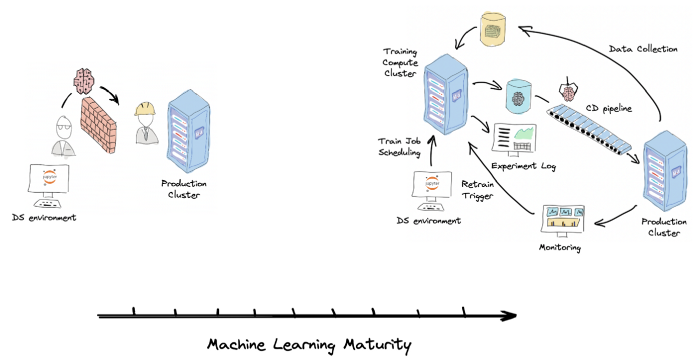
**Maturity of Machine Learning Systems**

**An overview of existing maturity frameworks for Machine Learning systems**



Machine Learning System Maturity. Image provided by the author

Low-quality Machine Learning Systems are bad for the business. They are hard to adapt and improve to serve business needs. At the same time, they are expensive. They consume a lot of operational resources on bug fixing and debugging. They tend to accumulate technical debt and entropy over time. They are fragile, non-reliable, and lead to bad customer experience. Thus it is crucial to continuously improve the quality and maturity of our Machine Learning Systems.

Yet, in my experience, low-quality systems prevail in the Machine Learning industry. Why? In my opinion, the lack of structured guidance is the most significant factor. Without such guidance, improvement activities are usually scarce and non-systematic. Machine Learning teams often have to sacrifice them for more tangible business goals. Or they get overwhelmed with the requirements of a fully mature environment without knowing the incremental steps to reach it.

In this article, I try to solve this problem and provide an overview of three existing frameworks that can guide your ML team in the direction of building mature and production-ready Machine Learning Systems:

* **Google Cloud MLOps Maturity Levels** and **Azure MLOps Maturity Model** — generic frameworks that focus on adopting MLOps practices such as CI/CD and Continuous Training.
* **ML Test Score** — Google framework that focuses on testing and monitoring. It provides a single metric to quantify Machine Learning System maturity and a backlog of concrete activities to improve it.

**Machine Learning Maturity**

When we talk about real-world applications of Machine Learning, we need to think about Machine Learning systems. Not just Machine Learning models. In [one of my previous articles](https://towardsdatascience.com/machine-learning-systems-versus-machine-learning-models-3955d038ea1f), I outlined the difference between these concepts and provided some examples. The former contains the latter and defines how we train and use it. I like to define the term Machine Learning System as all activities together with their artifacts that deliver value to a customer using a Machine Learning approach.

Why is it helpful to talk about Machine Learning systems instead of just focusing on improving our model quality? Because creating reliable, production-level Machine Learning systems exposes many new problems not found in small toy examples or research experiments. Machine Learning systems are hard to build and maintain.

* **Machine Learning systems are fragile**. We make a lot of implicit assumptions when we build them. A slight shift in data distribution or contract can make the whole system misbehave. In contrast with traditional software systems, these errors are hard to spot.
* **They accumulate technical debt.** Machine Learning systems have many additional sources of debt compared to traditional software. Technical debt is hard to quantify and prioritize work on it. Over time it consumes more and more operational costs and slows down any improvements.
* **They require the collaboration of siloed teams with misaligned incentives.** Data Scientists, ML Engineers, and Data Engineers in the default setup have contradicting goals and KPIs, which create an internal conflict in the system.

For the Machine Learning system to be production-ready, it must deal with these problems. That’s what we mean by maturity. Machine Learning maturity is a property of a Machine Learning system, not a Machine Learning model. It reflects:

* How fast your Data Science team is capable of making model improvements.
* The number of errors that happen in production.
* How fast we can identify these errors and recover from them.

Machine Learning Maturity is a property of a Machine Learning system, not a Machine Learning model.

Let’s dive into existing Machine Learning maturity frameworks.

**Google Cloud MLOps Maturity Levels**

This framework is defined in [this Google Cloud Architecture Center document](https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning). It puts emphasis on MLOps.

MLOps is a movement that adapts DevOps practices and culture to Machine Learning Systems. It aims at unifying Data Engineers, ML Engineers, Software Engineers, and Data Scientists by aligning their incentives and breaking the silos. DevOps has shown great success in Software Engineering. It helped resolve long-standing conflicts between different teams — Developers, Operations, Business, and Security. It has also shown that agility and robustness are not enemies, and it is possible to achieve both without sacrificing one. With MLOps, we try to replicate the core ideas of DevOps in the field of ML Engineering.

