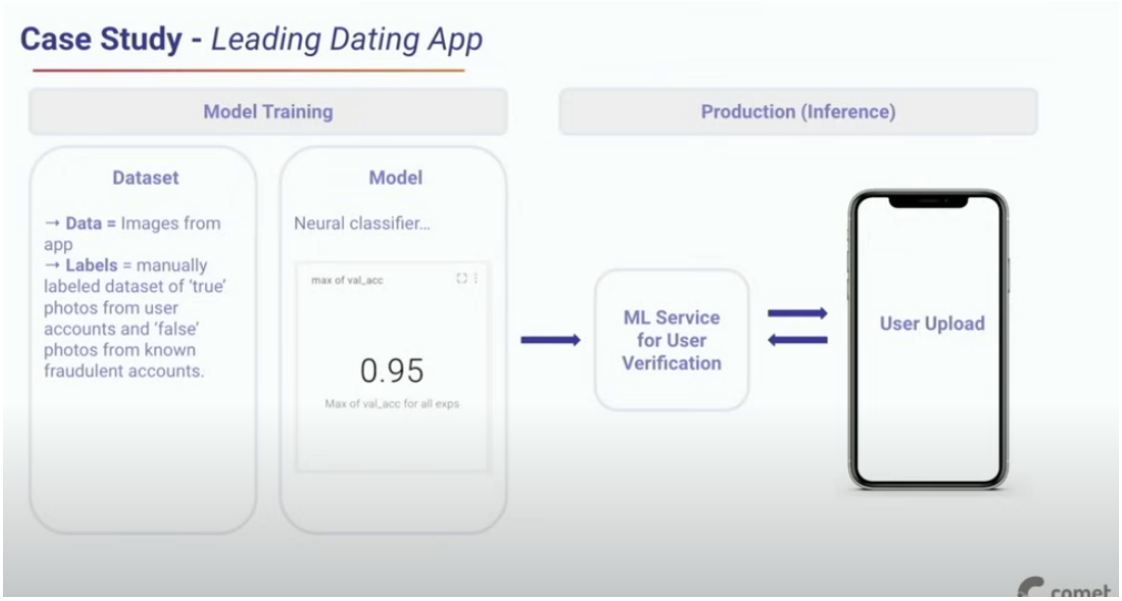
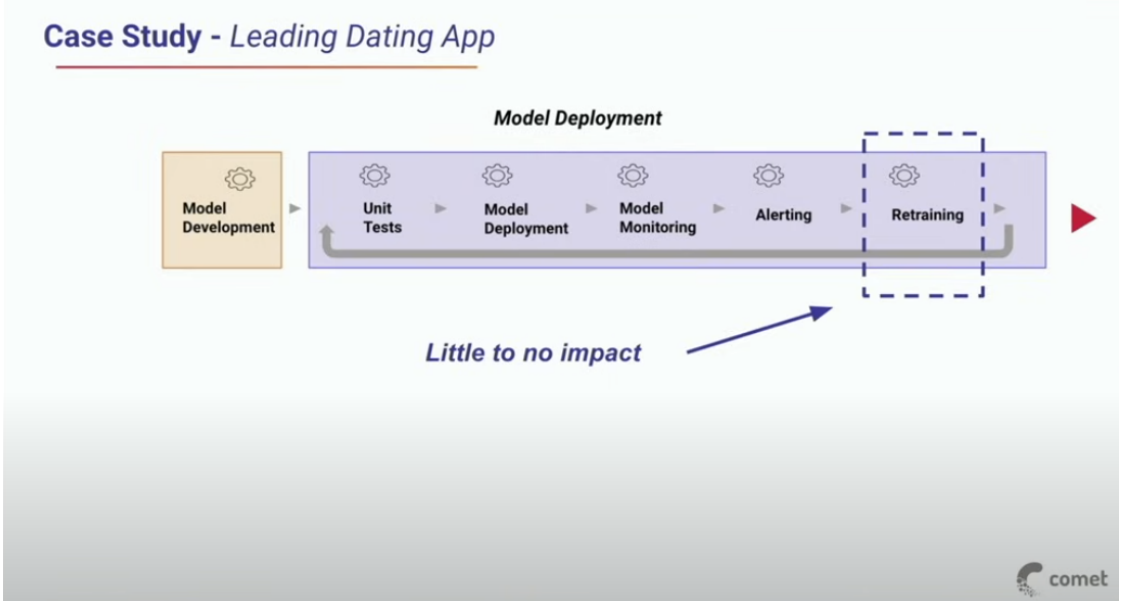
Wild Screenshots Collection

This document contains all the screenshots named in the format Screenshot_wild-xx.png.

