or a DevOps engineer, as they might not have as much context into model development and might spend a lot more time on setting up infrastructure*
* Here’s a brief look at what these steps mean:
* **Step 1. Project scoping**
* A **project starts with scoping the project, laying out goals, objectives, + constraints**
* **Stakeholders** should be **identified + involved**, and **resources should be estimated + allocated**
* Different stakeholders = MLE’s, sales team, product team, ML platform team, manager
* **Step 2. Data engineering**
* A vast majority of **ML models today learn from *data***, so **developing ML models starts with *engineering some data***
* Fundamentals of data engineering 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 + develop initial models leveraging these features**
* ***This is the stage that requires the most ML knowledge + is most often covered in ML courses***
* **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**
* **Step 5. Monitoring and continual learning**
* Once in production, **models need to be monitored for performance decay + maintained to be adaptive to changing environments and changing requirements**
* **Step 6. Business analysis**
* **Model performance needs to be evaluated against *business* goals + analyzed to generate business insights** **(can be used to eliminate unproductive projects/scope out new projects)**
* *This step is closely related to the first step*

### Framing ML Problems

* Ex: You’re an MLE tech lead at a bank that targets millennial users.
* Your boss hears about a rival bank that uses ML to speed up customer service support that supposedly helps the rival bank process their customer requests 2X faster
* He orders your team to look into using ML to speed up your customer service support too
* *Slow customer support is a problem,* ***but it’s not an ML problem***
* **An ML problem is defined by inputs, outputs, + the objective function that guides the learning process** (none of these 3 components are obvious from your boss’s request)
* **It’s *your* job, as a seasoned MLE, to use your knowledge of what problems ML *can* solve to frame this request as an ML problem**
* Upon investigation, you discover the bottleneck in responding to customer requests lies in routing customer requests to the right out of 4 departments: accounting, inventory, HR, + IT
* You can alleviate this bottleneck by **developing an ML model to predict which of these 4 departments a request should go to**
* This makes it a **classification problem** = ***input*** *is the* ***customer request****,* ***output*** *is the* ***department*** *the request should go to, +* ***objective function = minimize the difference between the predicted department + the actual department***

#### Types of ML Tasks

* **The output of your model dictates the task type of your ML problem**
* **Most general types of ML tasks = classification and regression**
* Within classification, there are more subtypes seen below



##### Classification versus regression

* **Classification models classify inputs into different categories** (classify email as spam or not spam)
* **Regression models output a continuous value** (model to predict/output the price of a given house)
* **A regression model can easily be framed as a classification model and vice versa**
* Ex: House prediction can become a classification task if we quantize house prices into buckets such as < $100,000, $100,000–$200,000, + so forth + predict the bucket the house should be in
* Ex: Email classification model can become a regression model if we make it output values between 0 and 1, + decide on a threshold to determine which values should be SPAM (say, if the value is > 0.5, the email is spam)