**Practices to Increase Robustness of ML Systems**

The framework highlights three broad practices to increase the robustness of ML systems:

* Continuously experiment with new implementations of ML pipelines
* Monitor the quality of models and data in production
* Frequently retrain production models

We implement these practices by introducing Continuous Integration, Continuous Delivery, and Continuous Training to our Machine Learning systems.

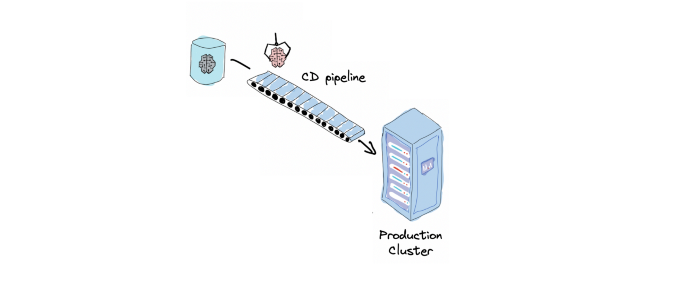
**Continuous Integration**

Continuous Integration (CI) is a DevOps practice that advocates that each team member’s work results must be merged frequently into a single source of truth called the trunk in a Version Control System (VCS). The latter is usually Git. Frequently means daily. And the trunk must always be ready to be deployed to development, staging, or production environments. That means we must guarantee that the trunk is stable while encouraging frequent changes. It sounds contradicting. The solution to this conflict is test automation. Continuous Integration is all about implementing all kinds of automated tests and integrating them into your pipeline, decreasing the chances of breaking the trunk.

Continuous Integration for ML projects means that Data Scientists and ML Engineers use VCS to share and collaborate on their functionality. They continuously refactor and offload the critical functionality from Jupyter notebooks to proper programming language source files (Python packages). They implement tests for their training and prediction pipelines and data. They continuously sync with the trunk.

CI can often be tricky in ML projects due to the lack of engineering skills of some team members and a one-time-use experimentation nature of a big chunk of a Data Science code base.

**Continuous Delivery**



Continuous ML Model Delivery. Image provided by the author.

Continuous Delivery is the next logical step after CI. While CI ensures that the codebase has all the latest changes in a single version that functions correctly, Continuous Delivery ensures that it is deployable. Deployable means it is ready to be deployed fast without manual work. Ideally, it should be just one button click. Continuous Delivery makes it possible again by automating all the steps in a deployment workflow — building packages and images, running integration and acceptance tests, security tests, and deploying to different environments.

Continuous Delivery for ML projects also adds another layer of complexity compared to traditional software development. We need to deliver training pipelines, prediction pipelines, and trained models.

CI/CD allows us to experiment and deploy new ML models faster and with lower costs. This means we can iterate faster and produce better results within a limited time.

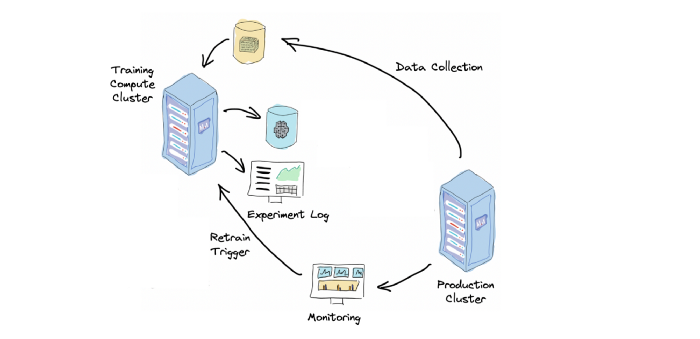
**Continuous Training**

Continuous Training (CT) is a new property unique to ML systems. It is concerned with automatically retraining and serving the models. ML models can get outdated regularly in real production scenarios. Examples of the situations where model retrain procedure is desirable:

* model performance is degraded (and we are capable of tracking it in production)
* out-of-distributions samples accumulate
* data/concept drift
* new training data is available, which can improve our current model performance.

In such situations, DS and ML engineering teams need to train the model on the updated training dataset, validate its performance, and deploy the new version. This can be problematic if these activities are costly or locked on specific team members. Continuous Training automates this process.

Continuous Training is the capability of a Machine Learning system to detect the need for retraining, retrain the model, validate it and sometimes deploy it automatically with little or no human intervention.



Continuous Training. Image provided by the author.

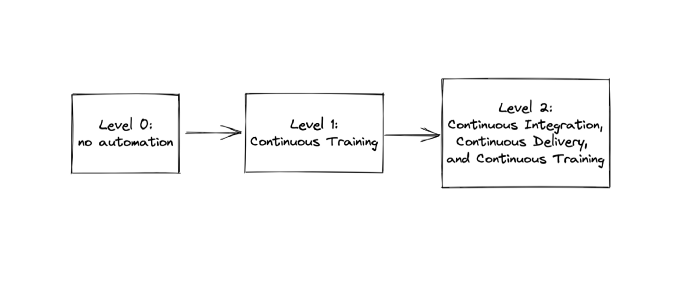
This makes a Machine Learning system more adaptive and, as a result, more robust.