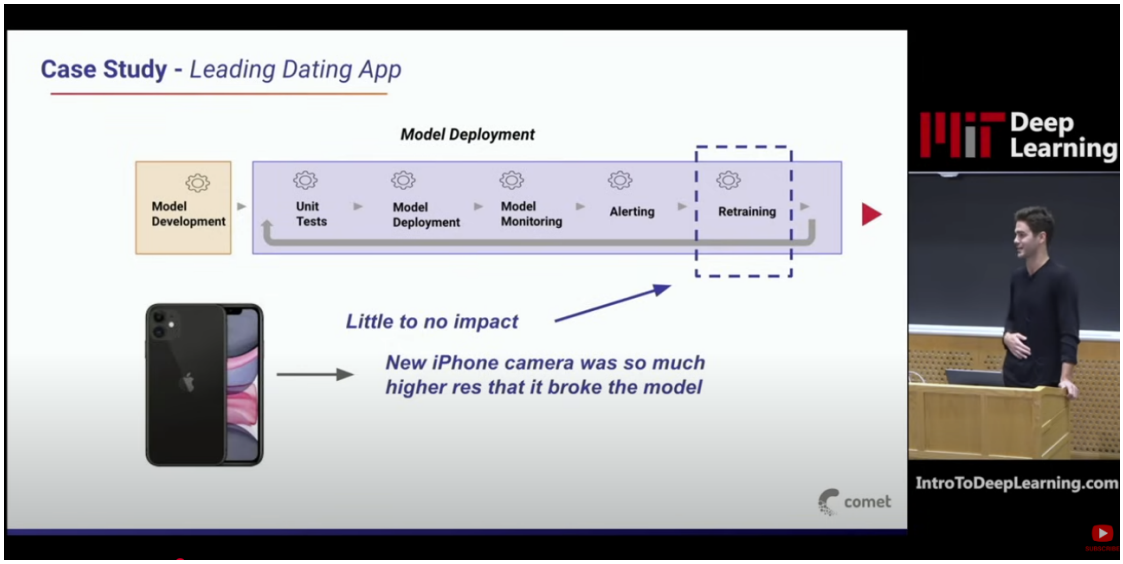
Screenshot_wild-01



Screenshot_wild-02



Screenshot_wild-03



Screenshot_wild-04

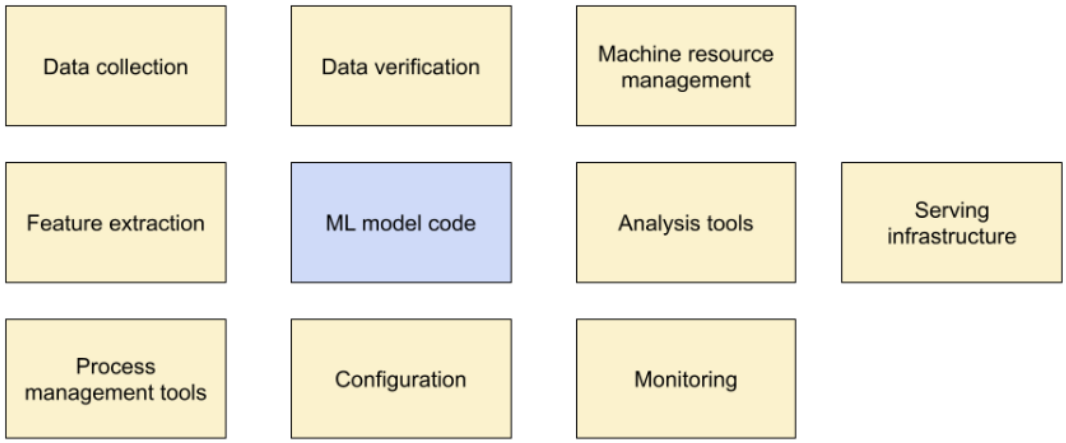


Figure 1. *A real-world production ML system comprises many components.*

Screenshot_wild-05

Table 1. Primary advantages and disadvantages.

	Static training	Dynamic training
Advantages	**Simpler.** You only need to develop and test the model once.	**More adaptable.** Your model will keep up with any changes to the relationship between features and labels.
Disadvantages	**Sometimes staler.** If the relationship between features and labels changes over time, your model's predictions will degrade.	**More work.** You must build, test, and release a new product all the time.

Screenshot_wild-06

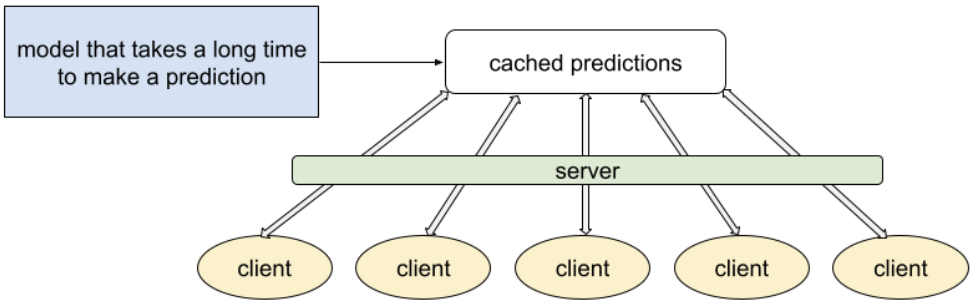


Figure 4. *In static inference, a model generates predictions, which are then cached on a server.*

Suppose this same complex model mistakenly uses dynamic inference instead of static inference. If many clients request predictions around the same time, most of them won't receive that prediction for hours or days.

