



MACHINE LEARNING

Ecosystem 101

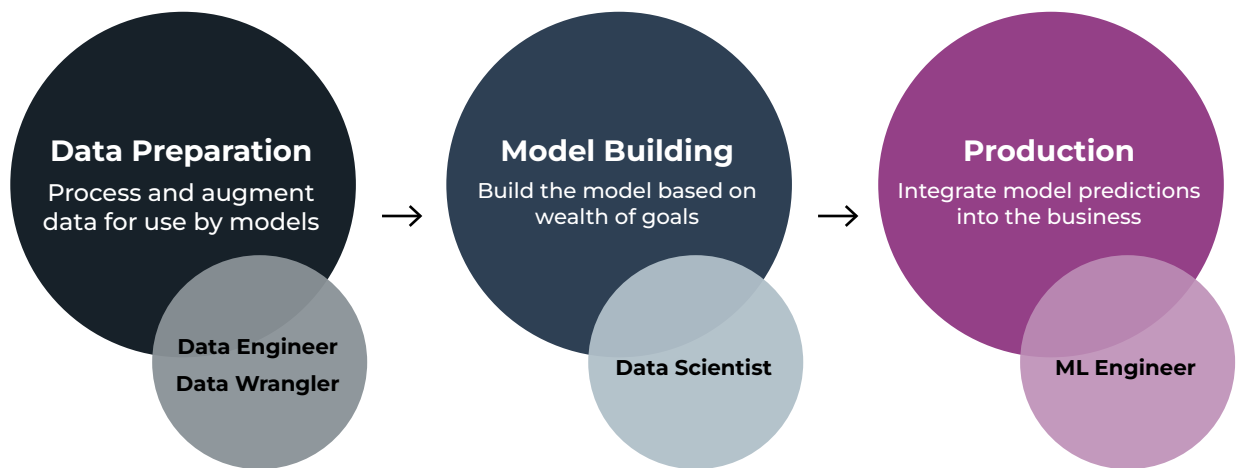
Table of contents

Introduction	1
STAGE 1: Data Preparation	2
STAGE 2: Model Building & Development Tools	7
STAGE 3: Model Validation	12
STAGE 4: Model Serving	18
STAGE 5: Observability	25

Introduction

Artificial Intelligence (AI) and Machine Learning (ML) are being adopted by businesses in almost every industry. Many businesses are looking towards ML Infrastructure platforms to propel their movement of leveraging AI in their business. Understanding the various platforms and offerings can be a challenge. The ML Infrastructure space is crowded, confusing, and complex. There are a number of platforms and tools spanning a variety of functions across the model building workflow.

To understand the ecosystem, we broadly break up the machine learning workflow into three stages — data preparation, model building, and production. Understanding what the goals and challenges of each stage of the workflow can help make an informed decision on what ML Infrastructure platforms out there are best suited for your business's needs.



Each of these broad stages of the Machine Learning workflow (Data Preparation, Model Building and Production) have a number of vertical functions. Some of these functions are part of a larger end-to-end platform, while some functions are the main focus of some platforms.

Since models are built and learned from data, the first step of building a model is data preparation — the process of extracting inputs to the model from data. There are a number of tools to help data scientists source data, transform data, and add labels to datasets. In this blog post, we will dive deep into understanding what are the goals of data preparation, challenges organizations face in this stage of the ML workflow, and when data scientists decide it is time to move onto the next stage of the workflow.



STAGE 01

What is Data Preparation?

Ask any data scientist and they will tell you A LOT of their time is spent in data preparation. The data preparation phase of the pipeline is used to turn raw data into model input features used to train the model. Features are transformations on the cleaned data that provide the actual model inputs.

In the early stages of the pipeline, raw data is sourced across different data stores and lakes in an organization. The next stage involves data processing to clean, transform and extract features to generate consistent inputs in the feature selection stage. Large tech companies at the forefront of using ML Infrastructure (Google, Facebook, Uber, etc) will typically have central feature storage, so many teams can extract value without duplicate work.

The data preparation stage involves a number of steps: sourcing data, ensuring completeness, adding labels, and data transformations to generate features.



Sourcing Data

Sourcing data is the first step and often the first challenge. Data can live in various data stores, with different access permissions, and can be littered with personally identifiable information (PII).

The first step in data preparation involves sourcing data from the right places and consolidating data from different data lakes within an organization. This can be difficult if the model's inputs, predictions, and actuals are received at

different time periods and stored in separate data stores. Setting a common prediction or transaction ID can help tie predictions with their actuals.

This stage can often involve data management, data governance and legal to determine what data sources are available to use. The roles working in this stage usually involve the data engineer, data scientist, legal, and IT.

Example ML Infrastructure Companies in Data Storage:



Completeness

Once the data is sourced, there are a series of checks on completeness needed to determine if the data collected can be turned into meaningful features. First, it is important to understand the length of historical data available to be used. This helps understand if the model builder has enough data for training purposes (a year's worth of data, etc). Having data that has seasonal cycles and identified anomalies can help the model build resiliency.