CT usually requires the implementation of at least these components:

* Automated Data Collection: ETL of the new production data to update training and validation datasets.
* Automated Training Pipeline: end-to-end model training procedure that takes training data and produces trained model parameters.
* Automated Model Validation: this allows the ML system to automatically determine if the model is valid to be deployable.
* Model Store: the storage for trained model parameters and metadata to track their lineage and performance. Allows unique model identification and querying.
* Retrain Trigger: the trigger signal to start the retraining procedure. It can be a data drift alarm, a cron job, or new data availability.

Google Cloud MLOps Maturity framework advocates the implementation of these three capabilities in ML systems: Continuous Integration, Continuous Delivery, and Continuous Training. The level of their automation reflects the maturity of an ML system.

The framework distinguishes three levels of maturity:



Google Cloud MLOps Maturity Levels. Image provided by the author.

* **Level 0: no automation in building and deploying ML models.** This is the basic level where building and deploying ML models is entirely manual.
* **Level 1: Continuous Training.** This is the next level of maturity where we implement automated Continuous Training. This increases the robustness and adaptability of our ML System and allows us to retrain production models frequently based on the new data.
* **Level 2: Continuous Integration, Continuous Delivery, and Continuous Training.** The most advanced level of maturity. To reach it, we automate the delivery of the ML pipelines via CI/CD. This allows us to experiment with new model implementations much faster. Data Science and Software Engineering teams are aligned. The ML system is highly adaptive and robust.

These levels provide direction and high-level milestones for improving your ML systems. But to use this model, you need to treat it as a generic template. It needs to be filled and adjusted to your organization and ML system. You’ll need to develop a systematic practice out of it. Some DevOps tools can help you with that. Among them are Continuous Improvement, Toyota Kata, and Value Stream Mapping.

Let’s look into the next maturity model, which is very similar to Google Cloud MLOps Maturity Levels.

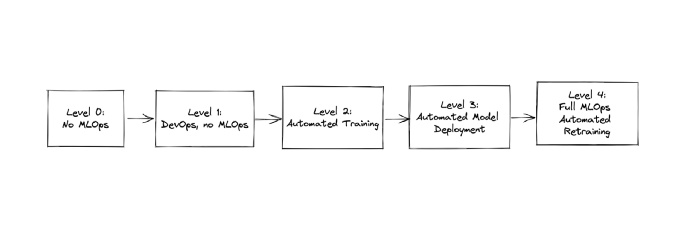
**Azure MLOps Maturity Model**

This framework is defined in the [following document](https://learn.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-maturity-model). It also focuses on adopting MLOps principles and practices to guide teams toward production-level Machine Learning systems.

It qualitatively assesses people, processes, and technology and emphasizes two aspects of your ML system:

* Collaboration between Software Engineers, Data Scientists, Data Engineers, and Machine Learning Engineers.
* Automation of an entire Machine Learning lifecycle.

It distinguishes five levels of maturity:



Azure MLOps Maturity Levels. Image provided by the author.

* **Level 0: No MLOps.** Data Science, ML/Data Engineering, and Software Engineerings teams are siloed. The workflows involve a lot of handovers and almost no automation.
* **Level 1: DevOps, no MLOps.** Software components of the application are delivered with automated CI/CD pipelines. Machine Learning models and pipelines are still delivered manually with handovers. The collaboration level between teams is low.
* **Level 2: Automated Training.** Data Scientists work together with ML engineers to convert experimentation scripts into reusable and reproducible training procedures. Experiment results are getting tracked.
* **Level 3: Automated Model Deployment.** This is the level when the deployment of ML models is automated via CI/CD pipeline. Data Science, ML Engineering, and Software Engineering teams are aligned.
* **Level 4: Full MLOps Automated Retraining.** The level of Continuous Training.

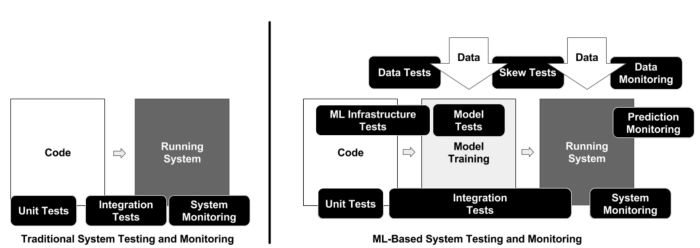
Two models we’ve reviewed so far have one downside. They are generic and not directly actionable. They describe the general desired states and patterns for the ML system but don’t provide you with concrete, actionable recipes for reaching them. And most importantly, they are not quantifiable.