##### Binary versus multiclass classification

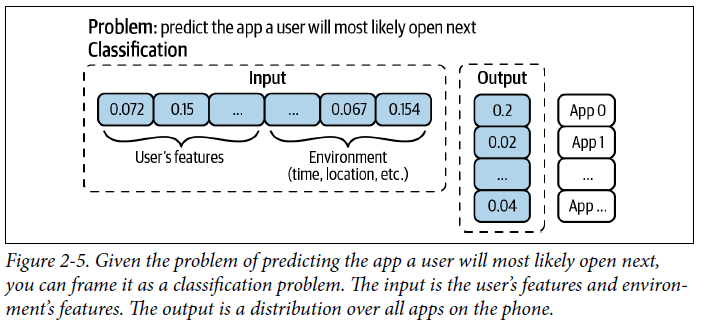
* **Within classification problems, the fewer classes there are to classify, the simpler the problem is**
* Simplest = **binary classification** w/ only 2 possible classes (classifying a comment as toxic, if a lung scan shows signs of cancer, if a transaction is fraudulent, etc.)
* *It’s unclear whether this type of problem is common in the industry because they are common in nature or simply because ML practitioners are most comfortable handling them*
* With **> 2 classes**, the problem becomes **multiclass classification**.
* **Dealing with binary classification problems is much easier than dealing with multiclass problems**
* *Ex: Calculating F1 and visualizing confusion matrices are a lot more intuitive w/ only 2 classes*
* **When the number of classes is high**, we say **the classification task has high cardinality** (Ex: disease diagnosis where the number of diseases can go up to thousands, product classifications where the number of products can go up to tens of thousands)
* **High cardinality problems can be very challenging**
* **1st challenge =** **data collection**
* ML models typically need at least 100 examples for each class to learn to classify that class.
* So, if you have 1,000 classes, you already need at least 100,000 examples
* The data collection can be especially difficult for **rare classes**
* *When you have thousands of classes, it’s likely that some of them are rare*
* **When the number of classes is large, hierarchical classification might be useful**
* Here, you **have a classifier to first classify each example into 1 of the *large* groups**
* Then you **have *another* classifier to classify this example into one of the *subgroups***
* Ex: Can first classify a product into 1 of 4 main categories: electronics, home & kitchen, fashion, or pet supplies
* After a product has been classified into a category, say fashion, you can use another classifier to put this product into 1 of the subgroups: shoes, shirts, jeans, or accessories

##### Multiclass versus multilabel classification

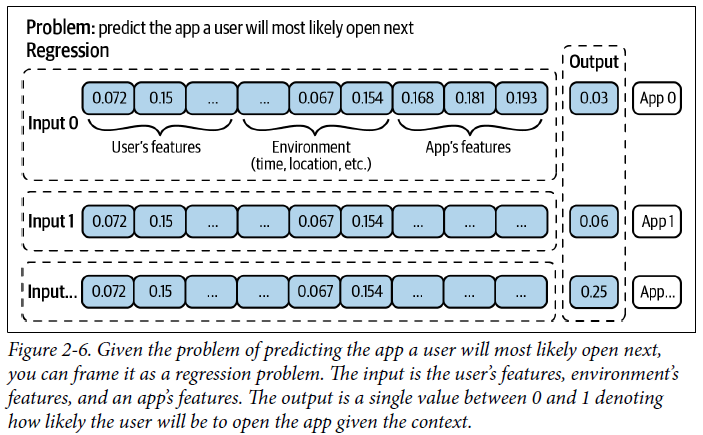
* In both **binary and multiclass classification, each example belongs to exactly 1 class**
* When **an example can belong to *multiple* classes**, we have a **multilabel classification**problem
* Ex: Building a model to classify articles into 4 topics (tech, entertainment, finance, politics), an article can be in both tech *and* finance
* There are **2 major approaches to multilabel classification problems**
* 1) To **treat it as you would a multiclass classification**
* In multiclass classification, if there are 4 possible classes [tech, entertainment, finance, politics] + the label for an example is entertainment, you represent this label with the vector [0, 1, 0, 0]
* In multilabel classification, if an example has *both* labels entertainment and finance, its label will be represented as [0, 1, 1, 0]
* 2) To **turn it into a set of binary classification problems**
* For the article classification problem, you can have 4 models corresponding to 4 topics, each model outputting whether an article is in that topic or not
* **Out of all task types, multilabel classification is usually the one companies have the most problems with**
* **“Multilabel” means that the number of classes an example can have varies from example to example**
* 1) This **makes it difficult for label annotation since it increases the** **label multiplicity problem**
* Ex: An annotator might believe an example belongs to 2 classes while another believes the same example to belong in only 1 class, + it might be difficult resolving this disagreement
* 2) This **varying number of classes makes it hard to extract predictions from raw probability**
* Consider the same task of classifying articles into 4 topics
* Given an article, your model outputs this raw probability distribution: [0.45, 0.2, 0.02, 0.33]
* In the *multiclass* setting, when you know that an example can belong to only *one* category, you simply pick the category with the highest probability
* In the ***multilabel*** setting, *because you* ***don’t know how many categories an example can belong to***, you *might* pick the *two* highest probability categories OR the *three* highest probability categories

