



Setting up a Machine Learning System

Let's go over the important steps that are mostly common among different ML-based systems. We will use this framework in problems that we discuss later in the course.

We'll cover the following



- Overview
- Key steps in ML system setup
 - Setting up the problem
 - Understanding scale and latency requirements
 - Defining metrics
 - Architecture discussion
 - Architecting for scale
 - Offline model building and evaluation
 - Online model execution and evaluation
 - Iterative model improvement

Overview#

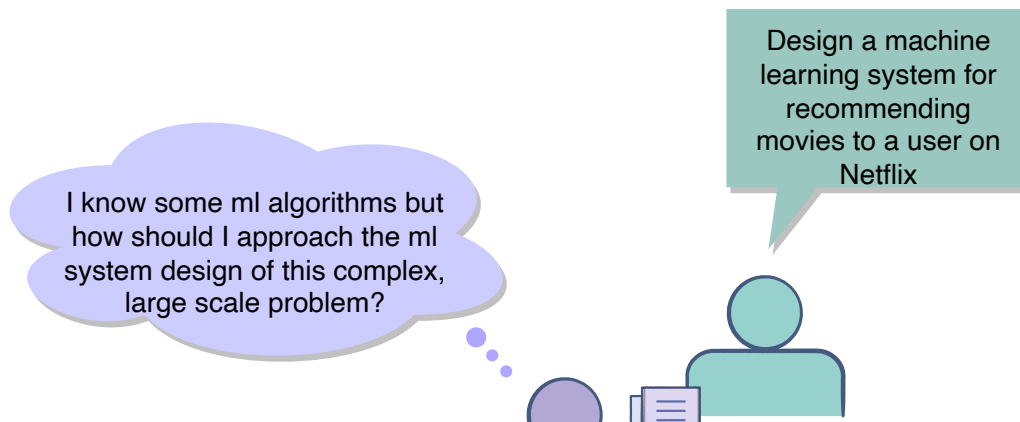
Interviewers will generally ask you to design a machine learning system for a particular task. For instance, they may ask you to design the machine learning system for the following problems:

- *Build a system that shows relevant ads for search engines.*
- *Extract all persons, locations, and organizations from a given corpus of documents.*
- *Recommend movies to a user on Netflix.*





In your interview preparation, you would find a plethora of resources to tell you details about how different machine learning models work. However, there are not many resources that show you how to approach the ML system design of such a complex and large scale problem.



Interviewee

What you are searching for is a comprehensive resource that provides practical knowledge to kickstart your preparation for machine learning interviews. Fortunately, this course concludes your search. 🎉

Before diving into the chapters, let's get you familiarized with the thought process required to answer an interviewer's questions.

Key steps in ML system setup#

In the following chapters, you will observe that the key steps involved in machine learning project set up are as follows:

Setting up the problem#

The interviewer's question is generally very broad. So, the first thing you need to do is *ask questions*. Asking questions will close the gap between your

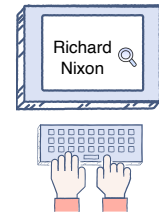


understanding of the question and the interviewer's expectations for the problem.



answer. You will be able to narrow down your problem space, chalk out the requirements of the system, and finally arrive at a precise machine learning problem statement.

For instance, you may be asked to design a search engine that displays the most relevant results in response to user queries. You could narrow down the problem's scope by asking the following questions:




- *Is it a general search engine like Google or Bing or a specialized search engine like Amazon's products search?*
- *What kind of queries is it expected to answer?*

This will allow you to precisely define your ML problem statement as follows:

Build a generic search engine that returns relevant results for queries like "Richard Nixon", "Programming languages" etc.

Or, you may be asked to build a system to display a Twitter feed for a user. In this case, you can discuss how the feed is currently displayed and how it can be improved to provide a better experience for the users.

 In the feed-based system [chapter](#), we discuss how the Twitter feed was previously displayed in chronological order, causing users to miss out on relevant tweets. From this discussion, we realized that we want to move towards displaying content in *order of relevance* instead. Read this chapter to find out more!