Data completeness can also include checking if the data has proper labels. Many companies have problems with the raw data in terms of cleanliness. There can be multiple labels that mean the same thing. There will be some data that is unlabeled or mislabeled. A number of vendors offer Data Labeling services that employ a mix of technology and people to add labels to data and clean up issues.

Example ML Infrastructure Companies in Data Labeling:



It is also important to have some check on whether the data seen is a representative distribution. Was the data collected over an unusual period of time? This is a tougher question because it is specific to the business and data will continue to change over time.

Data Processing

Once the data is collected and there is enough data across time with the proper labels, there can be a series of data transforms to go from raw data to features the model can understand. This stage is specific to the types of data that the business is using. For categorical values, it is common practice to use one-hot encoding. For numeric values, there can be some form for normalization based on the distribution of the data. A key part of this process is to understand your data, including data distributions.

Data processing can also involve additional data cleaning and adding data quality checks. Since models depend on the data they are training on, it is important to ensure clean data through removing duplicated events, indexing issues, and other data quality issues.

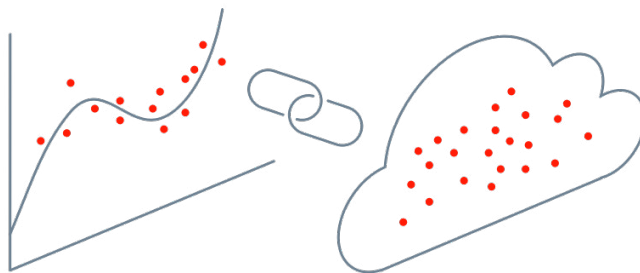


Image by Tecton.

A set of data wrangling companies allow data scientists, business analysts, and data engineers to define rules for transformations to clean and prepare the data. These companies can range from no code, low code, to developer focused platforms.

Lastly, there are ongoing data quality checks that are done on training data to make sure what is clean today will be clean tomorrow.

Data preparation is integral to the model's performance. There are a lot of challenges to getting complete and clean data. With all of the work that goes into building a training dataset from data sourcing to all of the data transformations, it can be difficult to track all of the versioned data transformations that can impact model performance. As an organization grows, a feature store with common data transformations can reduce duplicative work and compute costs.

ML Infrastructure Companies in Data Wrangling:



ML Infrastructure Companies in Data Processing:



ML Infrastructure Companies in Data Versioning, Feature Storage & Feature Extraction:



What Happens After Data Preparation

Once data scientists have the data ready, In some cases, the handoff between data preparation and model building is structured with a data file or feature store with processed data. In other cases, the handoff is fluid. In larger organizations the Data Engineering team is responsible for getting the data into a format that the data scientists can use for model building.

In many managed notebooks such as Databricks Managed Notebooks, Cloudera Data Science Workbench, Domino Data Labs Notebooks, the data preparation workflow is not separate from the model building. Feature selection is dependent on data so that function begins to blur the line between data preparation and model building.

ML Infrastructure Companies in Notebook Management:





STAGE 02

Model Building & Development Tools

The first step of model building begins with understanding the business needs. What business needs is the model addressing? This step begins much further at the planning and ideation phase of the ML workflow. During this phase, similar to the software development lifecycle, data scientists gather requirements, consider feasibility, and create a plan for data preparation, model building, and production. In this stage, they use the data to explore various model building experiments they had considered during their planning phase.

Feature Exploration and Selection

As part of this experimental process, data scientists explore various data input options to select features. Feature selection is the process of finding the feature inputs for machine learning models. For a new model, this can be a lengthy process of understanding the data inputs available, the importance of the input, and the relationships between different feature candidates. There are a number of decisions that can be made here for more interpretable models, shorter training times, cost of acquiring features, and reducing overfitting. Figuring out the right features is a constant iterative process.

ML Infrastructure companies in Feature Extraction:

 alteryx

 Paxata

Model Management

There are a number of modeling approaches that a data scientist can try. Some types of models are better for certain tasks than others (ex — tree based models are more interpretable). As part of the ideation phase, it will be evident if the model is supervised, unsupervised, classification, regression, etc. However, deciding what type of modeling approaches, what hyperparameters, and what features is dependent on experimentation. Some AutoML platforms will try a number of different models with various parameters and this can

be helpful to establish a baseline approach. Even done manually, exploring various options can provide the model builder with insights on model interpretability.

Experiment Tracking

While there are a number of advantages and tradeoffs amongst the various types of models, in general, this phase involves a number of experiments. There are a number of platforms to track these experiments, modeling dependencies, and model storage. These functions are broadly categorized as model management. Some platforms primarily focus on experiment tracking. Other companies that have training and/or serving components have model management components for comparing the performance of various models, tracking training/test datasets, tuning and optimizing hyperparameters, storing evaluation metrics, and enabling detailed lineage and version control. Similar to Github for software, these model management platforms should enable version control, historical lineage, and reproducibility.