Now consider a model that infers quickly, perhaps in 2 milliseconds using a relative minimum of computational resources. In this situation, clients can receive predictions quickly and efficiently through dynamic inference, as suggested in Figure 5.

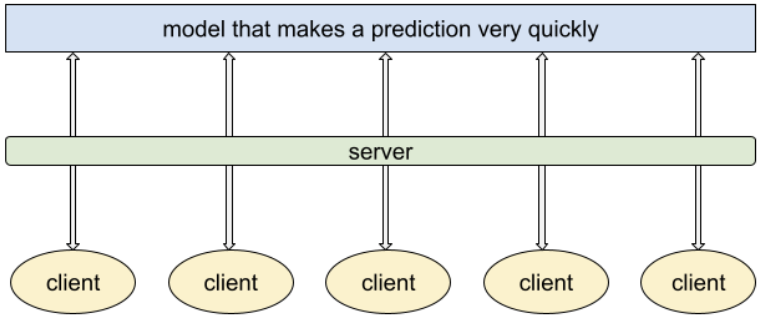


Figure 5. *In dynamic inference, a model infers predictions on demand.*

Screenshot_wild-07

Static inference

Static inference offers certain advantages and disadvantages.

Advantages

- Don't need to worry much about cost of inference.
- Can do post-verification of predictions before pushing.

Disadvantages

- Can only serve cached predictions, so the system might not be able to serve predictions for uncommon input examples.
- Update latency is likely measured in hours or days.

Dynamic inference

Dynamic inference offers certain advantages and disadvantages.

Advantages

- Can infer a prediction on *any* new item as it comes in, which is great for long tail (less common) predictions.

Disadvantages

- Compute intensive and latency sensitive. This combination may limit model complexity; that is, you might have to build a simpler model that can infer predictions more quickly than a complex model could.
- Monitoring needs are more intensive.

Screenshot_wild-08

Transforming data before training

In this approach, you follow two steps:

1. Write code or use specialized tools to transform the raw data.
2. Store the transformed data somewhere that the model can ingest, such as on disk.

Advantages

- The system transforms raw data only once.
- The system can analyze the entire dataset to determine the best transformation strategy.

Disadvantages

- You must recreate the transformations at prediction time. Beware of training-serving skew!

Training-serving skew is more dangerous when your system performs dynamic (online) inference. On a system that uses dynamic inference, the software that transforms the raw dataset usually differs from the software that serves predictions, which can cause training-serving skew. In contrast, systems that use static (offline) inference can sometimes use the same software.

Transforming data while training

In this approach, the transformation is part of the model code. The model ingests raw data and transforms it.

Advantages

- You can still use the same raw data files if you change the transformations.
- You're ensured the same transformations at training and prediction time.

Disadvantages

- Complicated transforms can increase model latency.
- Transformations occur for each and every batch.

Screenshot_wild-09

Check for training-serving skew

Training-serving skew means your input data during training differs from your input data in serving. The following table describes the two important types of skew:

Type	Definition	Example	Solution
Schema skew	Training and serving input data do not conform to the same schema.	The format or distribution of the serving data changes while your model continues to train on old data.	Use the same schema to validate training and serving data. Ensure you separately check for statistics not checked by your schema, such as the fraction of missing values
Feature skew	Engineered data differs between training and serving.	Feature engineering code differs between training and serving, producing different engineered data.	Similar to schema skew, apply the same statistical rules across training and serving engineered data. Track the number of detected skewed features, and the ratio of skewed examples per feature.

Screenshot_wild-10

Is each feature helpful?

You should continuously monitor your model to remove features that contribute little or nothing to the model's predictiv ability. If the input data for that feature abruptly changes, your model's behavior might also abruptly change in undesirable ways.

Also consider the following related question:

- Does the usefulness of the feature justify the cost of including it?

It is always tempting to add more features to the model. For example, suppose you find a new feature whose addition makes your model's predictions slightly better. Slightly better predictions certainly seem better than slightly worse predictions; however, the extra feature adds to your maintenance burden.

Is your data source reliable?

Some questions to ask about the reliability of your input data:

- Is the signal always going to be available or is it coming from an unreliable source? For example:
  - Is the signal coming from a server that crashes under heavy load?
  - Is the signal coming from humans that go on vacation every August?
- Does the system that computes your model's input data ever change? If so:
  - How often?
  - How will you know when that system changes?

Consider creating your own copy of the data you receive from the upstream process. Then, only advance to the next version of the upstream data when you are certain that it is safe to do so.

Is your model part of a feedback loop?

Sometimes a model can affect its own training data. For example, the results from some models, in turn, become (directly or indirectly) input features to that same model.

Sometimes a model can affect another model. For example, consider two models for predicting stock prices:

- Model A, which is a bad predictive model.
- Model B.