After inspecting the problem from all aspects, you can easily narrow it down to a precise machine learning problem statement as follows:

“Given a list of tweets, train an ML model that predicts the probability of engagement of tweets and orders them based on that score.”



Some problems may require you to think about hardware components that could provide input for the machine learning models.

Understanding scale and latency requirements#

Another very important part of the problem setup is the discussion about *performance and capacity considerations* of the system. This conversation will allow you to clearly understand the scale of the system and its requirements.

Let's look at some examples of the questions you need to ask.

Latency requirements

If you were given the search engine problem, you would ask:

- *Do we want to return the search result in 100 milliseconds or 500 milliseconds?*

Similarly, if you were given the Twitter feed problem, you would ask:

- *Do we want to return the list of relevant tweets in 300 milliseconds or 400 milliseconds?*





Scale of the data

Again, for the search engine problem, you would ask:

- *How many requests per second do we anticipate to handle?*
- *How many websites exist that we want to enable through this search engine?*
- *If a query has 10 billion matching documents, how many of these would be ranked by our model?*

And, for the Twitter feed problem, you would ask:

- *How many tweets would we have to rank according to relevance for a user at a time?*



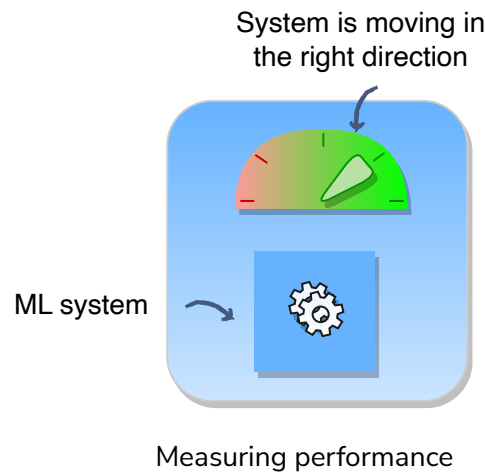
Performance and capacity considerations are detailed in this [lesson](#).

The answers to these questions will guide you when you come up with the architecture of the system. Knowing that you need to return results quickly will influence the depth and complexity of your models. Having huge amounts of data to process, you will design the system with scalability in mind. Find more on this in the *architecture discussion* section.

Defining metrics#

Now that you have figured out what machine learning problem you want to solve, the next step is to come up with metrics. Metrics will help you to see if your system is performing well.





✎ Knowing our success criteria helps in understanding the problem and in selecting key architectural components. This is why it's important to discuss metrics early in our design discussions.

Metrics for offline testing

You will use *offline metrics* to quickly test the models' performance during the development phase. You may have generic metrics; for example, if you are performing binary classification, you will use AUC, log loss, precision, recall, and F1-score. In other cases, you might have to come up with specific metrics for a certain problem. For instance, for the search ranking problem, you would use NDCG as a metric.

Metrics for online testing

Once you have selected the best performing models offline, you will use *online metrics* to test them in the production environment. The decision to deploy the newly created model depends on its performance in an online test.

While coming up with online metrics, you may need both **component-wise** and **end-to-end** metrics. Consider that you are making a search ranking