A tradeoff between these various model management platforms is the cost of integration. Some more lightweight platforms only offer experiment tracking, but can integrate easily with the current environment and be imported into data science notebooks. Others require some more heavy lifting integration and require model builders to move to their platform so there is centralized model management.

In this phase of the machine learning workflow, data scientists usually spend their time building models in notebooks, training models, storing the model weights in a model store, and then evaluating the results of the model on a validation set. There are a number of platforms that exist to provide compute resources for training. There are also a number of storage options for models depending on how teams want to store the model object.

ML Infrastructure AutoML:



ML Infrastructure companies in Experiment Tracking:



ML Infrastructure companies in Model Management:



ML Infrastructure companies in HyperParameter Opt.:



Model Evaluation

Once an experimental model has been trained on a training data set with the selected features, the model is evaluated on a test set. This evaluation phase involves the data scientist trying to understand the model's performance and areas for improvement. Some more advanced ML teams will have an automated backtesting framework for them to evaluate model performance on historical data.

Each experiment tries to beat the baseline model's performance and considers the tradeoffs in compute costs, interpretability, and ability to generalize. In some more regulated industries, this evaluation process can also encompass compliance and auditing by external reviewers to ensure the model's reproducibility, performance, and requirements.

ML Infrastructure Model Evaluation:



ML Infrastructure Pre-Launch Validation:



One Platform to Rule Them All

A number of companies that center on AutoML or model building, pitch a single platform for everything. They are vying to be the single AI platform an enterprise uses across DataPrep, Model Building and Production. These companies include DataRobot, H2O, SageMaker and a number of others.

This set splits into a low-code versus developer centric solutions. Datarobot seems to be focused on the no-code/low code option that allows BI or Finance teams to take up DataScience projects. This is in contrast with SageMaker and H2O which seem to cater to either data scientists or developer first teams that are the more common data science organizations today. The markets in both cases are large and can co-exist but it's worth noting that not all of the ML Infrastructure companies are selling to the same people or teams.

A number of the more recent entrants in the space can be thought as best of breed solutions for a specific part of the ML Infrastructure food chain. The best analog would be the software engineering space, where your software solutions GitHub, IDE, production monitoring are not all the same end-to-end system. There are reasons why they are different pieces of software; they provide very different functions with clear differentiation.

Best of Breed Platforms

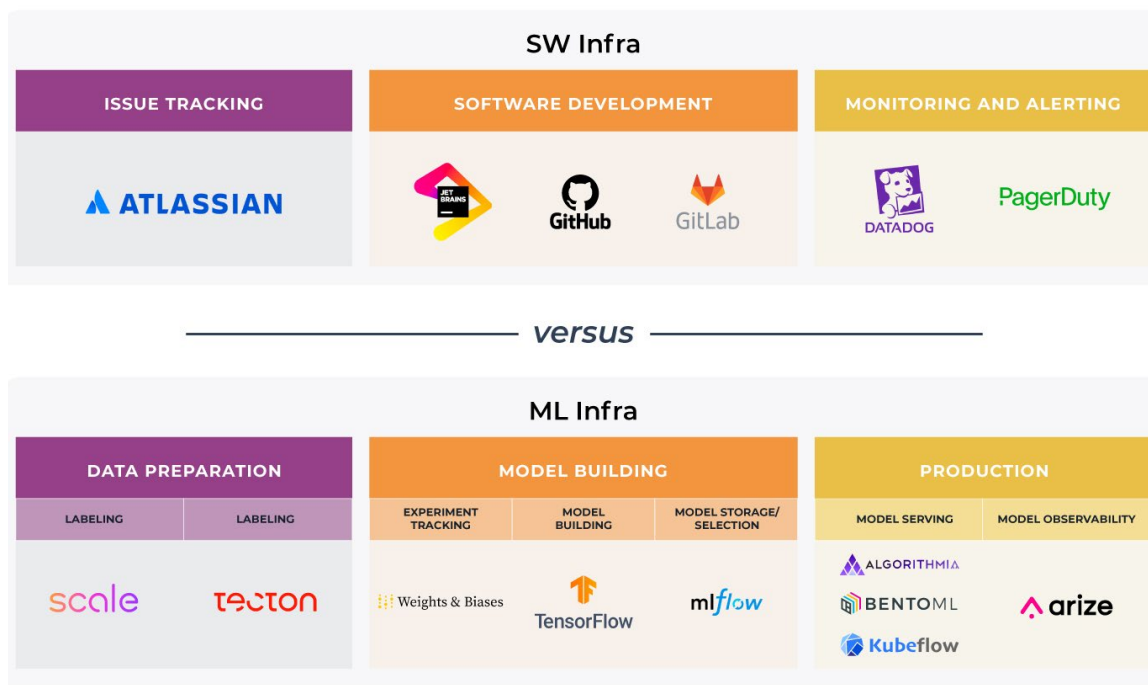



Image by author. Best of Breeds Platforms

The background of the slide is an abstract composition of numerous thin, parallel lines. These lines are oriented diagonally, running from the top-left towards the bottom-right. They vary in brightness, with some appearing as bright white or light pink streaks and others as darker, muted pink or greyish lines. The overall effect is a sense of depth and movement, reminiscent of a stylized architectural structure or a data visualization.