##### Multiple ways to frame a problem

* **Changing the way you frame your problem might make the problem significantly harder or easier**
* Ex: Predicting what app a phone user wants to use next
* Naive setup = frame this as a multiclass classification task + use the user’s + environment’s features (user demographic information, time, location, previous apps used) as input, + output a probability distribution for every single app on the user’s phone
* Let *N* be the number of apps you want to consider recommending to a user
* **In this framing, for a given user at a given time, there is only 1 prediction to make, and the prediction is a vector of the size *N***



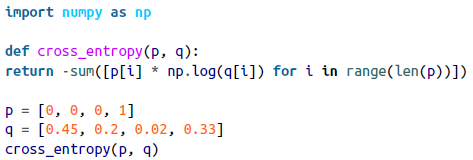
* **This is a bad approach because whenever a new app is added, you might have to retrain your model from scratch, or at least retrain all the components of your model whose number of parameters depends on *N***
* A better approach = frame this as a regression task w/ input as the user’s, the environment’s, *and the app’s features*, + output is a single value between 0 and 1 (the higher the value, the more likely the user will open the app given the context)
* **In *this* framing, for a given user at a given time, there are *N* predictions to make, 1 for each app, *but each prediction is just a number***



* **In this new framing, whenever there’s a new app you want to consider recommending to a user, you simply need to use new inputs with this new app’s feature instead of having to retrain your model or part of your model from scratch**

#### Objective Functions

* **In order to learn, an ML model needs an objective function/loss function to guide the learning process**
* **Objective *functions*** are **mathematical functions**, which ***are different from the business and ML objectives***
* “**Loss** function” = the **objective of the learning process is usually to minimize (or optimize) the loss caused by wrong predictions**
* For *supervised* ML, loss can be computed by comparing model outputs with ground truth labels using a measurement like RMSE or cross entropy
* Ex: Classifying articles into 4 topics: [tech, entertainment, finance, politics]
* Consider an article that belongs to the politics class (ground truth label = [0, 0, 0, 1])
* Given this article, your model outputs this raw probability distribution: [0.45, 0.2, 0.02, 0.33]
* **Cross-entropy loss of this model, for this example, is the cross entropy of [0.45, 0.2, 0.02, 0.33] relative to [0, 0, 0, 1]**
* In Python, you can calculate cross entropy:



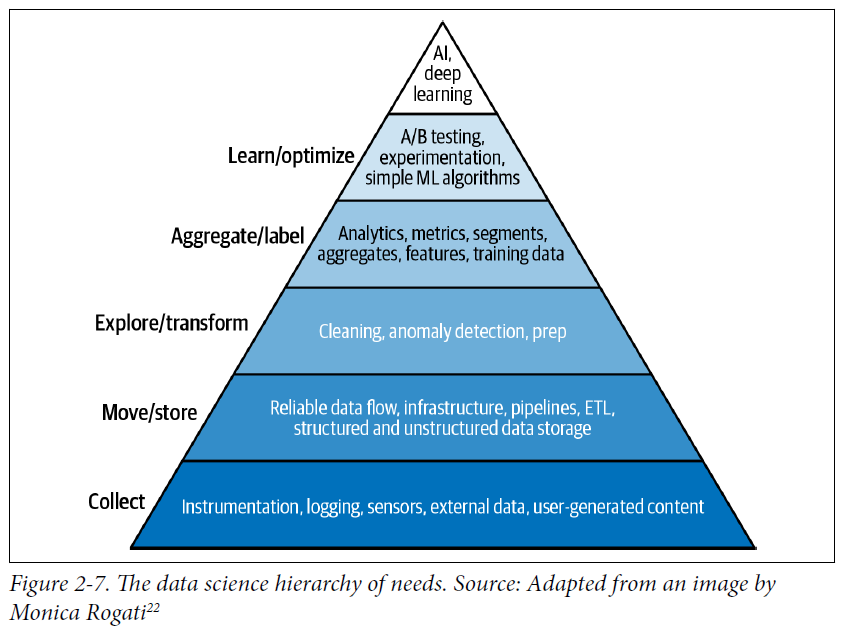
* **Choosing an objective function is usually straightforward, *though not because objective functions are easy***
* **Coming up with *meaningful* objective functions requires algebra knowledge**, so most MLE’s just use common loss functions like RMSE or MAE for regression, logistic loss (also log loss) for binary classification, + cross entropy for multiclass classification