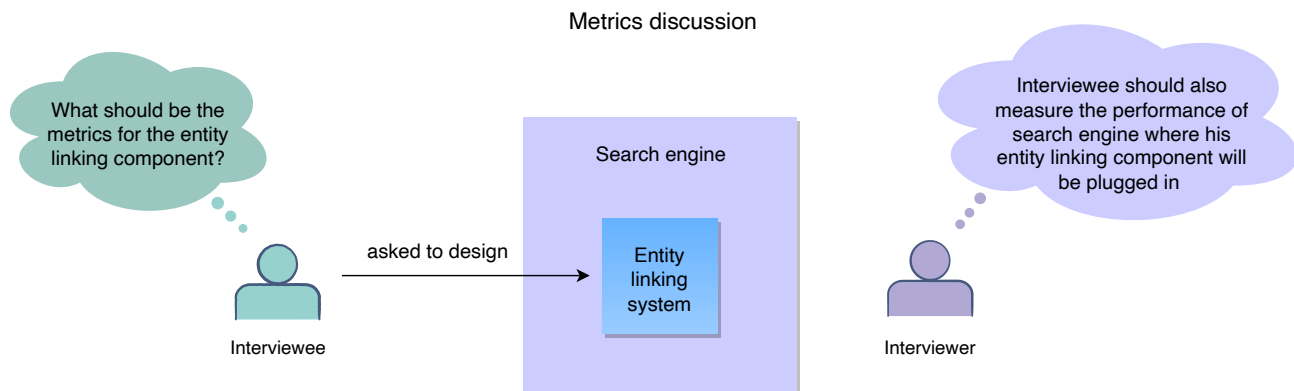
and end-to-end metrics. Consider that you are making a search ranking

model to display relevant results for search queries. You may



component-wise metric such as NDCG to measure the performance of your model online. However, you also need to look at how the system (search engine) is performing with your new model plugged in, for which you can use end-to-end metrics. A commonly used end-to-end metric for this scenario is the users' engagement and retention rate.

In another scenario, you may be asked to develop the ML system for a task that may be used as a component in other tasks. Again, you need both component level metrics and end-to-end metrics during online testing. For instance, you could be asked to design the ML system for *entity linking* which is going to be used to improve *search relevance*.



Think of end-to-end metrics as well as component metrics

Here, you will have component-wise metrics to evaluate the performance of the entity linking model individually. You will also be expected to come up with metrics for the search engine where the entity linking component will ultimately be plugged in.


Architecture discussion#

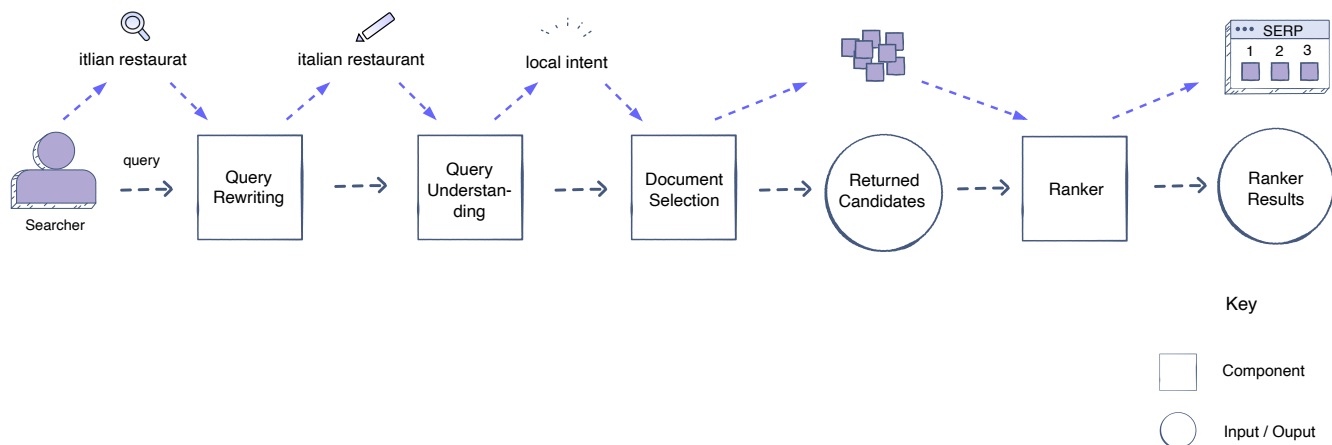
The next step is to figure out the architecture of the system. You need to think about the components of the system and how the data will flow through those components.





To get an idea, have a peek at how the architecture of a search engine's ML system may be designed, below.

 This is a simplified version of the architectural diagram. For complete details, read the [chapter](#) on search ranking.