STAGE 03

Model Validation



Launching a model into production can feel like crossing the finish line after a marathon. In most real-world environments, it can take a long time to get the data ready, model trained, and finally done with the research process. Then there's the arduous process of putting the model into production which can involve complex deployment pipelines and serving infrastructures. The final stage of standing a model up in production involves checking the model is ready for production, packaging the model for deployment, deploying the model to a serving environment and monitoring the model & data in production.

The production environment is by far, the most important, and surprisingly, least discussed part of the Model Lifecycle. This is where the model touches the business. It's where the decisions the model makes actually improve outcomes or cause issues for customers. Training environments, where data scientists spend most of their time and thought, consist of just a sample of what the model will see in the real world.

Research to Production

One unique challenge in the operationalizing of Machine Learning is the movement from a research environment to a true production engineering environment. A Jupyter Notebook, the most common home of model development, is ostensibly a research environment. In a well controlled software development environment an engineer has version control, test coverage analysis, integration testing, tests that run at code check-ins, code reviews, and reproducibility. While there are many solutions trying to bring pieces of the software engineering workflow to Jupyter notebooks, these notebooks are first and foremost a research environment designed for rapid and flexible experimentation. This coupled with the fact that not all data scientists are software engineers by training, but have backgrounds that span many fields such as chemical engineering, physicists, and statisticians — you have what is one of the core problems in production ML:

The core challenge in Production ML is uplifting a model from a research environment to a software engineering environment while still delivering the results of research.

In this blog post, we will highlight core areas that are needed to uplift research into production with consistency, reproducibility, and observability that we expect of software engineering.

Model Validation

Note: Model validation is **NOT** to be confused with the validation data set.

Quick Recap on Datasets: Models are built and evaluated using multiple datasets. The **training data set** is used to fit the parameters of the model. The **validation data set** is used to evaluate the model while tuning hyperparameters. The **test set** is used to evaluate the unbiased performance of the *final* model by presenting a dataset that wasn't used to tune hyperparameters or used in any way in training.

What is Model Validation?

You're a data scientist and you've built a well-performing model on your test set that addresses your business goals. *Now, how do you validate that your model will work in a production environment?*

Model validation is critical to delivering models that work in production. Models are not linear code. They are built from historical training data and deployed in complex systems that rely on real-time input data. **The goal of model validation is to test model assumptions and demonstrate how well a model is likely to work under a large set of different environments. These model validation results should be saved and referenced to compare model performance when deployed in production environments.**

Models can be deployed to production environments in a variety of different ways, there are a number of places where translation from research to production can introduce errors. In some cases, migrating a model from research to production can *literally* involve translating a Python based Jupyter notebook to Java production code. While we will cover in depth on model storage, deployment and serving in the next section, it is important to note that some operationalization approaches insert additional risk that research results do not match production results. Platforms such as Algorithmia, SageMaker, Databricks and Anyscale are building platforms that are trying to allow research code to directly move to production without rewriting code.

In the Software Development Lifecycle, unit testing, integration testing, benchmarking, build checks, etc help ensure that the software is considered

with different inputs and validated before deploying into production. In the Model Development Lifecycle, model validation is a set of common & reproducible tests that are run prior to the model going into production.

Current State of Model Validation in Industry

Model validation is varied across machine learning teams in the industry today. In less regulated use cases/industries or less mature data science organizations, the model validation process involves just the data scientist who built the model. The data scientist might submit a code review for their model built on a Jupyter notebook to the broader team. Another data scientist on the team might catch any modeling issues. Additionally model testing might consist of a very simple set of hold out tests that are part of the model development process.

More mature machine learning teams have built out a wealth of tests that run both at code check-in and prior to model deployment. These tests might include feature checks, data quality checks, model performance by slice, model stress tests and backtesting. In the case of backtesting, the production ready model is fed prior historical production data, ideally testing the model on a large set of unseen data points.

In regulated industries such as fintech and banking, the validation process can be very involved and can be *even longer than* the actual model building process. There are separate teams for model risk management focused on assessing the risk of the model and its outputs. Model Validation is a separate team whose job it is to break the model. It's an internal auditing function that is designed to stress and find situations where the model breaks. A parallel to the software engineering world would be the QA team and code review process.