##### Decoupling objectives

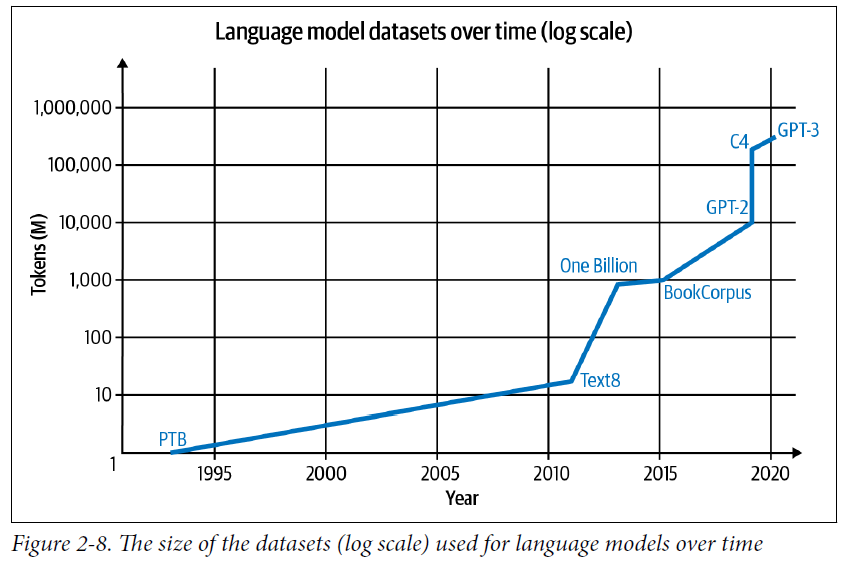
* **Framing ML problems can be tricky when you want to minimize *multiple* objective functions**
* Ex: Building a system to rank items on users’ newsfeeds w/ the goal to maximize users engagement achieved through the following 3 objectives:
* Filter out spam
* Filter out NSFW content
* Rank posts by engagement: how likely users will click on it
* However, you quickly learned that optimizing for users’ engagement alone can lead to questionable ethical concerns
* Because extreme posts tend to get more engagements, your algorithm learned to prioritize extreme content
* You want to create a more wholesome newsfeed, so you have a *new* goal: maximize users’ engagement *while minimizing the spread of extreme views and misinformation*
* To obtain this new goal, you *add 2 new objectives to your original plan:*
* Filter out spam
* Filter out NSFW content
* Filter out misinformation
* Rank posts by quality
* Rank posts by engagement: how likely users will click on it
* **Now 2 objectives are in conflict with each other** 🡪 *If a post is engaging but it’s of questionable quality, should that post rank high or low?*
* **An objective is represented by an objective function**
* To rank posts by quality, you first need to predict posts’ quality, + you want posts’ predicted quality to be as close to the actual quality as possible 🡪 Essentially, you **want to minimize quality\_loss: the difference between each post’s predicted quality and its true quality** (for simplicity, pretend we know how to measure a post’s quality)
* *Similarly*, to rank posts by engagement, you first need to predict the number of clicks each post will get 🡪 **want to minimize engagement\_loss: the difference between each post’s predicted clicks and its actual number of clicks**
* **1 approach is to combine these 2 losses into 1 loss and train 1 model to minimize that loss:**
* loss = α\*quality\_loss + β\*engagement\_loss
* **Can randomly test out different values of αand βto find the values that work best**
* To be **more systematic** about tuning these values, check out **Pareto optimization**: **an area of *multiple* criteria decision making concerned with mathematical optimization problems involving more than 1 objective function to be optimized simultaneously**
* **A problem with this approach is that *each time you tune α and******β* (say if the quality of your users’ newsfeeds goes up but users’ engagement goes down, you might want to decrease αand increase β), you’ll have to retrain your model**
* **Another approach is to train 2 *different* models, each optimizing 1 loss**
* **quality\_model**: **Minimizes quality\_loss**and **outputs predicted quality of each post**
* **engagement\_model**: **Minimizes engagement\_loss**and **outputs predicted number of clicks of each post**
* You **can combine the models’ outputs and rank posts by their combined scores:**
* α\*quality\_score + β\*engagement\_score
* Now you **can tweak α\*and β\*without retraining your models**!
* **In general, when there are multiple objectives, it’s a good idea to decouple them first because it makes model development and maintenance easier**
* **1) Easier to tweak your system without retraining models,** as explained above
* **2) Easier for maintenance since different objectives might need different maintenance schedules**
* Ex: Spamming techniques evolve much faster than the way post quality is perceived, so spam filtering systems need updates at a much higher frequency than quality-ranking systems