Architectural components for ML system of search engine

You start off with a searcher making a query on the search engine. Let's take an example query: "itlian restaurant". The query may be poorly worded or misspelled so you need the query rewriting component to rewrite it. It would correct the query to "italian restaurant". This allows you to retrieve better results. Moving forward, the query understanding component helps identify the intent of a query. The "italian restaurant" query has local intent. Next, the document selection component selects relevant documents from the billions of documents on the web. The ranker component ranks them according to relevance. Finally, the results are displayed on the search engine result page (SERP).

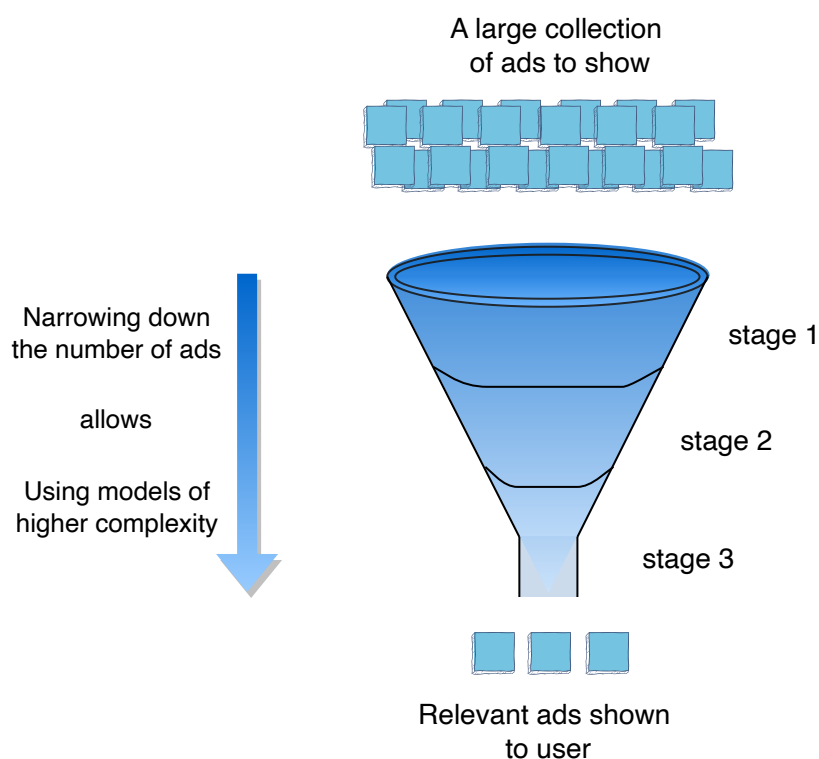
Architecting for scale#





As we mentioned previously, the requirements gathered during problem setup help you in chalking out the architecture. For instance, you are tasked with building an ML system that displays relevant ads to users. During its problem setup, you ask questions and realize that the number of users and ads in the system is huge and ever-increasing. Thus, you need a scalable system that quickly figures out the relevant ads for all users despite the increase in data.

Hence, you can't just build a complex ML model and run it for all ads in the system because it would take up a lot of time and resources. The solution is to use the funnel approach, where each stage will have fewer ads to process. This way, you can safely use complex models in later stages.



Funnel approach: quickly get relevant ads for a user

Offline model building and evaluation#



The next step is to start building the model offline and then evaluate it. This

step involves:

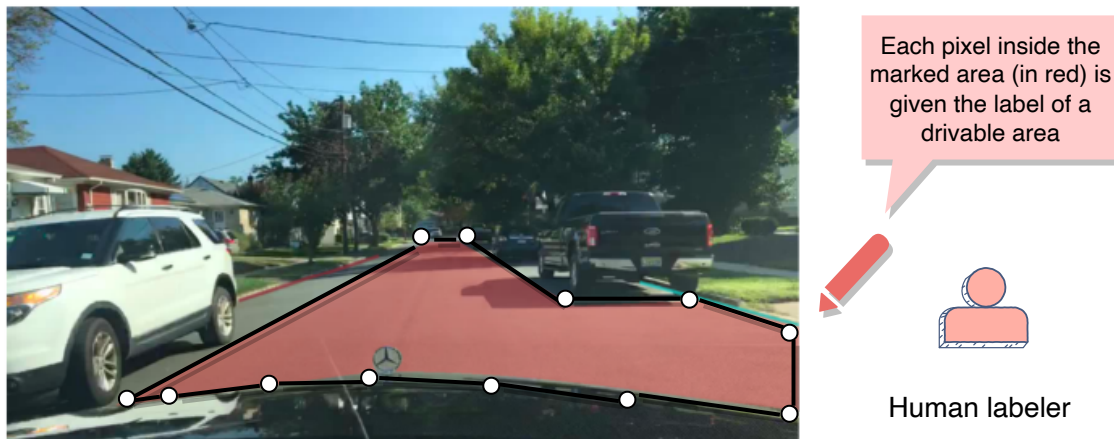


- **Training data generation:**

Training data is food for learning. Therefore, it is crucial that your training data is of good quality and quantity. Broadly speaking, we have two methods of training data collection/generation for supervised learning tasks:

1. Human labeled data

We can hire labelers who can label data for us according to the given ML task. For instance, if we have to perform segmentation of driving images, labelers will use software such as *label box* to mark the boundaries of different objects in the driving images.



Manual labeling of driving images to generate training data for segmentation

This is an expensive way to gather data. So we need to supplement it with in-house labelers or *open-source datasets*. “[BDD100K: A Large-scale Diverse Driving Video Database](#)” is an example of an open-source dataset that can be used as training data for the segmentation task. It contains segmented data for driving images.

 You will see another way to enhance training data in the image segmentation chapter!



2. Data collection through a user's interaction with the pre-deployed/pre-existing system



Another way to gather data is through the online (currently deployed/pre-existing) system.



Online system refers to a running system, currently in place, that people are interacting with. For example, a running search engine or ad system that people are using.

To get an idea, let's go back to the search ranking example, where it was asked to design a new ML-based system to show relevant search results. The currently deployed version of the search engine would also show *results* for user queries (using an existing ML-based system or a rule-based system). You can see how people interact with these *results* to gather training data. If a user clicks on a result (link), you can count it as a positive training example. Similarly, an impression can count as a negative example (refer to the search ranking chapter for details).



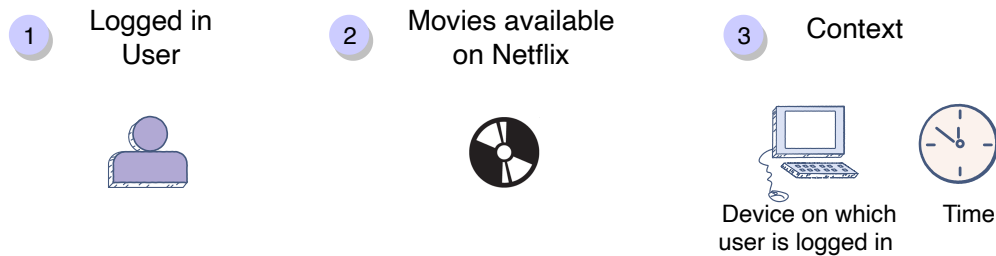
The training data collection strategies are detailed in the [lesson](#). Have a look to find out some innovative ways to collect and expand your training data.

- **Feature engineering:**

This is another very crucial step as good features influence the model's ability to a great extent. You start this process by explicitly pinpointing the actors involved in the given task, which you have implicitly identified during problem setup as well.