Model Validation Checks

Regardless of the industry, there are certain checks to do prior to deploying the model in production. These checks include (but are not limited to):

- Model evaluation tests (Accuracy, RMSE, etc...) both overall and by slice
- Prediction distribution checks to compare model output vs previous versions
- Feature distribution checks to compare highly important features to previous tests

- Feature importance analysis to compare changes in features models are using for decisions
- Sensitivity analysis to random & extreme input noise
- Model Stress Testing
- Bias & Discrimination
- Labeling Error and Feature Quality Checks
- Data Leakage Checks (includes Time Travel)
- Over-fitting & Under-fitting Checks
- Backtesting on historical data to compare and benchmark performance
- Feature pipeline tests that ensure no feature broke between research and production

ML Infrastructure tools that are focused on model validation provide an ability to perform these checks or analyze data from checks — in a repeatable, and reproducible fashion. They enable an organization to reduce the time to operationalize their models and deliver models with the same confidence they deliver software.

Example ML Infrastructure Platforms that enable Model Validation:



Other companies

Model Compliance and Audit

In the most regulated industries, there can be an additional compliance and audit stage where models are reviewed by internal auditors or even external auditors. ML Infrastructure tools that are focused on compliance and audit teams often focus on maintaining a model inventory, model approval and documentation surrounding the model. They work with model governance to enforce policies on who has access to what models, what tier of validation a model has to go through, and aligning incentives across the organization.

Example ML Infrastructure:



Continuous Delivery

In organizations that are setup for continuous delivery and continuous integration, a subset of the validation checks above are run when:

- Code is checked in — Continuous Integration
- A new model is ready for production — Continuous Delivery

The continuous integration tests can be run as code is checked in and are typically structured more as unit tests. A larger set of validation tests that are broader and may include backtesting are typically run when the model is ready for production. The continuous delivery validation of a model becomes even more important as teams are trying to continually retrain or auto-retrain based on model drift metrics.

A number of tools such as ML Flow enable the model management workflow that integrates with CI/CD tests and records artifacts. These tools integrate into Github to enable pulling/storing the correct model and storing results initiated by GitHub actions.


Example ML Infrastructure Platforms that Enable Continuous Delivery:





STAGE 04

Model Serving



Once the model has been trained, built, and validated, it is finally time to deploy and serve the model! In this last step of ML, all of the work of the previous steps are finally put to use by a data-driven model

The first decision that teams need to make is whether they should even build a model server at all. Most models deployed in the last five years were home-built serving approaches. In recent years, however, companies working with ML models have moved away from building everything from scratch. In fact, we predict that the approach of building everything from scratch will change drastically going forward, given the number of model servers coming to market.

Model serving options for models typically fit into a couple different types:

- **Internally built executable (PKL File/Java)** — containerized & non-containerized
- **Cloud ML Provider** — Amazon SageMaker, Azure ML, Google AI
- **Batch or Stream: Hosted & On-Prem** — Algorithmia, Spark/Databricks, Paperspace
- **Open Source** — TensorFlow Serving, BentoML, Kubeflow, Seldon, Anyscale, etc.

Which of these is the right choice for a given team? There are a number of considerations in the decision of model serving options. Here are a few questions teams ask themselves to determine which is the best ML option for them:

Key Questions to Consider

1. What are the data security requirements of the organization?

On-premise ML solutions may be required for organizations with strict data security requirements. Some good choices are BentoML, Algorithmia, Seldon, Tensorflow, Kubeflow, or home- built proprietary solutions. Some providers such as Algorithmia have security specific feature sets detailed below. Cloud solutions may be a better choice for those organizations needing less security but more remote access/virtualization.

2. Does the team want managed or unmanaged solutions for model serving?

A managed solution such as Algorithmia, SageMaker, Google ML, Azure, and Paperspace are a good idea for companies with a low IT presence. An un-managed solution such as BentoML, Kubeflow, Seldon, Tensorflow Serving, or Anyscale may be better for more technical organizations.

3. Is every team in the organization going to use the same deployment option?

Even if one team chooses a serving option, rarely is the whole organization using the same serving approach. Having common model management platform like ML-Flow can still help bridge the gap.

4. What does the final model look like? Is there an already established interface?

If a model is already deployed, it might not make sense to rip out the model serving system and replace it with a new model server. How easy it would be to replace the already-deployed model might depend on the model server that was chosen and its integrations with other systems, APIs, and Feature Pipelines.

5. Where does the model executable live? (for example, ML Flow or S3 Bucket)

Easy integration to ML-Flow or model storage systems is an important consideration.

6. Is GPU inference needed?

Predictions using GPU servers based on performance requirements will likely drive you to either cloud providers or Algorithmia for on-premises.

7. Are there separate feature generation pipelines or are they integrated into the model server?

Depending on where your feature pipelines are deployed, say, Amazon Web Services (AWS), that might direct you toward using SageMaker. This is probably one of more common reasons to use SageMaker as data is already deployed in AWS.