### Mind Versus Data

* Progress in the last decade shows that **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**
* Public debates on the power of **mind versus 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**
* Dr. Judea Pearl (Turing Award winner, work on causal inference + Bayesian networks) emphasizes “Data is profoundly dumb” + warned that data-centric ML people might be out of a job in 3-5 years
* Professor Christopher Manning, director of the Stanford AI Laboratory: “huge computation + 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 less data**”
* **In theory, you can both pursue architectural designs *and* leverage large data + computation, but spending time on one often takes time away from another**
* **Many people in ML today are in 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 claimed that researchers who chose to pursue intelligent designs over methods that leverage computation will eventually learn a bitter lesson
* “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”
* Peter Norvig, Google’s director of search quality, speaking on Google 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”
* Dr. Monica Rogati, former VP of data at Jawbone, argued that data lies at the foundation of data science: **If you want to use data science, a discipline of which ML 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”*



* **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, it might be possible for us to 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 recent decades **show the success of ML relies more and more on the *quality* and *quantity* of data**
* **Models are also getting bigger and using more data**
* 2013: the One Billion Word Benchmark for Language Modeling was released, containing 0.8 billion tokens
* 6 years later, OpenAI’s GPT-2 used a dataset of 10 billion tokens, + another year later, GPT-3 used 500 billion tokens



* **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 a** **model**
* **More data at *lower quality* (outdated, incorrect labels) might even *hurt* model performance**

### Summary

* **Every project must start with WHY this project needs to happen**, and **ML projects are no exception**
* **Assumption = most businesses don’t care about ML metrics *unless they can move business metrics***
* An **ML system is built for a business must be motivated by business objectives, which need to be translated into ML objectives to guide the development of ML models**.
* **Before building** an ML system, we **need to understand the requirements that the system needs to meet to be considered a good system**
* **Exact requirements vary from use case to use case**
* **4 most general requirements**: **reliability, scalability, maintainability, + adaptability**

Building an ML system isn’t a one-off task but an **iterative process that met those preceding requirements**

* Philosophical discussion of the role of data in ML systems
* Many still believe that having intelligent algorithms will eventually trump having a large amount of data
* However, **the success of systems** (like AlexNet, BERT, GPT, etc.) **showed that the progress of ML in the last decade at least 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**
* **Complex ML systems are made up of simpler building blocks**