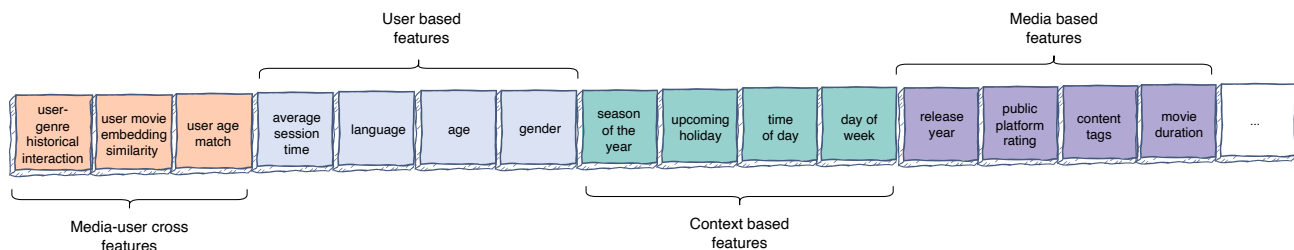
For example, if the task at hand is to make movie recommendations to a Netflix user, you would have the following actors:





Main actors involved in the movie recommendation task

In order to make features, you would individually inspect these actors and explore their relationships too. For instance, if you individually inspect the logged-in user, you can come up with features such as the user's age, gender, language, etc. Likewise, if you look at the context, an important feature could be the “upcoming holiday”. If Christmas is approaching and the movie under consideration is also a Christmas movie, there is a greater chance that the user would watch it. Similarly, we can look at the historical engagement between the user and media to come up with features. An example could be the user's interaction with the movie's genre in the last three months.



A subset of features in the training data row



For more ideas and details, read the [chapter](#) on recommendation systems.

• Model training:


Now, you can finally decide on the ML models that you should use for the given tasks, keeping the performance and capacity considerations in



mind. We can also try out different hyperparameter values to see what works best.


If you are using the funnel approach, you may select simpler models for the top of the funnel where data size is huge and more complex neural networks or trees based models for successive parts of the funnel.

We also have the option of utilizing pre-trained SOTA (state of the art) models to leverage the power of transfer learning (you don't need to reinvent the wheel completely each time).

 Look at how to use pre-trained SOTA models in the entity linking [chapter](#).

- **Offline evaluation:**

With careful consideration, divide the data into training and validation sets. Use the training data set during the model training part, where you come up with various modeling options (different models, hyperparameters, feature sets, etc.). Afterwards, we evaluate these models, *offline*, on the validation dataset. Utilize the metrics you decided on earlier, to measure their performance. The top few models, showing the most promise, are taken to the next stage.

 Offline learning is very beneficial, as it allows us to quickly test many different models so that we can select the best one for online testing, which is a slow process.

Online model execution and evaluation#



Now that you have selected the top-performing models, you will test them in

an online environment. Online testing heavily influences the decision of deploying the model. This is where online metrics come into



Depending on the type of problem, you may use both component level and end-to-end metrics. As mentioned before, for the search engine task, the component-wise metric will be NDCG in online testing. However, this alone is not enough. You also need an end-to-end metric as well, like session success rate, to see if the system's (search engine's) performance has increased by using your new search ranking ML model.

If you see a *substantial* increase in system performance during the online test, you can deploy it on production.



For more details, read the [lesson](#) on online experimentation.

Iterative model improvement#

Your model may perform well during offline testing, but the same increase in performance may not be observed during an online test. Here, you need to think about *debugging the model* to find out what exactly is causing this behavior.

Is a particular component not working correctly? Is the features' distribution different during training and testing time? For instance, a feature called "user's top five interest" may show a difference in distribution during training and testing, when plotted. This can help you to identify that the routines used to provide the top five user interests were significantly different during training and testing time.

Moreover, after the first version of your model has been built and deployed, you still need to monitor its performance. If the model is not performing as expected, you need to go towards debugging. You may observe a general failure from a decrease in AUC. Or, you may note that the model is failing in



particular scenarios. For instance, by analysing the video recording of a



self-driving car, you may find out that the image segmentation fails in rushy areas.

The problem areas identified during model debugging will guide you in building successive iterations of your model.

 Get some model debugging ideas in the model debugging & testing lesson.


 Back

Next 

How Does This Course Help in ML Int...

Performance and Capacity Considerati...

☒ Mark as Completed

 Report an Issue