Deployment Details

The format of the model can vary, based on the frameworks used to build the model, across organizations and projects. Some example formats include a pickle image of weights/parameters for the classifier, a Tensorflow SavedModel object, a Pytorch model, Keras model, XGBoost model, Apache TVM, MXNet, ONNX Runtime, etc.

Implementation

There are many ways that ML models can be implemented. These models can be integrated into a larger system's codebase, deployed as a microservice, or even live on a device. If the model is code that is integrated into a larger system, the interface into the model is simply a function call. If the model is in its own service/executable or server, it can be seen as a service. These services have well defined APIs or interfaces to pass inputs to the models and get responses. The model servers described above take the trained model artifact generated in the above formats and allow you to deploy it to a containerized model server that generates well defined APIs.

Containerization

A modern server approach is to containerize the model executable so there is a common interface into models and a common way of standing them up. The model is pulled from the model management system (such as ML-Flow) into a container when it is deployed. There are many ways to accomplish this, either building a custom container for your company, using open source solutions like BentoML, KubeFlow or Seldon AI, or using the common cloud provider tools such as Algorithmia, SageMaker, Azure or Google AI.

Real-Time or Batch Model

Another important deployment consideration to make is whether to have a real-time/online model or a batch model. Online models are used when predictions need to be immediate and take in real-time application input. If this isn't a requirement, batch inferences can be appropriate. A number of serving platforms allow you to build a single model and have different deployment options (Batch or Real Time) to support both types of serving regimes.

Things to Look for in Model Servers

Easy Scale Out:

As the prediction volume grows of an application, the initial approach of having a single server supporting predictions can get easily overwhelmed. The ability to simply add servers to a prediction service, without the need to re-architect or generate a large amount of additional model operations work, is one of the more useful features of a model server.

Canary A/B Framework:

A canary A/B framework allows developers to roll out software to a small subset of users to perform A/B testing to figure out which aspects of the software are most useful and provide the best functionality for users. Once deployed, some teams run a A/B (canary) model side-by-side with the production model, initially predicting on only a small subset of traffic for the

new model. This is done as a simple test before deploying the new model across the full volume of predictions. A lot of teams we talk to have home-built their own A/B testing framework. That said, some of the model server solutions also support easy A/B deployments out of the box, for example, to choose the % of traffic to the B model with the click of a button.

Ensemble Support:

The ability to co-locate multiple models in the same server or easily connect the prediction (inference) flow between models might be an important consideration. Most of the time, the model response will be consumed by the end application, but as systems get more complex, some models' outputs are inputs to another model. In cases of fast prediction response, co-locating models can be desired.

Fall Back Support:

As you deploy a new model into production, you might find that performance drops drastically. The ability to have a different model, maybe a previous version or a very simple model during periods of degraded performance, can be very helpful in situations like this.

Security:

If security is extremely important to the organization, some platforms have very well thought out security feature sets. These span a set of security requirements focused on: access rights, application security, network security, and memory security. The model in production needs to grab data/inputs from somewhere in the system, and it needs to generate predictions/outputs used by other systems. The application who has access to the predictions might not have rights to the input data. Also, if the application is using Python packages in a Kubernetes-hosted model, many companies want to make certain that those packages are not public packages. Lastly, if you are running in a shared memory environment like a GPU, you will need to take stock of what data protections you have in place around memory encryption and access. Some platforms, such as Algorithmia, have more developed security feature sets that provide solutions for a myriad of situations.

Feature Pipeline Support:

In containerized solutions, input-to-feature transformations may reside in the container itself, or may have separate feature transformation pipelines. The larger the infrastructure, the more likely the input-to-feature transformation is a pipeline or feature store system with inputs to the container being pre-processed.

In the serving layer, there are also some new platforms such as Tecton AI which are focused on feature serving. A global feature store allows teams to

easily deploy the feature pipeline directly into production environments — minimizing feature pipeline mistakes, and allowing teams to take advantage of cross-company feature builds.

Monitoring:

Some model servers support basic monitoring solutions. Some servers support monitoring for serving infrastructure, memory usage, and other operational aspects. Our view is that this type of raw model ops monitoring and visualization is important to model scale out, but is not observability. We are obviously biased, but have an opinion that true model observability is really a separate platform.

Example ML Infrastructure Platforms for Deployment and Serving include:

H₂O



Tecton



ALGORITHMIA



DataRobot

Model Observability vs Model Monitoring

It may seem like anyone can do monitoring — green lights are good, and red lights are bad. You can set alerts, and if a value falls below a certain level, this triggers sending an email to the staff.

Yet, if that were the case, Amazon Cloud Watch would have killed Datadog. The issue here is — what do you do when you get that alert?

