Machine Learning Systems Versus Traditional Software

Since ML is part of software engineering (SWE), and software has been successfully used in production for more than half a century, some might wonder why we don't just take tried-and-true best practices in software engineering and apply them to ML.

That's an excellent idea. In fact, ML production would be a much better place if ML experts were better software engineers. Many traditional SWE tools can be used to develop and deploy ML applications.

However, many challenges are unique to ML applications and require their own tools. In SWE, there's an underlying assumption that code and data are separated. In fact, in SWE, we want to keep things as modular and separate as possible (see the Wikipedia page on separation of concerns (*https://oreil.ly/kH67y*)).

On the contrary, ML systems are part code, part data, and part artifacts created from the two. The trend in the last decade shows that applications developed with the most/best data win. Instead of focusing on improving ML algorithms, most companies will focus on improving their data. Because data can change quickly, ML applications need to be adaptive to the changing environment, which might require faster development and deployment cycles.

In traditional SWE, you only need to focus on testing and versioning your code. With ML, we have to test and version our data too, and that's the hard part. How to version large datasets? How to know if a data sample is good or bad for your system? Not all data samples are equal—some are more valuable to your model than others. For example, if your model has already trained on one million scans of normal lungs and only one thousand scans of cancerous lungs, a scan of a cancerous lung is much more valuable than a scan of a normal lung. Indiscriminately accepting all available data might hurt your model's performance and even make it susceptible to data poisoning attacks.[31]

The size of ML models is another challenge. As of 2022, it's common for ML models to have hundreds of millions, if not billions, of parameters, which requires gigabytes of random-access memory (RAM) to load them into memory. A few years from now, a billion parameters might seem quaint—like, "Can you believe the computer that sent men to the moon only had 32 MB of RAM?"

However, for now, getting these large models into production, especially on edge devices,[32] 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

31 Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song, "Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning," *arXiv*, December 15, 2017, *https://oreil.ly/OkAjb*.

32 We'll cover edge devices in Chapter 7.

the time it takes to suggest the next character is longer than the time it takes for you to type.

Monitoring and debugging these models in production is also 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. Back in 2018, when the Bidirectional Encoder Representations from Transformers (BERT) paper first came out, people were talking about how BERT was too big, too complex, and too slow to be practical. The pretrained large BERT model has 340 million parameters and is 1.35 GB.[33] Fast-forward two years later, BERT and its variants were already used in almost every English search on Google.[34]