Let’s look at the last model in this article.

**ML Test Score**

Google team introduced this framework in their paper [“The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction” (E.Breck et al., 2017)](https://research.google/pubs/pub46555/). Its goal is to guide teams on how to identify, prioritize and fight the technical debt in ML systems. The systems can be of any size and maturity level: from a student thesis research project to a highly-available scalable and low-latency production system.

The primary weapon to fight a technical debt in software systems (and ML systems in particular) is testing and monitoring. ML Systems require much more extensive use of these two practices because of their inherent higher complexity. The picture from the paper below shows this.



Testing of traditional Software systems vs. ML systems. Image source: E.Breck et al. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction.

But the lack of a systematic approach leads to two problems. Tests rarely have enough priority to be implemented because there are always “more important things to do.” And when they do, it’s unclear what to test to get the most benefit.

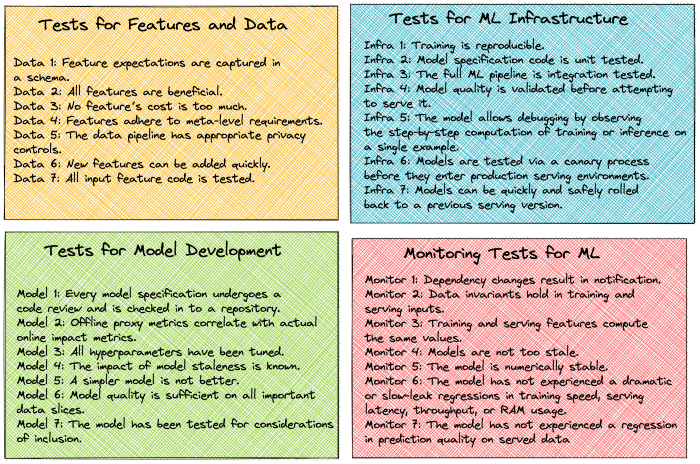
So to overcome these two problems, the framework's authors made two main design decisions. To be of use, the framework must be:

* **Actionable**: it must provide a concrete set of recipes to formulate a backlog of team activities. The framework consists of concrete, actionable tests that can be converted to tickets or epics in a team’s backlog.
* **Quantifiable**: it must provide a single scoring metric to allow teams to prioritize these activities between each other and over other tasks. The framework provides an algorithm to calculate the such metric.

**ML System Assertions**

ML Test Score consists of 28 actionable tests that are divided into 4 categories:

* Tests for features and data
* Tests for model development
* Tests for ML infrastructure
* Monitoring tests for ML



ML System Assertions. Image provided by the author.

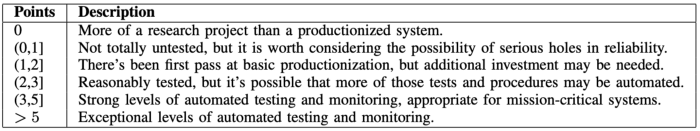
Each test acts as an assertion to be checked. Its violation indicates a possible failure in the ML system. Robust ML systems must check these assertions frequently. I encourage you to read the original paper for a detailed description of each test.

**ML System Scoring System**

Besides the tests, the framework provides a scoring system with a single scalar metric. The final test score is computed as follows:

* For each test from the above, half a point is awarded for executing the  
  test manually, with the results documented and distributed.
* A full point is awarded if there is a system in place to  
  run that test automatically on a repeated basis.
* Sum the score for each of the 4 sections individually.
* The final ML Test Score is computed by taking the  
  minimum of the scores aggregated for each of the 4 sections.

There are 6 levels of maturity of an ML system based on the value of the test score:



ML Test Score. E.Breck et al. Image source: The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction.

You can assess your current maturity level based on this metric. Then you can set the target KPI for your MLOPs / ML Engineering team for the next milestone period. The framework allows you to identify the gap and plan the necessary activities to reach the planned KPI.

Let’s say you’ve identified your current ML maturity level as below 1 based on this metric. And you’ve set the goal to improve it to 2 by the end of the current quarter. You’ve figured out that Monitoring Tests for ML is the weakest area contributing to poor ML Test Score performance. So you have a backlog of possible monitoring tests to implement, which you can prioritize based on the concrete specifics of your project.

Not only do you have a concrete algorithm for improving your ML maturity, but you have a concrete KPI to defend such activities when it comes to setting priorities. And you have a clear message to the management asking about your contributions and deliverables.

**Conclusion**

The above frameworks can help your ML Engineering team to establish a systematic continuous improvement practice in increasing the maturity of your ML systems.

You can use them as a guide to:

* Identify current gaps in ML maturity.
* Estimate the scope of the work for new activities to cover these gaps.
* Establish realistic success criteria for them.
* Identify deliverables you’ll hand over as a result of such activities.