Our opinion is that the difference between a monitoring solution and an Observability platform is the ability to troubleshoot and bottom out issues seamlessly. In the ML ecosystem, these problems surface as issues linking AI research to engineering. Is the platform designed from the bottom up to troubleshoot problems, or was an alerting system tacked on to some pre-existing graphs? Troubleshooting models + data in the real world is a large and complex space. That's why Observability platforms are designed from the ground up to help research and engineering teams jointly tackle these problems.

Why the model server is not a great spot for Observability:

The model server does not have the right data points to link the complex layers needed to analyze models. The model server is missing essential data such as training data, test runs, pre-one hot encoded feature data, truth/label events, and much more. In the case of feature data, for a number of larger models we have worked on, the insertion point into the data pipeline for troubleshooting is a very different technology than the model server. Lastly, many organizations have as many model serving approaches as they have models in production, and it is very unlikely they will move to a single server to rule them all. What happens when you have a mix of models served that feed each other data but you want a cohesive picture?

It's the same in software infrastructure; your infrastructure observability solutions are not tied to the infrastructure itself.

STAGE 05

Observability



As a machine learning community we have made huge strides in building models that empower computers to do amazing feats of intelligence. Now that these models are out in the world making decisions, how do we make sure that these technologies are actually working, that we deliver them continuously with quality, that we actively improve them and ensure research results match production?

As most data scientists quickly realize a Jupyter Notebook is not the real world. Model Observability is the key that bridges the gap. Model Observability is the foundational platform that empowers teams to continually deliver and improve results from the lab to production.

In this blog series, we will be discussing the topic of model observability. We will dig into the problems models run into, in the real world. We will also take a look at tools that help one understand how models are behaving.

Is Model Observability just a fancy word for ML monitoring?

Model observability begins with the process of collecting model evaluations in environments such as training, validation, and production, then tying them together with analytics that allows one to connect these points to solve ML Engineering problems. These inferences are stored in a model evaluation store (credit to Josh Tobin for this term) which hosts the raw inference data. *An evaluation store holds the response of the model, a signature of the model decisions, to every piece of input data for every model version, in every environment.*

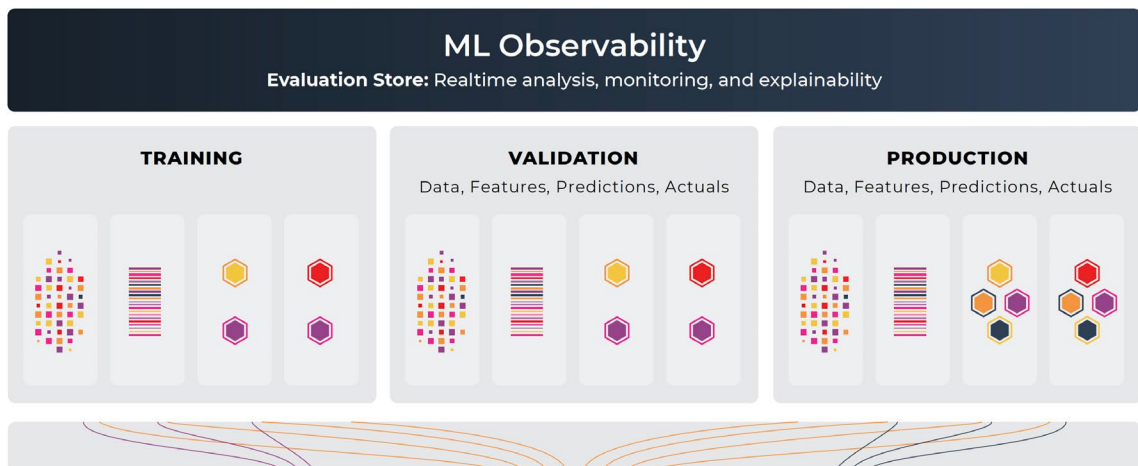


Image by author. Making complex simple

The ML Observability platform allows teams to analyze model degradation and to root cause any issues that arise. This ability to diagnose the root cause of a model's issues, by connecting points across validation and production, is what differentiates model observability from traditional model monitoring.

While model monitoring consists of setting up alerts on key model performance metrics such as accuracy, or drift, model observability implies a higher objective of getting to the bottom of any regressions in performance or anomalous behavior. We are interested in the why. Monitoring is interested in only aggregates and alerts. Observability is interested in what we can infer from the model's predictions, explainability insights, the production feature data, and the training data, to understand the cause behind model actions and build workflows to improve.

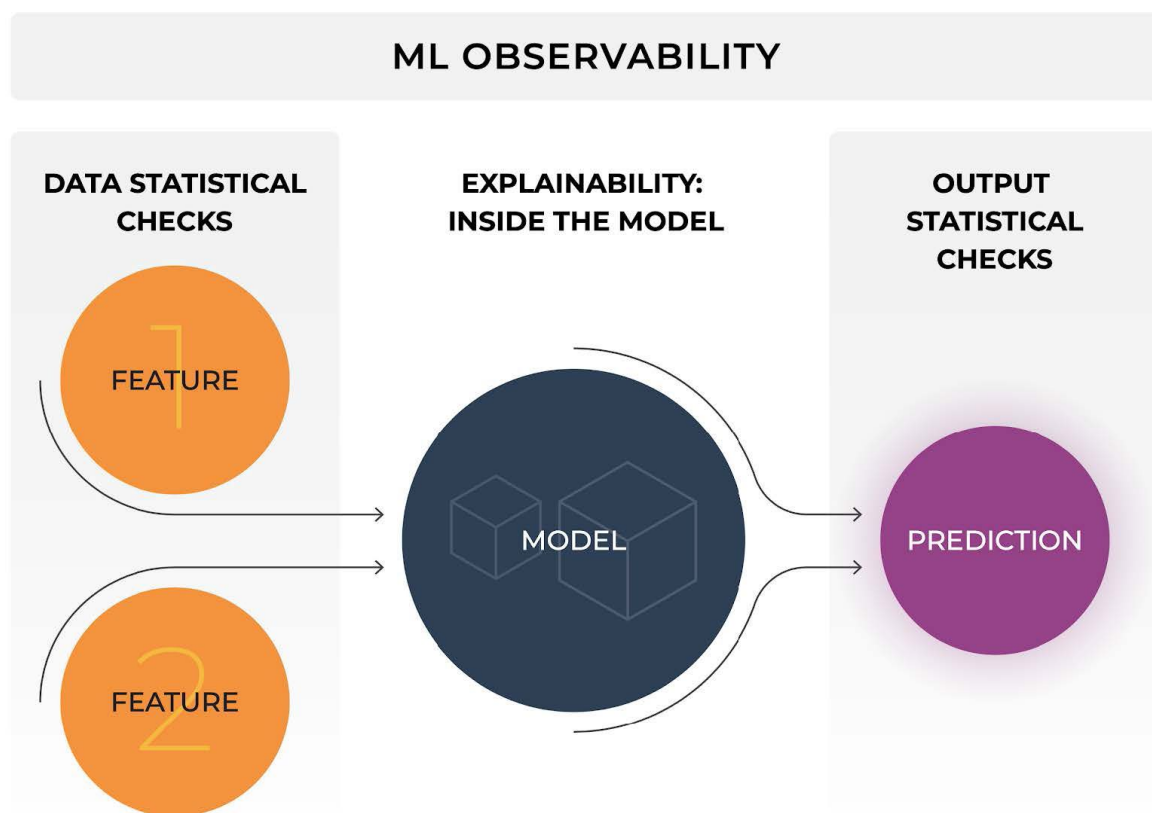


Image by author. ML Observability Flow

ML Observability backed by an Evaluation Store:

- Move seamlessly between production, training, and validation dataset environments
- Natively support model evaluation analysis by environment
- Designed to analyze performance Facets/Slices of predictions
- Explainability attribution designed for troubleshooting and regulatory analysis
- Performance analysis with ground truth — Accuracy, F1, MAE
- Proxy performance without ground truth — prediction drift
- Distribution drift analysis between data sets and environments
- Designed to answer the why behind performance changes
- Integrated validation
- Architected to iterate and improve

Why ML Observability matters

Our team has personally worked on models deployed in pricing, forecasting, marketing, credit and delivery time estimates, to name a few. In all cases there is a common story line; we would build a model, deploy it and it would work great in one city and not in another. It would work great across a set of neighborhoods/customers/types of products and poorly across others. It would be great in the average case but horrible on the tail end of predictions. Or it would work great on initial launch and then the model would slowly degrade. Other times you would have an instantaneous change caused by an upstream data pipeline mistake, that would slowly poison the training data. In all these cases it was clear there was a missing foundational piece of ML infrastructure to help make models work as teams deployed them to production.

In another common use of Machine Learning, such as fraud model analysis, understanding the why behind performance changes is extremely important. In fraud there is a constant landscape of changing behavior based on adversarial actions of bad actors. These bad actors create pockets of performance degradation that might throw a red or green light in a monitoring system. But the causal connection, “the why”, is typically the insight for stopping the fraud scheme.

In companies that build a model per customer, trained on customer specific data, every model is trained differently. Teams can be inundated with models showing data/model drift but it can be hard to show if that drift is a problem.

How do you bottom out issues at scale and how does your ML team scale as customers grow? Teams want to quickly know where issues are occurring at scale, compare production swiftly to validation and clear the issues quickly with confidence.

Without the tools to reason about mistakes a model is making in the wild, teams are investing a massive amount of money in the data science laboratory but essentially flying blind in the real world.

Contact Us

If this whitepaper caught your attention and you're eager to learn more, follow us on [Twitter](#) and [Medium](#)!

If you'd like to hear more about what we're doing at Arize AI, reach out to us at contacts@arize.com.

Join our rockstar engineering crew to make models successful in production, reach out to us at jobs@arize.com.

Contact us

Join us

